TESE DE DOUTORADO Nº 416

CONTROL OF DISCRETE EVENT SYSTEMS SUBJECT TO CYBER ATTACKS

Michel Rodrigo das Chagas Alves

DATA DA DEFESA: 09/11/2022

MICHEL RODRIGO DAS CHAGAS ALVES

CONTROLE DE SISTEMAS A EVENTOS DISCRETOS SUJEITOS A CYBER ATTACKS

Versão Final

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Engenharia Elétrica.

Orientador: Profa. Dra. Patrícia Nascimento Pena & Profa. Dra. Karen Rudie

Belo Horizonte

Maio de 2023

MICHEL RODRIGO DAS CHAGAS ALVES

CONTROL OF DISCRETE-EVENT SYSTEMS SUBJECT TO CYBER ATTACKS

Final Version

Thesis presented to the Graduate Program in Electrical Engineering of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

Advisor: Profa. Dra. Patrícia Nascimento Pena & Profa. Dra. Karen Rudie

Belo Horizonte

May 2023

Alves, Michel Rodrigo das Chagas.
A474c Controle de sistemas a eventos discretos sujeitos a *cyber attacks* [recurso eletrônico] / Michel Rodrigo das Chagas Alves. – 2023.
1 recurso online (145 f. : il., color.) : pdf.

Orientadora: Patrícia Nascimento Pena.
Coorientadora: Karen Rudie.

Tese (doutorado) – Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: f. 135-145.

Bibliografia: f. 127-134.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica – Teses. 2. Tecnologia da informação – Teses. 3. Informática – Teses. 4. Internet – Sistemas de segurança – Teses. 5. Software – Proteção – Teses. 6. Cybercrimes – Teses. 7. Vírus de computador – Teses. I. Pena, Patrícia Nascimento. II. Rudie, Karen. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pelo Bibliotecário Marcio Anderson de Andrade Gomes CRB/6 2812
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"CONTROL OF DISCRETE EVENT SYSTEMS SUBJECT TO CYBER ATTACKS"

MICHEL RODRIGO DAS CHAGAS ALVES

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica. Aprovada em 09 de novembro de 2022. Por:

Profa. Dr. Patrícia Nascimento Pena
DELT (UFMG) - Orientadora

Prof. Dr. Karen Rudie
Electrical and Computer Engineering (Queen's University)

Prof. Dr. Lilian Kawakami Carvalho
Departamento de Engenharia Elétrica (UFRJ)

Prof. Dr. Max Hering de Queiroz
Departamento de Automação e Sistemas (UFSC)

Prof. Dr. Ricardo Hiroshi Caldeira Takahashi
DMAT (UFMG)

Prof. Dr. Carlos Andrey Maia
DEE (UFMG)

Documento assinado eletronicamente por **Patricia Nascimento Pena**, **Professora do Magistério Superior**, em 10/11/2022, às 15:02, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lilian Kawakami Carvalho**, **Usuária Externa**, em 16/11/2022, às 13:42, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Karen Gail Rudie**, **Usuária Externa**, em 16/11/2022, às 14:29, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Carlos Andrey Maia**, **Professor do Magistério Superior**, em 22/11/2022, às 14:33, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Max Hering de Queiroz**, **Usuário Externo**, em 02/12/2022, às 08:03, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Ricardo Hiroshi Caldeira Takahashi**, **Professor do Magistério Superior**, em 02/12/2022, às 15:05, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1869853** e o código CRC **A109C4FD**.

Referência: Processo nº 23072.264674/2022-14 SEI nº 1869853

Este trabalho é dedicado aos meus pais, por todo o apoio.

Acknowledgments

First I would like to thank my parents, Maria Celeste and João Batista, for always supporting me. I also thank my friends, my sister Aline and my dog Belize for all the support and company they shared with me over the last years. Without them, this journey would not be possible. Finally, I would like to express that I am very lucky to have two brilliant women as my advisors, Patrícia and Karen.

"La liberté commence où l'ignorance finit"
(Victor Hugo)

Resumo

O objeto de estudo dessa tese são os sistemas modelados como sistemas a eventos discretos. O sistema é composto por vários subsistemas cujo comportamento é coordenado pela ação de um controlador, que por sua vez é uma estrutura composta, dentre outros elementos, pelos supervisores. Considera-se que tanto o controlador quanto cada um dos subsistemas estão conectados a uma rede de comunicação e que estão sujeitos à ação de um agente malicioso que pode modificar as mensagens enviadas e recebidas. Inicialmente são apresentados resultados relativos a uma propriedade chamada P-observabilidade para um conjunto de ataque, que garante, sob certas circunstâncias, a obtenção de um supervisor que assegura que o comportamento desejado é sempre obtido, independentemente da ação do agente malicioso. Em seguida, apresenta-se um novo modelo de ataque, que considera diferentes ações que um atacante pode realizar num sistema real. Além disso, é definida uma nova classe atacantes, chamados de persistentes. Adicionalmente, apresenta-se uma técnica para o projeto dessa classe de atacantes, que têm o objetivo de realizar o ataque e ao mesmo tempo permanecerem ocultos. Finalmente, os detalhes da implementação de um *testbed* para técnicas de projeto de atacantes são apresentados.

Palavras-chave: Cyber ataques, sistemas a eventos discretos, ataque persistente, teoria de controle supervisório

Abstract

The objects of study of this thesis are systems modeled as discrete-event systems. The system is composed of several subsystems whose behavior is coordinated by the action of a controller, which in turn is a structure composed, among other elements, by supervisors. It is considered that the controller and each of the subsystems are connected to a communication network and that they are subject to the action of a malicious agent that is able to modify the messages sent and received. Initially, results related to a property called P-observability for an attack set are presented, which guarantees, under certain circumstances, obtaining a supervisor that ensures that the desired behavior is always obtained, regardless of the action of the malicious agent. Next, a new attack model is presented, which considers the different types of actions an attacker is able to do in a real system. Additionally, a new class of attackers is introduced, called persistent attackers. Moreover, new design technique for this class of attackers is presented, which have the goal to perform the attack while remaining stealthy. Finally, the details about the implementation of a testbed for attacker design techniques is presented.

Keywords: Cyber-attacks, discrete-event systems, persistent attackers, supervisory control theory

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

The advances in computing, communication and related hardware technologies enabled the rapid development of cyber-physical systems (CPSs), which are an integration of physical processes, ubiquitous computation, efficient communication and effective control. Cost-saving and real-time deployment are the two dominant features of CPSs, which have been considered as a core ingredient in the 4th Industrial Revolution (Ding et al., 2018). Because devices and networks have vulnerabilities, their extensive use increases the system's overall vulnerabilities to cyber-attacks. The conventional network defenses, such as firewalls, are not suitable to CPS, since they can introduce delays that may interfere with the control logic (Lima et al., 2022). Thus, it is important to investigate security in this context.

The application fields are multidisciplinary and include aircraft transportation systems, battlefield surveillance, chemical production, energy, food supply, healthcare, industrial automation, manufacturing systems, maritime processes, mobile devices, robotics and transportation (Lu, 2017). Furthermore, networked control systems, wireless sensor and actuator networks and wireless industrial sensor networks are considered a subgroup of CPSs (Ding et al., 2018; Leitão et al., 2016; Rasmussen et al., 2017).

Regardless of the application, one of the elemental tasks in a CPS is to decide which actuators/sensors should be activated to perform a particular action or how to manage control/sampling actions properly. Due to physical constraints or technological limitations, data from sensors or directed to actuators and other networked components may be transmitted over networks without proper security protections. Many of these applications are safety critical and continuously require reliable operation and monitoring, despite the potential occurrence of cyber-attacks and thus the requirement for security measures handling such incidents increases (Rashidinejad et al., 2019; Fritz and Zhang, 2018; Lu, 2017). In (Wang and

Yang, 2019a), a survey on the recent development of securing networked control systems is presented, as well as some real incidents caused by attacks.

Within the area of Discrete-Event Systems (DES), especially in the context of supervisory control, several works tackle the issue of protecting a system from malicious attacks. Normally, the aim of an attacker is to cause damage to the system by leading it to an unsafe state or to acquire critical information from it and there is no unified model that describes them all. There are different aspects from which one can consider an attack, such as: attack location, attacker action and defense strategy.

Regarding the attack location, it can happen in the communication channel from plant to supervisor, called *sensor attack* or *output attack* (Su, 2018; Zhang et al., 2018; Meira-Goes et al., 2019; Mohajerani et al., 2020), in the communication channel from supervisor to plant, called *actuator attack* or *input attack* (Carvalho et al., 2016; Lin et al., 2019a, 2020), or in both channels (Lima et al., 2019; Wang and Yang, 2019a).

Furthermore, there are three main types of actions that an attacker can use to tamper with the communication: insertion, deletion or replacement of symbols. Concerning the defense strategy, it includes design of resilient supervisors against attacks, as in (Meira-Goes et al., 2021; Zhu et al., 2019a; Wang and Yang, 2019a; Lin et al., 2019b; Wakaiki et al., 2019), among others, and approaches for detecting and preventing attacks, as presented in (Fritz and Zhang, 2018; Li et al., 2020) and (Seatzu, 2018).

Moreover, some authors focus on studying the design methods for the attackers, using as an argument that a good understanding about the adversaries can provide better insight on how to defend against them (Lin and Su, 2020; Zhang et al., 2020; Mohajerani et al., 2020; Zhang et al., 2018) and (Meira-Goes et al., 2017).

However, the literature that describes cyber-attacks in general, do not distinguish attacks on different communications channels. Indeed, the communication channels are normally not separated and the attacks often happen in the devices connected to the communication network, instead of the network itself (Cao et al., 2020; Bhamare et al., 2020; Hemsley and Fisher, 2018). This gap between the security-related approaches in the DES framework to the general literature on cyber-attacks is an obstacle for the application of theoretical results in the real world. This thesis proposes a DES attack model that is closer to real-world applications, as well as a design technique for a special class of attackers, called *persistent attackers*. The goal of a persistent attacker is to cause the production of off-specification products or to insert small delays in a way that the attacker remains stealthy, even after its action. Furthermore, a prototype of a networked DES control system was built to serve as a testbed for attack and

defense techniques.

The next section summarizes the main contributions of this thesis.

1.2 Contributions

The goal of this thesis is to fill some gaps in the literature related to cyber-attacks in DES control. The contributions are summarized next.

- Development of a new test and a visual interpretation for the P-observability for an attack set property. The property is related to the ability of a supervisor to enforce, under some conditions, the desired behavior of the system even when a malicious agent is acting.
- Development of an attack model, that describes attacks in the network interface of the devices and considers that an attacker can insert events to the device observation or to the network and can erase events from the device observation and from the network.
- Description of a new class of attacker, called *persistent attacker*. Such attackers have the goal to cause the production of off-specification products or to introduce small delays in the process. Furthermore, a persistent attacker wants to remain hidden even after its action, so it can act more than once.
- Development of a design technique for persistent attackers, aiming to improve the knowledge about the attackers which will, ultimately, help with the creation of defense strategies. This thesis only covers the attacker's design process.
- Development of a testbed for attack and defense strategies and that can capture real-world attacks. The testbed is a prototype of a networked DES control system and offers access to incoming and outgoing communication of all devices in the network, which allows one to implement and evaluate attack and defense techniques. Additionally, an implementation scheme for a networked DES control system was proposed and the details shared in an online repository.

Next, the structure of this document is described.

1.3 Text Organization

This text is organized as follows. In Chapter 2 a review of the pertinent literature on cyber-attacks is presented. Chapter 3 presents the basic concepts of discrete-event systems modeled by languages and automata (Sections 3.1 and 3.2, respectively), as well as the main

concepts of supervisory control theory (Section 3.3) and some details about implementation of DES control (Section 3.4).

The first results of this thesis are presented in Chapter 4, Firstly, the attack model and the property P-observability for an attack set are presented, (Sections 4.1 and 4.2, respectively). Then, the new test for P-observability for an attack set is presented (Section 4.3), followed by algorithms that allows one to verify the property in a computational tool (Section 4.4) and their complexity analysis (Section 4.5). The chapter ends by presenting a case study (Section 4.6).

Chapter 5 presents the results related to persistent attackers. In Section 5.1, the proposed attack model is presented and in Section 5.2 a technique for design of persistent attacks is described. The chapter ends with some discussion about the results (Section 5.3).

The details about the implementation of the testbed are presented in Chapter 6. The chapter describes the proposed system architecture (Section 6.1), the physical process under control (Section 6.2), the hardware (Section 6.3) and software (Section 6.4) developed to implement the testbed and some details about how to implement cyber-attacks in it (Section 6.5). Then, some discussion is presented (Section 6.6).

Finally, Chapter 7 presents the conclusions of this thesis and directions for future work (Section 7.1).

Chapter 2

Literature Review

Cyber Physical Systems feature heavily in Industry 4.0, since they are able to integrate the physical and virtual worlds by providing real-time data processing services (Lu, 2017). A CPS consists of a physical system and a cyber system. It is the result of the integration of physical processing, sensing, computation, communication and control. More specifically, a CPS allows a physical system to be equipped with a virtual system as a monitor, enabling data collected from the physical world to be analyzed in the virtual world such that decisions can be made to affect the course of the physical world. Therefore, a CPS enables integration, sharing and collaboration of information, as well as real-time monitoring and global optimization of systems (Duo et al., 2022).

The integration of systems and technologies in CPS tends to be complex and diverse, making it a compatible and open system, which unfortunately provides a platform for malicious agents to exploit CPS vulnerabilities and may result in numerous security issues. One of the most ubiquitous problems is cyber-attacks, which can degrade system performance, or even cause catastrophic consequences. Due to the impact that can be caused by cyber-attacks on CPS, this subject has been receiving much attention over the last years. There are different surveys on the topic available in the literature (Duo et al., 2022; Ding et al., 2021; Zhang et al., 2021; Tan et al., 2020; Cao et al., 2020; Singh et al., 2020; Dibaji et al., 2019; Rashidinejad et al., 2019; Mahmoud et al., 2019; Giraldo et al., 2018; Ding et al., 2018).

In (Cao et al., 2020), the authors classify attacks on CPS in three different types: a) attacks on the execution layer, which is comprised of devices as sensors and actuators; b) attacks on the data transmission layer and; c) attacks on the application control layer. The authors also point out that most of the research described in the literature on cyber-attacks focuses on attacks at the execution layer and data transmission layer. This does not follow the literature that describes real cases of cyber-attacks on control systems such as (Shareef, 2022; Bhamare et al., 2020; Slowik, 2020; Prinsloo et al., 2019; Wang and Yang, 2019b; Ginter,

Andrew, 2018; Jang-Jaccard and Nepal, 2014; Cárdenas et al., 2011), which describes mostly attacks on the execution layer and on the application control layer.

According to Shareef (2022), a ransomware attack targeted the largest and most important oil pipeline in the United States, bringing the facility to a complete halt for a few days. This caused an acute fuel shortage, causing the fuel prices to increase drastically. The attack was executed by hackers that gained entry into the company's network through a dormant virtual private network (VPN) account that had remote access to the company's computer network. The company had to pay a ransom to the hacker group in exchange for the decryption tool to restore its computer network.

In a generic attack on an industrial control system, described by Ginter, Andrew (2018), an organized crime syndicate targets known vulnerabilities in Internet-exposed services and gain access to the ICS network. Then, the attackers download and analyze control system configuration files. They then reprogram a single PLC, causing it to misoperate a single, vital, piece of physical equipment, while reporting to the plant HMI that the equipment is operating normally. The equipment wears out prematurely, in a season of high demand for the plant's commodity output, e.g., gasoline. The plant shuts down for emergency repairs, of this apparently random equipment failure. The same attack occurs at two nearby plants. Once the equipment has failed, the perpetrators erase all evidence of their presence from the affected plants' ICS networks. Prices of the affected commodity spike on commodities markets. When plant production at all plants returns to normal, commodity prices return to normal. This attack is repeated in the next season of high demand.

In contrast to the examples given, most of the literature on cyber-attacks on control systems, in the context of continuous-time systems, as in (Pan et al., 2022; Pang et al., 2021; Tahoun and Arafa, 2021; Wang and Yang, 2019a), among others, or in the context of discrete-event systems, as will be seen next, describes attacks on the data transmission layer. It is important to highlight that the literature on cyber-attacks actually describes cases of attacks on the data transmission layer. The converse is not common, namely, attacks on control or the execution layer reported by the literature that describes research on security against cyber-attacks on control systems. This is a gap this thesis intends to reduce. Table 2.1 summarizes the different approaches on cyber security found in the literature and how the results presented in this thesis compare with other works.

An overview of the different approaches applied to the supervisory control of DES under attacks is provided by Rashidinejad et al. (2019). The work proposes a framework for classifying the approaches under different aspects and associates to them works found in the literature, with the aim to identify possible gaps on the subject. Firstly, the attacks are

Table 2.1: Summary of works on cyber security

Work	Literature on security of control systems			Attack location			Strategy		
	DES	CT	CA	S	A	D	Def	AD	O
This thesis	■			■		■	■	■	
(Shareef, 2022), (Slowik, 2020), (Prinsloo et al., 2019), (Ginter, Andrew, 2018), (Hemsley and Fisher, 2018)			■	■	■	■			■
(Meira-Goes et al., 2017), (Meira-Góes et al., 2020), (Mohajerani et al., 2020), (Zhang et al., 2018), (Zhang et al., 2020)	■			■				■	
(Alves et al., 2022c)	■			■		■			■
(Alves et al., 2022a), (Chen et al., 2022), (Gao et al., 2019), (Li et al., 2020), (Meira-Goes et al., 2019), (Meira-Goes et al., 2021), (Su, 2018), (Wakaiki et al., 2019), (You et al., 2022)	■			■			■		
(Khoumsi, 2019), (Lin et al., 2019b), (Lin and Su, 2020), (Wang and Pajic, 2019a)	■			■	■			■	
(Alves et al., 2022b)	■			■	■	■		■	
(Zhu et al., 2019b)	■			■	■				■
(Fritz and Zhang, 2018), (Fritz et al., 2019), (Lima et al., 2017), (Lima et al., 2018), (Lima et al., 2019), (Lima et al., 2022), (Lima et al., 2022), (Lin et al., 2019a), (Wang and Yang, 2019a), (Wang et al., 2021), (Wang et al., 2020), (Zhang et al., 2021), (Zhang and Feng, 2020), (Zhou et al., 2020), (Carvalho et al., 2018)	■			■	■		■		
(Lin et al., 2020)	■				■			■	
(Carvalho et al., 2016), (Zhu et al., 2019a)	■				■		■		
(Cárdenas et al., 2011)		■				■	■	■	
(Pan et al., 2022)		■		■			■		
(Orojloo and Azgomi, 2019), (Pang et al., 2021), (Tahoun and Arafa, 2021), (Weerakkody et al., 2017)		■		■	■		■		

Legend:
DES: discrete-event systems
CT: continuous-time
CA: literature on cyber attacks in general
S: sensor channel
A: actuator channel
D: device
Def: defense
AD: attack design
O: other

classified as active or passive. In an active attack, the attacker's goal is to cause damage to the system, while in a passive attack, the goal is to learn secrets about the system. The attacks can be also classified according to the attack location, which may occur in the observation channel, in the control channel or in both. Furthermore, they can be classified regarding the way in which they modify information, i.e., by deletion, insertion or replacement. The last aspect is the security mechanism. The defense strategies can be mainly classified as detection and/or prevention and synthesis of resilient supervisors. Finally, they indicate some directions for future research, such as actuator disablement attack and replacement attack and attacks with delay/disordering impact and supervisor obfuscation

The problem of attacks in the supervisory control context was addressed by Carvalho et al. (2016), where they consider an attack scenario where the attacker has infiltrated a set of vulnerable actuators, and where the attacker overrides the control actions from the supervisor. Attack detection is formulated as a fault diagnosis problem and the proposed defense strategy is to disable all controllable events once the attack is detected. The condition for the system to be able to detect and stop the attack before it is successful is characterized and called AE-safe controllability. These results are improved in (Carvalho et al., 2018), in which the authors extended the analysis by considering attacks in the sensor channel as well. Furthermore, the AE-safe controllability property is extended to a more generic property called GF-safe controllability, which is a property that the system has to satisfy in order to the damage to be prevented, regardless of the type of the attack.

The authors of (Lima et al., 2017) also improve the work of (Carvalho et al., 2016) proposing a defense strategy that detects intrusions and prevents damage caused by man-in-the-middle[1] attacks in the sensor and/or control communication channels in supervisory control systems. This is done by an Intrusion Detection Module that observes the traces seen by the supervisor and disables all controllable events, after the attack is detected. The new condition for the system, that allows the proposed defense strategy, is called NA-safe controllability. Later improvements are presented in (Lima et al., 2018), where the security module does not disable all controllable events once the attack is detected. Only the events that may lead the system to an unsafe region are disabled. Then, Lima et al. (2019) improves the previous results by providing an algorithm that verifies the NA-safe controllability property, as well as an implementation method for the security module. In a more recent work, the authors propose a new defense strategy that prevents damages in the system caused by man-in-the-middle attacks (Lima et al., 2022). For this, a security supervisor is proposed to disable controllable events when there is a risk or reaching unsafe states. The security supervisor operates together

[1] A man-in-the-middle attack is a type of cyber attack in which the attacker secretly intercepts and relays messages between two parties who believe they are communicating directly with each other. The attack is a type of eavesdropping in which the attacker intercepts and then controls the entire conversation.

with the existing supervisor.

In (Alves et al., 2019), a supervisory control architecture is considered, where the communication between plant and supervisor is made through a network that can have multiple channels subject to communication delays, that can cause changes in the order of observations, and intermittent loss of observations. Necessary and sufficient conditions for the existence of a robust supervisor that is able to operate even under intermittent loss of observations and also enforces a specification language are provided, as well as a method to obtain it. In the same context, in (Alves et al., 2020), a methodology to construct an equivalent untimed automaton model that takes into account all possible effects of delays and loss of observations is presented. It is assumed that the minimal activation times of the plant transitions and maximum observation delay since the event occurrence until its observation by the supervisor are known. Additionally, based on the untimed model, a supervisory control problem of Networked Discrete-Event Systems with Timing Structure is formulated and its solution is proposed, as well as an implementation scheme. In the context of diagnosis, the authors of (Alves et al., 2022c) consider a networked discrete-event system (NDES) subject to denial-of-service (DoS) and deception attacks that flood some communications channels with fake packets causing delays and loss of observations and insertion of fake observations. Then, they propose an automaton model for NDES subject to such attacks that represents the adverse effects of DoS-D attacks on the observations of local diagnosers. Finally, they introduce a new codiagnosability definition called DoS-D-robust codiagnosability, and present a necessary and sufficient condition for a language to be DoS-D-robustly codiagnosable.

The authors of (Li et al., 2020) address the problem of detection and prevention of cyber-attacks where the supervisor communicates with the plant via network channels. Random control delays may occur in such a networked system, in addition to a cyber-attacker targeting the vulnerable actuators. The attacker can corrupt the control input of the supervisor, and aims to drive the plant to unsafe states. Approaches to model the closed-loop system subject to control delays and attacks are proposed. The notion of AE-safe controllability in the networked control system is defined. It describes the ability to prevent the plant from reaching unsafe states after attacks are detected. A method for testing AE-safe controllability is also presented.

The authors of (Wang and Pajic, 2019b) use finite state transducers (FSTs), which are an extended definition of finite state automata, to model the attacks. Using this formalism, a wide class of attacks can be modeled. Three types of attacks are considered: attacks in the observation channel, attacks in the control channel and attacks in both channels. For each type, a controllability condition is presented along with synthesizing algorithms for attack-resilient supervisors. This work is extended in (Wang and Pajic, 2019a), where the

problem of computing the maximal controllable sub-language (MCSL) of a desired language is solved. A desired language is called controllable if there exists a security-aware supervisor that ensures that the restricted language executed by the plant for any possible attack behavior is the desired one. Such supervisor is said to be attack-resilient. Then, a design algorithm for an attack-resilient supervisor is presented.

In (Lin et al., 2019b), the authors investigate the security approach of synthesizing resilient supervisors against combined actuator and sensor attacks. A constraint-based approach for the bounded synthesis of resilient supervisors is developed by reducing it to the Quantified Boolean Formula Problem. The synthesis is bounded by the model's size of the supervisor and attacker.

In (Zhu et al., 2019a), the authors address the problem of supervisor obfuscation against actuator enablement attack. A method to obfuscate a supervisor, that is, to make it resilient against actuator enablement attack in such a way that the behavior of the original closed-loop system is preserved, is proposed. The approach involves a combination of two basic ideas: 1) the formulation of the problem of computing behavior-preserving supervisors as the problem of computing separate finite state automata under controllability and observability constraints, and 2) the use of a previous proposed technique for the verification of attackability, with a normality assumption imposed on both the actuator attackers and supervisors.

The authors of (Wakaiki et al., 2019) consider a multi-adversary version of the supervisory control control problem for DES. Each adversary corrupts in a different way the observations available to the supervisor and has the purpose of leading the system to an unsafe or undesirable state. It is assumed that only one adversary is acting although the information about which one is not known. The goal of the paper is to propose a design method for a supervisor that enforces a specific language in spite of the opponent's actions and without knowing which adversary it is playing against. It is shown that this problem has a solution if and only if the desired language is controllable, in the classical sense, and observable in a novel sense, that takes the adversaries into account. This new observability property is defined and called *P-observability for an attack set*. The authors also show that testing the existence of a supervisor and building the supervisor can be done using tools developed for the classical DES supervisory control problem, by considering a family of automata with modified output maps. In (Alves et al., 2022a), which is a paper resulting from this thesis, a visual interpretation for the P-observability for an attack set is proposed, as well as a test that verifies the property itself.

The problem of attack detection in the framework of partially observable discrete-event systems modeled by automata is tackled by Gao et al. (2019). Using the attack model of (Wakaiki et al., 2019), the authors assume that the observation produced by a plant can be

corrupted by an intruder which, through one or more attack dictionaries, can change events into different strings. The problem addressed is that of detecting if a plant has been attacked and, if such is the case, of identifying the nature of the attack, i.e., which attack dictionaries have been used. It is shown that the problem of attack detection can be reduced to a classical problem of state estimation or fault diagnosis for a new structure which describes the behavior of the plant under attack.

The work of Fritz and Zhang (2018) presents a method for detection of attacks in CPSs, modeled by Petri nets, which are another formalism to describe DESs. The attacks considered are the covert attack and replay attack. The first is a sophisticated attack, where an attacker has complete knowledge of the system model and access to the inputs and outputs. The attacker changes the actuator signals to achieve an attack goal and while remaining stealthy. The second is an attack of two stages. First, sensor data generated by the plant is gathered by the attacker. This data is then replayed into the network, rendering attacks on the actuator signals invisible. In order to make an attack's detection possible, a scheme that uses a permutation matrix in the communication channel is proposed. This idea is similar to encryption methods, but the computational effort is lower. Lastly, a detection module that is based on the idea of fault detection is presented.

In (Fritz et al., 2019) the authors introduce a detection method called time guard detection. Industrial control systems (ICS) are considered, in particular, at the PLC (programmable logic controller) level and process level. The attacks considered can hide manipulations in the logical level of the PLC, but are visible and detectable in the temporal level. The proposed method allows detection of targeted manipulations on the ICS, which results in a temporal change of the production process. The application of the time guard detection is possible for systems where all sensors and actuator channels or only part of them are vulnerable to attacks, i.e., can be observed and changed.

In (Zhang et al., 2018) and (Zhang and Feng, 2020), a plant is considered which generates a sequence of events and such a sequence is observed by an operator through an observation mask. The sensor readings may be altered by an intruder, which can insert or erase events, and whose goal is to make the operator think that the plant is in a safe state while in reality it is in an unsafe one. In addition, the intruder is required to be stealthy, i.e., the operator should not be able to detect that the system is under attack. The goal of the paper is to provide a systematic approach to determine if a stealthy potentially harmful attack function exists. If such is the case, the system is not robust to attacks in the considered setting. For this, a structure called *attacker observer* that describes all possible attack strings that can be generated by the attacker is proposed. Moreover, a structure called *operator observer* is also presented, which describes all possible observations an observer can make, considering the

attacks and including the ones that can reveal the attacker's presence. Finally, a structure called *unbounded attack* is obtained by the composition of both observers. Using this structure, an attacker can choose which action to take while remaining stealthy. An extended version of the previous work was presented in (Zhang et al., 2021), where the authors consider a more specific problem statement and algorithms are provided. The same authors also proposed the application of similar methods in systems modeled by Petri nets (Zhang et al., 2020; Wang et al., 2021). Also in the context of Petri nets, the authors of (You et al., 2022) propose a method for design of liveness-enforcing supervisors under attacks on the sensor channel.

In (Mohajerani et al., 2020), the synthesis of successful sensor deception attack functions in supervisory control using abstraction methods to reduce computational complexity is investigated. In sensor deception attacks, an attacker is able to intercept sensor signals and feeds incorrect information to the supervisor with the intent on causing damage to the supervised system. The attacker is successful if its attack causes damage to the system and it is not identified by an intrusion detection module. The authors present results that enable the existence test and the synthesis method of successful sensor deception attack functions to be realized using abstractions, in order to reduce the computational effort in solving these problems.

The problem of synthesizing an attack strategy for a given controlled system in the context of supervisory control is studied by (Meira-Goes et al., 2017). The proposed model captures a class of deception attacks, where the attacker has the ability to modify a subset of sensor readings and mislead the supervisor, with the goal of inducing the system into an undesirable state. A new type of a bipartite transition structure is also introduced, called Insertion-Deletion Attack structure (IDA), to capture the game-like interaction between the supervisor and the environment, which includes the system and attacker. This structure is a discrete transition system that embeds information about all possible actions of the attacker that keeps it stealthy, and all states, some of which are possibly unsafe, that become reachable as a result of those actions. A procedure for the construction of the IDA is presented as well as the characterization of successful stealthy attacks, i.e., attacks that avoid detection from the supervisor and cause damage to the system. The work of (Meira-Goes et al., 2019) addresses the problem of synthesizing a supervisor that is robust against a large class of edit attacks on the sensor readings. It is formulated and solved using a methodology that is based on the solution of a partially observed supervisory control problem with arbitrary control patterns. Results on the existence of a supremal robust supervisor are also provided. Later, in (Meira-Góes et al., 2020), the authors present a new model for deception attacks on CPS, based on the IDA structure, and present a method for synthesizing attackers. The dual problem, which is the synthesis of robust supervisors against deception attackers is addressed in (Meira-Goes et al., 2021) and relaxes some conditions presented in (Meira-Goes et al., 2019).

In (Su, 2018) one special type of attack is investigated, where an attacker can arbitrarily alter sensor readings after intercepting them from a target system, aiming to trick a given supervisor to issue improper control commands. This type of attack can drive the system to an undesirable state. Firstly, the cyber-attack problem is considered from an attacker's point of view, and it is formulated as an attack-with-bounded-sensor-reading-alterations (ABSRA) problem. Then it is shown that the supremal ABSRA exists and can be computed, as long as the plant model and the supervisor model are regular, i.e., representable by finite-state automata. Upon the synthesis of the supremal ABSRA, a synthesis algorithm is also presented, which computes a supervisor that is ABSRA-robust in the sense that any ABSRA will either be detectable or inflict no damage to the system. Based on the attack model proposed in (Su, 2018), the authors of (Chen et al., 2022) propose the definitions of attackable strong detectability and attackable weak detectability, in the context of sensor attacks, as well as necessary and sufficient conditions for the verification of the proposed properties.

In (Lin et al., 2019a), the problem of covert actuator attacker synthesis is addressed. It is assumed that the actuator attacker partially observes the execution of the closed-loop system and is able to modify each control command issued by the supervisor on a specified attackable subset of controllable events. The authors present a characterization for the existence of a successful attacker and prove the existence of the supremal successful attacker, when both the supervisor and the attacker are normal. Moreover, they present an algorithm to synthesize the supremal successful normal attackers. Further investigation of this problem is done in (Lin et al., 2020), where the authors provide straightforward but in general exponential-time reductions from the covert actuator attacker synthesis problems to the Ramadge-Wonham supervisor synthesis problems. Thus, they claim that it is possible to use the many techniques and tools already developed for solving the supervisor synthesis problem to solve the covert actuator attacker synthesis problem for free. In particular, it is shown that, if the attacker cannot attack unobservable events to the supervisor, then the reductions can be carried out in polynomial time.

The authors of (Zhu et al., 2019b) study the problem of supervisory control of networked discrete-event systems with communication delays and channels that are subject to message loss. Both the observation and control communication channels are represented by finite automata under the assumption that all communication delays are bounded. By a transformation of the plant and specification, they show that it is possible to reduce networked supervisor synthesis to supervisor synthesis in the standard Ramadge-Wonham framework.

According to (Wang et al., 2020), when unsafe behaviors cannot be prevented with certainty, mitigation strategies are limited. The work proposes the use of a probabilistic

discrete-event system (PDES) framework to incorporate a likelihood measure for unsafe behavior in the attack models presented in (Carvalho et al., 2018). The least-unsafe supervisor problem is introduced to minimize this unsafe likelihood measure and improve existing attack mitigation techniques.

The authors of (Zhou et al., 2020) propose a technique to encrypt the signals in a network, using a matrix notation of automata, in the context of supervisory control. An encryption framework based on the matrix notation of automata is proposed, making it suitable for homomorphic encryption schemes over integers, which are emerging in the cryptography area.

In (Weerakkody et al., 2017), the problem of securely designing a decentralized control system to prevent a special class of integrity attacks known as perfect attacks is considered, where an attacker can manipulate the state without affecting the measurements of the system. The goal considered was the design of systems which are not perfectly attackable while simultaneously minimizing communication in the system. Such a design ensures deterministic detection of attacks.

In (Khoumsi, 2019), the problem of sensor and actuator attacks is firstly studied with a defense viewpoint, where methods to detect attacks and avoid failures, and determine conditions of attack detectability and failure avoidability are developed. Then, the attacker's viewpoint is also studied, where a strategy for the attacker to be undetectable is proposed.

The next chapter presents the main concepts on discrete-event systems.

Chapter 3

Preliminaries

When the state space of a system is naturally described by a discrete set and state-transition dynamics are driven by discrete *events*, then these events may be associated to the transitions. Such systems are called *discrete-event systems*.

An event can be associated with a specific action taken, or a spontaneous occurrence or even with several conditions which are all met at a given point of time. An event occurs instantaneously and causes transitions from one state to another. This state transition mechanism is called *event-driven*.

There are many mathematical tools used to describe the behavior of a discrete-event system. In this work, languages and automata are employed. This section aims to present the concepts that are relevant for a good understanding of the content of the next chapters. For a more detailed overview, the reader is referred to (Wonham and Cai, 2019) and (Cassandras and Lafortune, 2007).

3.1 Language

A finite set of symbols Σ is associated to the physical events of a DES. It is common practice to call the elements of Σ as *events* as well. The set Σ is also called an *alphabet* and sequences of elements of Σ can be called *words*, *strings* or *traces*. A string with no event is called the empty string and is denoted by ε. The length of a string is the number of events contained in it, counting multiple occurrences of the same event. If s is a string, we denote its length by $|s|$. By convention, $|\varepsilon| = 0$.

A formal way to study the behavior of a DES is based on the theory of languages. It specifies all possible sequences of events that the DES is capable of processing. A *language* defined over an event set Σ is a set of finite-length strings formed with events in Σ. The most

basic operation while building strings is the *concatenation*. The concatenation uv of strings u and v is the new string consisting of the events in u immediately followed by the events in v. If $s = uv$, then it is said that u is a *prefix* of s, denoted by $u \leq s$, while v is a *suffix* of s. The empty string ε is the identity element of concatenation, which means $u\varepsilon = \varepsilon u = u$ for any string u.

The set Σ^* , defined by

$$\Sigma^* := \bigcup_{i=0}^{\infty} \Sigma^i,$$

with $\Sigma^0 = \{\varepsilon\}$, and Σ^i being a language formed by all strings of length i that can be constructed with elements in Σ, is the infinite set of all finite strings of elements in Σ, including the empty string ε. The $*$ operation is called *Kleene-closure*. Any language over an event set Σ is therefore a subset of Σ^*. As languages are sets, all the usual operations such as union, intersection and difference are applicable to them. Next, some other useful operations are presented.

Let $A, B \subseteq \Sigma^*$, then the concatenation AB of languages A and B is

$$AB := \{s \in \Sigma^* | (s = ab) \wedge (a \in A) \wedge (b \in B)\}.$$

In words, a string is in AB if it can be written as the concatenation of a string in A with a string in B. Now, let $L \subseteq \Sigma^*$, then the prefix-closure of L, denoted by $\overline{L}$ is

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

The prefix-closure of L consists of all prefixes of all strings in L. The language L is said to be *prefix-closed* if $L = \overline{L}$.

The operation Kleene-closure can also be applied over a language $L \subseteq \Sigma^*$, as follows:

$$L^* := \bigcup_{i=0}^{\infty} L^i.$$

By convention $L^0 = \{\varepsilon\}$ and L^i is formed by the concatenation of L with itself a number of times given by i. An element of L^* is formed by the concatenation of a finite number of elements of L, including the concatenation of zero elements.

Another type of operation performed on strings and languages is the so-called *natural projection* or simply *projection*, from a set of events Σ to a smaller set of events Σ_o, where $\Sigma_o \subseteq \Sigma$. This operation is denoted by P and a subscript can be added to specify the sets involved in the operation when dealing with multiple sets. The projection $P : \Sigma^* \rightarrow \Sigma_o^*$ is

defined recursively as follows:

$$P(\varepsilon) := \varepsilon$$
$$P(\sigma) := \begin{cases} \sigma & \text{if } \sigma \in \Sigma_o \\ \varepsilon & \text{if } \sigma \in \Sigma \setminus \Sigma_o \end{cases}$$
$$P(s\sigma) := P(s)P(\sigma) \text{ for } s \in \Sigma^*, \sigma \in \Sigma.$$

Basically, the projection operation takes a string formed from the larger set Σ and erases events in it that do not belong to the smaller event set Σ_o. The corresponding inverse map $P^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$ is defined as

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

where the notation 2^A, with A being a set, is the set of all subsets of A. Given a string of events in the smaller event set Σ_o, the inverse projection P^{-1} returns the set of the strings formed by events from the larger event set Σ that project, with P, to the given string.

Both the projection P and the inverse projection P^{-1} are extended to languages by applying them to all the strings in the language. For $L \subseteq \Sigma^*$,

$$P(L) := \{t \in \Sigma_i^* | (\exists s \in L)[P(s) = t]\}$$

and for $L_i \subseteq \Sigma_i^*$,

$$P^{-1}(L_i) := \{s \in \Sigma^* | (\exists t \in L_i)[P(s) = t]\}.$$

Some useful properties of projections and inverse projections are given next (Cassandras and Lafortune, 2007).

1. $P[P^{-1}(L)] = L$
   $L \subseteq P^{-1}[P(L)]$.

2. $P(A \cup B) = P(A) \cup P(B)$
   $P(A \cap B) \subseteq P(A) \cap P(B)$.

3. $P^{-1}(A \cup B) = P^{-1}(A) \cup P^{-1}(B)$
   $P^{-1}(A \cap B) = P^{-1}(A) \cap P^{-1}(B)$.

4. $P(AB) = P(A)P(B)$
   $P^{-1}(AB) = P^{-1}(A)P^{-1}(B)$.

In the next subsection, automata are introduced. If a language can be represented by a finite-state automaton, then this language is said to be *regular*. Otherwise, it is said to be *non-regular*. In what follows, only regular languages are considered.

3.2 Automata

A regular language can be represented by a finite *automaton*, which can be deterministic or nondeterministic. The definition of a deterministic finite-state automaton is given by Def. 1.

Definition 1 (Deterministic Finite-State Automaton)**.** *A Deterministic Finite-State Automaton, or DFA, denoted by G, is a tuple*

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

where

Q is the finite set of states;

Σ is the finite set of events;

$\delta : Q \times \Sigma \to Q$ is the transition function. A transition $\delta(q_1, \sigma) = q_2$, with $q_1, q_2 \in Q$ and $\sigma \in \Sigma$, means that there is a transition labeled by event σ from state q_1 to state q_2. Normally, δ is a partial function;

q_0 is the initial state;

$Q_m \subseteq Q$ is the set of marked states.

$\Diamond$

Remark 1. *The selection of which states are in the set Q_m is a modeling issue that depends on the problem of interest. The marked states represent that the system has completed some operation or some task. When this is not relevant to the problem under study, we can refer to the automaton G as a tuple $G = (Q, \Sigma, \delta, q_0)$ if we are not interested in the marked states or if all states are understood to be marked.* $\square$

The automaton is deterministic because from any state there can only exists one outgoing transition with a given event label. If multiple transitions were allowed on the same event label, the automaton would be called a *nondeterministic finite automaton*. An automaton can be also represented by its *state transition diagram*, as shown in Figure 3.1. Each state is represented by a circle, while the double circle represents the marked states. An arrow connecting two states represents a transition. The label of the transitions is the event that triggers them. The

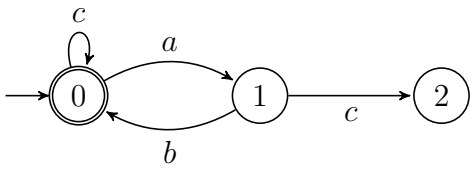


Figure 3.1: Example of an automaton represented by its state transition diagram.

alphabet Σ is the set of all transition's labels. The initial state is indicated by an arrow that doesn't connect two states.

Example 1. *For the automaton of Figure 3.1,* $G = (Q, \Sigma, \delta, q_0, Q_m)$*, with* $Q = \{0, 1, 2\}$*,* $\Sigma = \{a, b, c\}$*,* $q_0 = 0$*,* $Q_m = \{0\}$ *and* $\delta(0, c) = 0$*,* $\delta(0, a) = 1$*,* $\delta(1, c) = 2$ *and* $\delta(1, b) = 0$*.* □

The operation of an automaton starts at the initial state q_0 and upon the occurrence of an event σ such that $\delta(q_0, \sigma)$ is defined, it will make the transition to state $\delta(q_0, \sigma)$. This process continues based on the transitions for which δ is defined. The notation $\delta(q, \sigma)!$, for $q \in Q$ and $\sigma \in \Sigma$, is used to indicate that $\delta(q, \sigma)$ is defined.

Remark 2. *Sometimes it is more convenient to represent the transition function as a set* Δ*. For each* $q, q' \in Q$ *and* $\sigma \in \Sigma$ *such that* $q' = \delta(q, \sigma)$*, then* $(q, \sigma, q') \in \Delta$*. It is assumed that any changes in the transition function* δ *or in the set* Δ*, automatically changes the other.* □

Remark 3. *Another useful map when dealing with automata is the map* $\Gamma : Q \to 2^{\Sigma}$*, which is called the feasible event function. Thus,* $\Gamma(q)$*, for* $q \in Q$*, is the set of all events* $\sigma \in \Sigma$ *for which* $\delta(q, \sigma)$ *is defined and it is called the feasible event set of* G *at state* q*.* □

The transition function δ can be extended from domain $Q \times \Sigma$ to domain $Q \times \Sigma^*$, as follows:

$$\begin{aligned} \delta(q, \varepsilon) &:= q; \\ \delta(q, s\sigma) &:= \delta(\delta(q, s), \sigma) \quad \text{for } s \in \Sigma^* \text{ and } \sigma \in \Sigma. \end{aligned}$$

Each automaton G defines a *generated* language and a *marked* language. The language generated by G is

$$\mathcal{L}(G) := \{s \in \Sigma^* | \delta(q_0, s)!\} \tag{3.1}$$

while the language marked by G is

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) | \delta(q_0, s) \in Q_m\}. \tag{3.2}$$

The language $\mathcal{L}(G)$ represents all the directed paths that can be followed along the state transition diagram, starting at the initial state. The string corresponding to a path is the

concatenation of the event labels of the transitions composing the path. The second language represented by G, $\mathcal{L}_m(G)$, is the subset of $\mathcal{L}(G)$ consisting only of the strings s for which $\delta(q_0, s) \in Q_m$, that is, these strings correspond to paths that end at a marked state.

If an automaton G has a state which cannot be reached by any sequence of event from the initial state, then the state is called non-accessible. The operation $Ac(G)$ removes all non-accessible states from G, as well as their associated transitions. This operation does not affect the generated and marked languages of G. If $G = Ac(G)$, then G is said to be *accessible*.

On the other hand, if G has a non-marked state from which it is not possible to reach any other marked state, then this state is called not coaccessible. The operation $CoAc(G)$ removes all not coaccessible states fo G. This operation does not affect the marked language, but can reduce the generated language. If $G = CoAc(G)$, the automaton is called *coaccessible* and in this case, $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$.

When the accessible part of an automaton is not coaccessible, it is called *blocking*. If an automaton G is accessible and coacessible, then G is said to be *trim*. The $Trim$ operation is defined as

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

Let G be an automaton, Σ its event set and consider a set $\Sigma_i \subset \Sigma$. The projections from Σ^* to Σ_i^*, $P(\mathcal{L}(G))$ and $P(\mathcal{L}_m(G))$ can be obtained from G by replacing all transitions labels in $\Sigma \setminus \Sigma_i$ by ε. This may result in a nondeterministic automaton that generates and marks the desired languages. A method for transforming a nondeterministic automaton in a deterministic one can be found in (Hopcroft et al., 2006, Section 2.3.5).

For the inverse projection, P be the projection from Σ^* to Σ_i^*. An automaton that generates $P^{-1}(\mathcal{L}(G))$ and marks $P^{-1}(\mathcal{L}_m(G))$ can be obtained by adding self-loops for all the events in $\Sigma \setminus \Sigma_i$ at all the states of G.

In general, when modeling systems composed of interacting components, the event set of each component includes private events that pertain to its own internal behavior and common events that are shared with other automata and capture the coupling among the respective system components. The standard way of building models of entire systems from models of individual system components is by *parallel composition*. The parallel composition of automata G_1 and G_2 is the automaton

$$G_1||G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

where

$$\delta((q_1,q_2),\sigma) := \begin{cases} (\delta_1(q_1,\sigma),\delta_2(q_2,\sigma)) & \text{if } \delta_1(q_1,\sigma)! \wedge \delta_2(q_2,\sigma)! \\ (\delta_1(q_1,\sigma),q_2) & \text{if } \delta_1(q_1,\sigma)! \wedge \sigma \notin \Sigma_2 \\ (q_1,\delta_2(q_2,\sigma)) & \text{if } \delta_2(q_2,\sigma)! \wedge \sigma \notin \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

In the parallel composition, a shared event σ, that is, an event that belongs to $\Sigma_1 \cap \Sigma_2$, can be executed if the two automata can execute it simultaneously. This makes the automata to synchronize on the shared events. The private events, i.e., the ones that belong to $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$, do not have any restriction and can be executed whenever they are possible.

3.3 Supervisory Control Theory

Assume that a given DES is modeled by automaton G, where Σ is the event set of G. It is said that G models the *uncontrolled behavior* or the *open-loop behavior*. The premise is that some of the states in G must be avoided, as they represent unsafe or blocking states. This is achieved when the system is under control, which restricts the behavior of the system to a subset of $\mathcal{L}_m(G)$. The controller is called a *supervisor*, denoted by S.

In this context, the set Σ is partitioned into two subsets

$$\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$$

where

Σ_c is the set of *controllable* events: these events can be prevented from happening, or *disabled*, by the action of a supervisor S;

Σ_{uc} is the set of *uncontrollable* events: these events cannot be prevented from happening by the action of a supervisor S.

In general, uncontrollable events are associated with changes in sensor readings, as opposed to the controllable events, that are associated with control actions, as the command to an actuator.

Assume that all the events in Σ executed by G are observed by supervisor S. Thus, in Figure 3.2, s is the string executed so far by G and s is entirely seen by S. The transition function of G can be controlled by S in the sense that the controllable events of G can be dynamically enabled or disabled by S. Formally, a supervisor S is a function $S : \mathcal{L}(G) \to 2^{\Sigma}$. For each $s \in \mathcal{L}(G)$ generated so far by G, under the control of S, $S(s)$ is the set of enabled

events that G can execute at its current state $\delta(q_0, s)$.

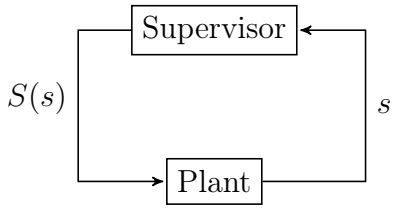


Figure 3.2: Feedback loop of supervisory control.

The supervisor S is said to be *admissible* if, for all $s \in \mathcal{L}(G)$, $\Sigma_{uc} \subseteq S(s)$, which means that S is not allowed to disable an uncontrollable event. From now on, only admissible supervisors will be considered.

The set $S(s)$ is called *control action* at s, while S is the *control policy*. The system under control is represented by S/G and its generated language is given by

i. $\varepsilon \in \mathcal{L}(S/G)$;

ii. If $s \in \mathcal{L}(S/G)$, $\sigma \in S(s)$ such that $s\sigma \in \mathcal{L}(G)$, then $s\sigma \in \mathcal{L}(S/G)$;

iii. No other strings belong to $\mathcal{L}(S/G)$.

The marked language is defined as

$$\mathcal{L}_m(S/G) := \mathcal{L}(S/G) \cap \mathcal{L}_m(G).$$

It is clear that $\mathcal{L}(S/G) \subseteq \mathcal{L}(G)$ and it is prefix-closed by definition. Also, $\mathcal{L}_m(S/G)$ consists of the marked strings of G that survive under control of S. The notion of blocking of a DES S/G is formalized in Definition 2.

Definition 2 (Blocking in a controlled system)**.** *The DES S/G is blocking if*

$$\mathcal{L}(S/G) \neq \overline{\mathcal{L}_m(S/G)}$$

and nonblocking when

$$\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$$

$\Diamond$

If the DES S/G is blocking, we also say that the supervisor S is blocking. On the other hand, if S/G is nonblocking, then we say that S is nonblocking.

The subset of $\mathcal{L}(G)$ that represents the behavior one wants to restrict the system to is usually represented by an automaton H, that marks a language K, i.e., $K = \mathcal{L}_m(H)$. There are numerous ways to obtain the automaton H, such as to take automaton G and remove from it some of the states (and its associated transitions) that are states to be avoided. The language K is normally known as the *desired language* and the goal of the control design is to obtain a supervisor S such that $\mathcal{L}(S/G) = \overline{K} \subseteq \mathcal{L}(G)$

The main requirement for the existence of a supervisor is known as the *controllability* property, presented in Definition 3.

Definition 3 (Controllability)**.** *Consider a DES G whose event set is partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Let $K \subseteq \mathcal{L}(G)$, $K \neq \emptyset$, be a desired language. The language K is said to be controllable with respect to $\mathcal{L}(G)$ and Σ_{uc} if*

$$\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}.$$

$\Diamond$

In other words, a language $K \subseteq \mathcal{L}(G)$ is controllable with respect to $\mathcal{L}(G)$ if for all $s \in \overline{K}$, for all $\sigma \in \Sigma_{uc}$, if $s\sigma \in \mathcal{L}(G)$, it implies that $s\sigma \in \overline{K}$. For a string $s \in \mathcal{L}(G)$ and an event $\sigma \in \Sigma$, $s \in \overline{K}$ but $s\sigma \notin \overline{K}$, it means that the supervisor should *disable* event σ after s. If K is controllable with respect to $\mathcal{L}(G)$, then the supervisor will never disable an uncontrollable event. In other words, if an event cannot be prevented from happening, then it should be legal.

A supervisor S can be represented as an automaton, which is called a *realization* of S, denoted as $\mathcal{S}$. In order to build an automaton realization of S it suffices to build an automaton that marks the language $\mathcal{L}_m(S/G)$.

When K is not controllable with respect to $\mathcal{L}(G)$, it is possible to say that there exists an unique maximal, i.e., a supremal controllable sublanguage, denoted by $\sup \mathcal{C}(K, \mathcal{L}(G))$. Thus, if K is not controllable, the supremal controllable sublanguage contained in K preserves the restrictions imposed by K and can be seen as an optimal and minimally restrictive approximation of K. If the automata G and H have, respectively, l and k states, then the calculation of $\sup \mathcal{C}(K, \mathcal{L}(G))$ has polynomial complexity in l and k. However the number of states of H and G grows exponentially on the number of subsystems G_i, that compose the global system G. The process of obtaining the supervisor $S = \sup \mathcal{C}(K, \mathcal{L}(G))$, is called *monolithic* synthesis, since only one supervisor is obtained.

3.3.1 Supervisory Control under partial observation

Sometimes, a DES has events that occur in the system modeled by the automaton but are not seen, or observed, by an outside observer of the system behavior. This lack of observability can be due to the absence of a sensor to record the occurrence of the event, for example. Such events are characterized as *unobservable*, while the events that are always seen are said to be *observable*. Thus the event set Σ of G can be partitioned into the subsets

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

where Σ_o and Σ_{uo} are the sets of observable and unobservable events, respectively. Such DES are referred to as being *partially-observed*. The closed loop for control under partial observation is shown in Fig. 3.3 and includes a natural projection P between the plant and the supervisor. The presence of unobservable events in a DES imposes some limitations on the controlled behaviors that can be achieved by the action of a supervisor. An additional condition for a desired language to have in order to be possible to obtain a supervisor that enforces it is called *observability* and is presented in Definition 4.

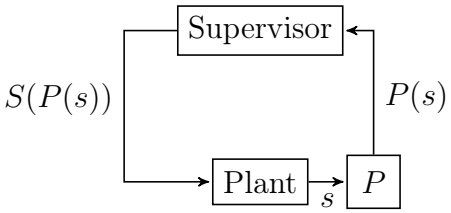


Figure 3.3: Feedback loop of supervisory control in the case of partial observation.

Definition 4 (Observability (Lin and Wonham, 1988))**.** *Let K and $\mathcal{L}(G)$ be languages defined over event set Σ, with $K \subseteq \mathcal{L}(G)$. Also, let $P : \Sigma^* \to \Sigma_o^*$ be a natural projection and Σ_c the set of controllable events. The language K is said to be observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c if for all $s \in \overline{K}$ and for all $\sigma \in \Sigma_c$,*

$$(s\sigma \notin \overline{K}) \text{ and } (s\sigma \in \mathcal{L}(G)) \implies P^{-1}(P(s))\sigma \cap \overline{K} = \emptyset.$$

$\Diamond$

The term $P^{-1}(P(s))\sigma \cap \overline{K}$ identifies all strings in $\overline{K}$ that have the same projection as s and can be continued within $\overline{K}$ with event σ. If this set is not empty, namely, if K is not observable, this means that K contains two strings, s and s', such that $P(s) = P(s')$, and where $s\sigma \notin \overline{K}$ while $s'\sigma \in \overline{K}$. In other words, if a supervisor cannot differentiate between two strings, then these strings should require the same control action. An alternative way to define

observability is given next (Lin and Wonham, 1988).

A prefix-closed language $K \subseteq \mathcal{L}(G)$ is observable *with respect to* $\mathcal{L}(G)$ if

$$\ker P \subseteq \text{act}_{K \subset \mathcal{L}(G)}, \tag{3.3}$$

where $\ker P$ denotes the relation on Σ^* defined by

$$\ker P := \{(w, w') \in \Sigma^* \times \Sigma^* | P(w) = P(w')\} \tag{3.4}$$

and $\text{act}_{K \subset \mathcal{L}(G)}$ is the binary relation on Σ^* defined by

$$\begin{aligned} \text{act}_{K \subset \mathcal{L}(G)} :=& \{(w, w') \in \Sigma^* \times \Sigma^* | \\ & (w, w' \in K)(\nexists \sigma \in \Sigma)[w\sigma \in K \wedge w'\sigma \in L \setminus K] \vee [w\sigma \in L \setminus K \wedge w'\sigma \in K]\}. \end{aligned} \tag{3.5}$$

The relation $\text{act}_{K \subset \mathcal{L}(G)}$ has all pairs of strings $w, w' \in K$ such that the new strings $w\sigma$ and $w'\sigma$ are either both in K or both in $\mathcal{L}(G) \setminus K$, for all $\sigma \in \Sigma$. Thus a language is P-observable if for any two strings that result in the same observation, the control action required after these strings has to be consistent, i.e., an event should be enabled after both strings or disabled after both of them. In (Wakaiki et al., 2019), the authors refer to the concept of observability with respect to a natural projection P and $\mathcal{L}(G)$ as *P-observability with respect to* $\mathcal{L}(G)$.

The next section presents a literature review on implementation of DES control.

3.4 Implementation of DES control

Since this thesis focuses on attacks on systems and one of the goals is to have a model that can capture real-world attacks, in addition to a prototype of a DES control system to be used as a testbed, a brief review of how researchers have implemented supervisory control on hardware and software and what issues arise in moving from mathematical modeling to implementation is given here. As will be seen next, many authors have identified issues that had to be dealt with when implementing DES control. These issues were also taken into account during the development of the testbed, which is described on Chapter 6.

One of the first works that addressed the problem of implementing the supervisory control theory (SCT) on manufacturing systems is (Balemi et al., 1993). The authors start by identifying the inconsistency of the assumption that the plant in the SCT generates events with real systems that produces responses as a consequence of commands received. In this sense, the plant can be seen as a process with input and output. The controllable events can be associated with the input while the uncontrollable ones with the plant output. Furthermore,

inputs are referred to as *commands* and outputs as *responses*. In this setup, both plant and supervisor generate events. Commands are generated by the supervisor while the responses are generated by the plants. The dynamics of the closed-loop system are described as follows. Let the plant and supervisor be understood as finite-state constructs. Out of the commands and responses from its current state, the supervisor *schedules* the commands to be transmitted to the plant. The plant schedules the responses from its internal state. Both plant and supervisor processes can be thought as *competing* for the first occurrence of one of the events they trigger. Both plant and supervisor behave like active scheduling units, while in the original model only the plant has that authority.

Balemi et al. (1993) propose a procedure for design of *controllers*, which are input-output supervisors. They also identify different types of responses produced by a plant, as well as different types of commands produced by controllers. The plant is not readily available as a logical plant model. It needs to be brought from the *physical* level where we deal with voltages and bits to a *logical* level that is suitable for a behavioral language specification. To accomplish this, they introduce the concept of interface, which is a component that is responsible to map the physical plant with its hardware, sensor and actuator routines into the logical plant. Under these assumptions, the feedback loop of SCT can be represented as in Fig. 3.4.

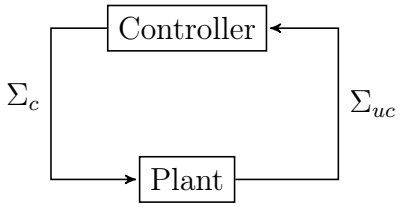


Figure 3.4: Input/Output interpretation of the feedback loop of supervisory control.

Some of the problems that arise from the implementation of supervisory control in Programmable Logic Controllers (PLC) were addressed in (Fabian and Hellgren, 1998). The authors identify the problems as:

- Events and signals: a PLC handles boolean valued signals, that are defined at all moments. In contrast, events only exist momentarily. Also, multiple signals can have their values changed within the duration of a single PLC scan, which can be interpreted as a simultaneous occurrence of events, which contradicts one of the basic assumptions of the DES theory. A property called *interleave insensitivity* is proposed and captures the requirement of the supervisor to generate the same event independently of the order in which events occur in the plant. With this property, the problem of simultaneous events can be remedied.

- Causality: it regards the problem of defining who generates what. As pointed out by Balemi et al. (1993), events in manufacturing systems are not generated spontaneously, but only as *responses* to given *commands*. Thus, the PLC generates the commands and the plants generates the responses which can be associated to controllable and uncontrollable events, respectively.

- Choice: the control action of the supervisor is a set of enabled events. However, only one command can be sent to the plant at a time, or otherwise other problems may arise. The problem of finding the best choice can be solved by scheduling techniques, for example (Pena et al., 2022; Alves et al., 2021).

- Inexact synchronization: this problem occurs when the plant generates an event while the PLC is computing the output and the new event generated by the plant invalidates it. To avoid this problem, a property called *delay insensitive language* was proposed by Balemi et al. (1993) and if met by a language, it assures that a controllable event is not preempted by an uncontrollable one.

The authors also give some programming directives that will avoid most of the aforementioned problems.

In (Dietrich et al., 2002), the authors tackle the problem of implementing supervisory control over a system. In manufacturing systems, to implement a controller which actually starts the machines it is not always sufficient to disable controllable events. Sometimes, the controller also has to choose exactly one event among the set of enabled ones. This is the case where controllable events chosen by the controller are interpreted as commands for the plant. In such a context, a problem that often arises is that the implemented controller may be blocking even if the given system is nonblocking. The authors start by defining a special supervisor, called an *implementation*, enabling at most one controllable event at a time. They also provide three system properties that, when met, guarantee that every implementation of the DES is nonblocking. The properties are:

- Termination: a terminating DES cannot have an infinite sequence consisting only of controllable events, otherwise, the system will block. In a terminating DES, after the occurrence of controllable events, the system will eventually stabilize, that is, will reach a state in which only uncontrollable events are possible;

- Confluence: this property ensures that independently of the choices taken, all implementation will reach states, by means of controllable events only, with the same future;

- Nonblocking under Control: it is a stronger definition of nonblocking. It states that for every string in the system, there exists a controllable continuation. A controllable

continuation is one in which uncontrollable events occur only if no controllable events are enabled.

In (De Queiroz and Cury, 2002) the authors propose a three-level structure for the implementation of supervisory control. A set of supervisors is obtained by the local modular approach (De Queiroz and Cury, 2000), followed by the application of the supervisor reduction technique (Vaz and Wonham, 1986), aiming to obtain small automata. The authors claim that this three-level structure is different from others in the literature since the supervisors can be implemented exactly as they are obtained.

Each state of the reduced supervisors has a control action defined as the disablement of a set of events. In basic terms, the implementation then consists of executing the automata for the supervisors in parallel, according to the events read from the real system, and of sending disabling signals to the plant, according to the current state of the automata.

However, the plant model used for supervisor synthesis is an abstraction of the real system behavior. Events representing commands launch operational procedures that update the control system output signals according to the devices' internal control logic. Uncontrollable events do not correspond directly to input signals and model logical events (as the end of an operation) generated by operational procedures. Thus, the control system has also to act as an interface between the abstract model and the real input/output signals.

To avoid the composition of the automata, a control system is proposed to be programmed in a three-level hierarchy, as shown in Fig. 3.5. The set of reduced local modular supervisors is implemented in the top level of the control system exactly as they are conceived. The program updates the active states of the supervisors according to their transition structures and to the events in the Product System level. A feedback map associates the active states to a set of disabling signals that control the Product System.

Since the supervisor automata may not have complete information of the plant behavior, their automata are implemented concurrently in the Product System level. The main function of this level is to execute commands by means of accepted controllable transitions. A controlled transition is considered accepted if its preceding state is active and the supervisors do not disable it. When multiple transitions in the Product System are accepted at the same time, the program needs to make a choice based on a priority scheme or to pass this degree of freedom to a higher level of decision.

The Operational Procedures level works as an interface between the theoretical Product System and the Real System. In this level, the program interprets the abstract commands

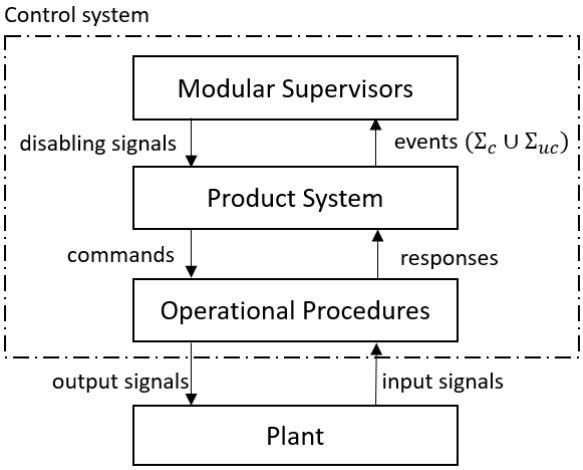


Figure 3.5: Control system architecture. From (De Queiroz and Cury, 2002).

from the Product System as logical procedures that guide the operation of each particular subsystem. These low-level procedures generate the control system output signals and read the input signals, supplying the Product System with logical responses that reflect the occurrence of uncontrollable events.

The three-level architecture is further investigated in (Vieira et al., 2006), where the authors present an approach to implement it in a PLC. Referring to Fig. 3.5, the Modular Supervisor level is composed of a set of local supervisors; the Product System by a set of structures named product system modules, each one associated with a real subsystem; the Operational Procedure level is composed of a set of structures named operational procedures. Generally, one operational procedure is uniquely associated with each event in the whole set of events; one disabling signal and one command are uniquely associated with each controllable event. One response is uniquely associated with each uncontrollable event.

In (Vieira et al., 2017), still based on the three-level implementation architecture, the authors distinguish two classes of problems while applying control in an industrial system. The first one is controlling each individual subsystem considering its own sensors, actuators and specialized controllers to perform sequences of activities. For this class of problems, there are well known solutions based on, among other type of devices, PLCs. The second class of problem is coordinating the concurrent operation of these subsystems to produce what is requested as efficiently as possible while guaranteeing the integrity and system's safety. The SCT is a formal approach particularly suited to this class of problem. In their work, the authors present a detailed method that allows a designer to systematically convert the results obtained from SCT to coordinate the concurrent operation of subsystems in a PLC application program.

Chapter 4

New test for P-observability for an attack set restricted to stealthy attackers

This chapter presents the first result of this thesis, which is the development of a test for a variation of a property called P-observability for an attack set. The results presented in this chapter are an improved version of the results published in (Alves et al., 2022a). The property P-observability for an attack set was introduced in (Wakaiki et al., 2019), in the context of supervisory control under partial observation and under the influence of cyber-attacks. Section 4.1 presents the attack model, while Section 4.2 presents the definition of P-observability for an attack set. Section 4.3 presents the main result of this chapter. Section 4.4 presents the related algorithms, while Section 4.5 discuss their time complexity. Finally, Section 4.6 presents a case study.

4.1 Attack Model

In this section, attackers whose goal is to prevent the supervisor from achieving the desired language $K = \mathcal{L}(S/G) \subseteq \mathcal{L}(G)$, which is prefix-closed, are considered. The attack model considered in here is different from the one considered in (Wakaiki et al., 2019). In this work, stealthy attackers are considered. A stealthy attacker chooses its actions in a way that whatever move the attacker does at this moment, will not be revealed immediately after that action that the attacker has attacked. Note that the attacker may be revealed in the future. In Wakaiki et al. (2019), this restriction does not exist. The attackers have full observation of the communication channel between plant and supervisor and can corrupt the string of output symbols $P(w)$, $w \in \Sigma^*$ in multiple ways, by erasing and/or inserting specific output symbols that are symbols sent from the plant to the supervisor, as an outcome of sensor readings. Also,

it is assumed that the supervisor sends a new control action to the plant whenever it receives new information.

Figure 4.1 represents a closed-loop controlled system under attack. The plant executes the string $w \in \mathcal{L}(G)$, but only the string $P(w)$ can be observed by the supervisor, because of the partial observation. Nevertheless, an attacker placed in the communication channel from the plant to the supervisor can corrupt the string $P(w)$ changing it to string $y \in \Sigma_o^*$. Upon reception of string y, the supervisor will then issue the control action $S(y)$. Depending on how the attacker chooses to modify $P(w)$, it can induce the supervisor to issue a control action that will make the plant reach an undesirable state. Next it is characterized how the attacker can modify $P(w)$.

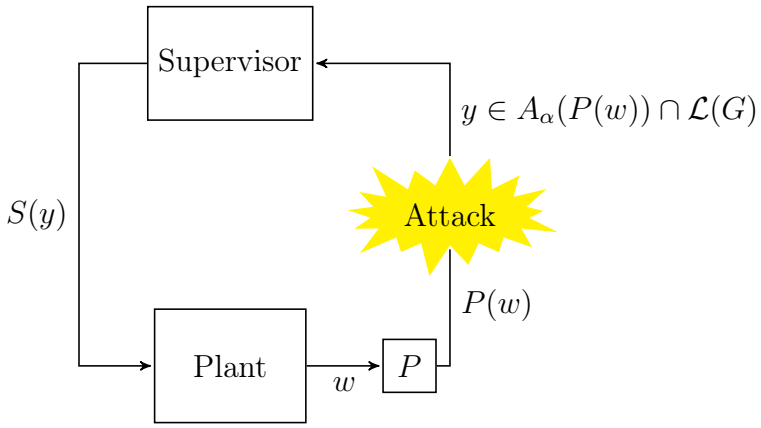


Figure 4.1: Closed loop controlled system under attacks (Adapted from Wakaiki et al. (2019)).

Given a set of symbols α[1]$\subseteq \Sigma_v \subseteq \Sigma_o$ in the observation alphabet, where Σ_v is the set of *vulnerable* events, the map $R_{\neg\alpha} : \Sigma \to (\Sigma \setminus \alpha) \cup \{\varepsilon\}$ is called *α-removal observation map* and is defined as

$$R_{\neg\alpha}(t) := \begin{cases} \varepsilon & t \in \alpha \\ t & t \notin \alpha \end{cases} \tag{4.1}$$

and can be extended to a map defined for strings of output symbols in the same way as a natural projection P. Note that the α-removal observation map is similar to a natural projection. An *observation attack* is a set-valued map $A_\alpha : \Sigma_o^* \to 2^{\Sigma_o^*}$ that assigns to each string $u \in \Sigma_o^*$ the set of all strings $v \in \Sigma_o^*$ that can be obtained from u by an arbitrary number of insertions or deletions of symbols in $\alpha \subseteq \Sigma_v$. It is given by

$$A_\alpha(u) := \{v \in \Sigma_o^* | R_{\neg\alpha}(u) = R_{\neg\alpha}(v)\}. \tag{4.2}$$

[1]In order to maintain consistency with the original attack model, proposed in (Wakaiki et al., 2019), in this work, the symbol α denotes a set.

If $\alpha = \emptyset$, then $A_\emptyset$ corresponds to the absence of attack.

Thus, instead of receiving the string $P(w)$, the supervisor receives one string among the set $A_\alpha(P(w))$. The map A_α is called an *observation attack* or simply *attacker* and the map $A_\alpha P : \Sigma^* \to 2^{\Sigma_o^*}$ obtained by the composition $A_\alpha P := A_\alpha \circ P$ the corresponding *attacked observation map*. In other words, the map $A_\alpha P(w)$ results in all strings that can be formed with the manipulation of events in $\alpha \subseteq \Sigma_o$, by inserting events into or erasing events from the observation $P(w)$. The goal of the attacker is, by changing the observation string, to make the supervisor accept a string that is not in K or to reject a string that is in K. The attack can be also be interpreted as a factor that increases the supervisor's uncertainty about which state the plant is really in.

The next example shows the application of the attacked observation map over a simple string, for two different scenarios.

Example 2. *Let* $\Sigma_o = \{a, b, c\}$ *and* $u = abc$. *For* $\alpha_1 = \{a\}$, $A_{\alpha_1}(u) = \{a\}^* b \{a\}^* c \{a\}^*$, *and for* $\alpha_2 = \{a, b\}$, $A_{\alpha_2}(u) = \{a, b\}^* c \{a, b\}^*$. □

The reader can observe, from Example 2, that the map A_α is similar to an inverse projection. Nevertheless, there are two main differences. The first one is the fact that the domain and codomain of map A_α are defined over the same set, while in the inverse projection, the domain is defined over a set that can be smaller than the codomain. The second difference is related to the fact that the map A_α can describe the effect of erasing events, in addition to the effect of adding events, whereas the inverse projection can only insert events.

At this point, it is important to emphasize the dynamics of the system. The plant doesn't send to the supervisor the events in string w all at once. At the initial state, $w = \varepsilon$, since nothing was executed in the plant yet. If there was no attack, then the supervisor would receive $P(\varepsilon) = \varepsilon$ and then issue the control action $S(\varepsilon)$. The plant would execute an event $\sigma \in S(\varepsilon)$, meaning now that $w = \sigma$. The supervisor would see the observation $P(\sigma)$, issue the control action $S(P(\sigma))$ and the process repeats indefinitely. In other words, the string w is executed event by event. Although the map A_α is able to insert or erase any event $\sigma \in \alpha$ along the string w, the insertions or erasures will only happen at the end of the string. Inserting or erasing an event at the beginning of the string is not possible, as it would mean that the attacker modified an event that is in the past.

When the attack is considered, as $w = \varepsilon$ at the initial state, there will be nothing for the attacker to intercept. At this moment, the attacker can insert any event in α, including none, and send it to the supervisor. To illustrate, suppose that $\sigma_1 \in \alpha$ was inserted by the attacker. Upon reception of σ_1, the supervisor updates its state and issue the control action

$S(\sigma_1)$. Observe that the plant is still at the initial state, but the supervisor "thinks" it has executed σ_1. The plant can now execute an event $\sigma_2 \in S(\sigma_1)$, if $\delta(q_0, \sigma_2)!$, which generates the observation $P(\sigma_2)$. The attacker can intercept this observation and erase it if $\sigma_2 \in \alpha$, and/or insert another event $\sigma_3 \in \alpha$, or even let the observation remain untouched.

In a sense, the attacker achieves its goal of making the supervisor accept a string that is not in K or to reject a string that is in K, by desynchronizing the plant and the supervisor. Furthermore, if $w \in K$ was executed in the plant until now, then it is assumed that the attacker has knowledge about the system and will not insert an event $\sigma \in \alpha$ in the observation if $P(w)\sigma \notin K$, since that would easily reveal the attackers presence if an intrusion detection system were in place. This is illustrated in the next example.

Example 3. *Consider the automaton H of Fig. 4.2 and a language K such that $\mathcal{L}(H) = K$, where $\Sigma_v = \{u, x\}$. If $w = \varepsilon$, $\sigma = u$ and $\sigma' = a$, then $P(w)\sigma = u \in K$, $P(w)\sigma' = a \in K$ but $P(w)\sigma\sigma' = ua \notin K$. In other words, if an attacker inserts event u into the supervisor's observation at the initial state, the supervisor will update its state estimate to state 3. Then, upon the occurrence of event a, which is not vulnerable and is feasible at the plant's current state, the supervisor would see a string (i.e., ua) that is not in K. Thus, the attacker's presence would be revealed.*

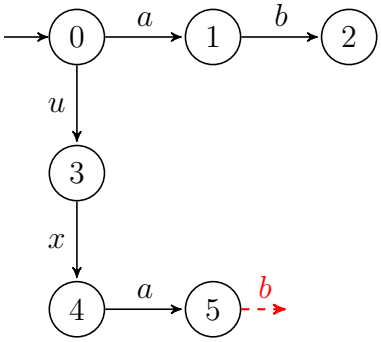


Figure 4.2: Automaton of Example 3.

□

Even when the plant and the supervisor are desynchronized as a consequence of insertion or erasure of events, the system keeps working, unless the attacker is successful, i.e., it makes the plant reach an undesirable state.

A more interesting scenario consists of multiple attackers, instead of only one. These multiple attackers are represented by an *attack set*, denoted by $\mathcal{A}$, such that $\mathcal{A} = \{A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_M}\}$, where $\alpha_i \subseteq \Sigma_v \subseteq \Sigma_o$, for $i = 1, \ldots, M$.

It is assumed that only one of the attackers in an attack set $\mathcal{A}$ is acting, but the information about which one is not available. Thus, all possible scenarios have to be considered when trying to protect the system from the attacks.

The authors of (Wakaiki et al., 2019) present two conditions for a desired language K that will allow the design of a supervisor that can enforce it, regardless of which attacker in the attack set $\mathcal{A}$ is acting. The language K must be 1) controllable with respect to $\mathcal{L}(G)$ and 2) P-observable for the attack set $\mathcal{A}$.

The P-observability for an attack set is an extended version of the classical P-observability and is detailed in the next section.

4.2 P-Observability for an attack set

A prefix-closed language $K \subseteq \mathcal{L}(G)$ is P-observable *for an attack set* $\mathcal{A}$, if

$$R_{A,\overline{A}} \subset \text{act}_{K \subset L} \tag{4.3}$$

where the relation $R_{A,\overline{A}}$ contains all pairs of strings that may result in attacked observation maps AP and $\overline{A}P$ with a common string of output symbols, i.e.,

$$R_{A,\overline{A}} := \{(w, w') \in \Sigma^* \times \Sigma^* | AP(w) \cap \overline{A}P(w') \neq \emptyset\} \tag{4.4}$$

and $\text{act}_{K \subset L}$ is defined in (3.5).

Thus, P-observability for an attack set is related to the supervisor's ability to always be able to differentiate strings that require different control actions. In the remainder of this work, unless said otherwise, the term "P-observability" and "P-observability for an attack set" will be used as equivalents.

The work in which the P-observability for an attack set was first introduced also presents a result that gives the conditions for the existence of a supervisor that, regardless of which attacker $A_\alpha \in \mathcal{A}$ is acting, can enforce a desired language K. These conditions are summarized in the following theorem.

Theorem 1 ((Wakaiki et al., 2019))**.** *For every nonempty prefix-closed set $K \subset \mathcal{L}(G)$ and every attack set $\mathcal{A}$:*

i. there exists a solution f to the supervision of K under the attack set $\mathcal{A}$ if and only if K is controllable and P-observable for $\mathcal{A}$;

ii. if K is controllable and P-observable for $\mathcal{A}$, the map $f : \bigcup_{A_\alpha \in \mathcal{A}} A_\alpha P(\mathcal{L}(G)) \to \Sigma$ defined by

$$f(y) := \Sigma_{uc} \cup \{\sigma \in \Sigma_c | \exists w \in K, A_\alpha \in \mathcal{A} \text{ s.t. } [y \in A_\alpha P(w), w\sigma \in K]\}, \quad \forall y \in \bigcup_{A_\alpha \in \mathcal{A}} A_\alpha P(\mathcal{L}(G)) \tag{4.5}$$

is a solution to the supervision of K under the attack set $\mathcal{A}$.

♦

The fact that controllability of K is a necessary condition for the existence of a supervisor is to be expected. Regarding the P-observability for an attack set $\mathcal{A}$, if this condition does not hold, it is possible to find two attacks $A_{\alpha_i}, A_{\alpha_j} \in \mathcal{A}$ that would result in the same observation $y \in \Sigma_o^*$ for two distinct strings $w_1, w_2 \in K$, and w_1 would transition to an element in K and w_2 to an element outside K, or vice-versa. This means that, upon observing y, the supervisor cannot decide which control action to take. Contrarily, if K is controllable and P-observable for the attack set $\mathcal{A}$, then there exists a supervisor that enforces K and such policy is given by (4.5).

The next theorem shows how the test for P-observability for an attack set $\mathcal{A}$ can be done by reducing it to a classical observability test.

Theorem 2 ((Wakaiki et al., 2019)). *For every nonempty prefix-closed set $K \subseteq \mathcal{L}(G)$ and attack set $\mathcal{A} = \{A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_M}\}$ consisting of $M \geq 1$ observation attacks, K is P-observable for the set of attacks $\mathcal{A}$ if and only if K is $(R_{\neg\alpha} \circ P)$-observable (in the classical sense, i.e., without attacks) for every set $\alpha := \alpha_i \cup \alpha_j$, $\forall i, j \in \{1, 2, \ldots, M\}$.* ♦

Theorem 2 states that P-observability for an attack set $\mathcal{A}$ can be tested by picking every possible pair of two attackers A_{α_i}, A_{α_j} and checking if regular observability is obtained if the symbols affected by the two attackers are removed from the observation. The authors claim that, using the test for classical observability presented in (Cassandras and Lafortune, 2007, Section 3.7.3), P-observability for an attack set can be tested with time complexity $O(M^5)$, where $M = |\mathcal{A}|$. However, this complexity was obtained from a very specific attack set and a generalization was not provided.

4.3 New test for P-observability restricted to stealthy attackers

This section presents a new test for P-observability for an attack set when the attackers are required to be stealthy. Rather than executing multiple tests of classical observability,

the proposed test checks P-observability for an attack set itself. The test is applied over an automaton that implements a desired language K.

Before presenting the new test, some definitions are introduced. Firstly, how the attacks can be interpreted visually is shown, through an example.

Example 4. *Consider a system whose representation is the automaton H of Fig. 4.3, where $\Sigma_o = \{u, x, y, z, e\}$, $\Sigma_v = \{u, x, y, z\}$ and $\mathcal{A} = \{A_\emptyset, A_{\{u,x\}}, A_{\{y,z\}}\}$.*

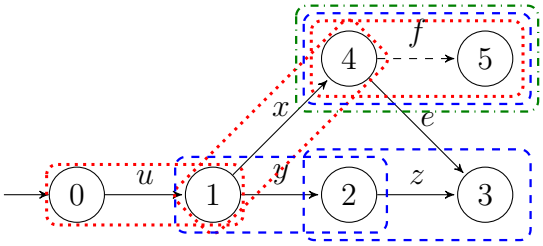


Figure 4.3: Automaton H of Example 4. The rectangles with dashed-dotted (green), dotted (red) and dashed (blue) borders represent the effect of the attackers $A_\emptyset$, $A_{\{u,x\}}$ and $A_{\{y,z\}}$, respectively.

Attacker $A_\emptyset$ represents the absence of attack. This means that the only cause of confusion about which state the system is at is due to the partial observation. This is represented by the dashed-dotted rectangle involving states 4 and 5. Now consider attacker $A_{\{u,x\}}$. If an observer sees string $y = u$, it cannot be sure if the system is really at state 1 or at state 0 or at state 4. This is because event u may never have happened and was inserted by the attacker or event u has happened and was not modified by the attacker or events u and x have happened but x was erased by the attacker. Thus, state 0 can be confused with state 1 and state 1 can be confused with state 4, which is represented by the dotted rectangles encircling states 0 and 1 and also 1 and 4. In addition, as event f is unobservable, the observer can confuse states 4 and 5, which is represented by the dotted rectangle surrounding these states. The same reasoning can be applied to find the dashed rectangles around states 1 and 2, 2 and 3 and again 4 and 5, regarding attacker $A_{\{y,z\}}$. Independently of which attacker is considered, an observer can always confuse a state q with itself. The visual representation for such cases was omitted. □

An attack can increase the uncertainty for an observer about which state the plant is in, by manipulating the symbols in the observation. This uncertainty can be represented as a relation of pairs of states and each attacker induces a different one. In the previous example, each pair of states involved by a rectangle is composed of states that can be confused with each other, which is due to the attack or to the partial observation. Thus, each rectangle gives rise to a pair of states in the relation of states that can be confused with each other and with respect to an attacker, where the pair order is given by the transition connecting both states. One can easily conclude that if the pairs (q_1, q) and (q, q_2) are in the relation, for $(q_1, \sigma, q), (q, \sigma', q_2) \in \Delta$,

then the pair (q_1, q_2) should also be, since the attacker can insert or erase both events. In other words, this relation is transitive. The following definition formalizes a binary relation on the set of states of an automaton induced by an attacker. Henceforward, this relation, denoted by Π_α, will be called the relation of *indistinguishable* states with respect to attacker $A_\alpha \in \mathcal{A}$.

Definition 5 (Indistinguishable states with respect to attacker A_α)**.** *For a given attacker $A_\alpha \in \mathcal{A}$ and automaton G, the relation $\Pi_\alpha \subseteq Q \times Q$ defined as*

$$\Pi_\alpha := \{(q, q') \in Q \times Q | (\forall wv \in \mathcal{L}(G))[\delta(q_0, w) = q \wedge \delta(q_0, wv) = q' \wedge v \in (\alpha \cup \Sigma_{uo})^*]\} \quad (4.6)$$

is the relation of indistinguishable states with respect to attacker A_α. $\Diamond$

According to Def 5, the relation Π_α has all pairs of states (q, q') such that state q' is reachable from state q with events in α, the events that attacker A_α can manipulate, or with events in Σ_{uo}. This definition was inspired by (Wang et al., 2007), where the pairs of indistinguishable states arise due only to partial observation.

Definitions 6 and 7 formalize other possible ways obtain new pairs of indistinguishable states. These definitions are very similar. The first one considers pairs of states that are connected by vulnerable events (pairs of states obtained in the relation according to Def. 5).

Definition 6 (Indistinguishable states due to a common state)**.** *The relation $\widehat{\Pi}(\Pi_\alpha) \subseteq Q \times Q$ defined as*

$$\widehat{\Pi}(\Pi_\alpha) = \{(q_1, q_2), (q_2, q_1) \in Q \times Q | (\exists q \in Q)(q \neq q_1 \wedge q \neq q_2)[(q, q_1), (q, q_2) \in \Pi_\alpha]\} \cup \Pi_\alpha, \quad (4.7)$$

is the relation of indistinguishable states due to a common state. $\Diamond$

The second one is later used when new pairs of indistinguishable states that are not directly connected via vulnerable events are created. Because now the states do not necessarily need to be connected by vulnerable events, sometimes non-vulnerable events help to make them distinguishable and thus, these states should not be combined.

Definition 7 (Indistinguishable states due to a common state after continuations)**.** *Let G be an automaton and $Obs(G) = (X, \Sigma_o \setminus \Sigma_v, \delta', x_0)$ its observer, obtained after the removal of all unobservable and vulnerable events. The relation $\widehat{\Pi}'(\Pi_\mathcal{A}) \subseteq Q \times Q$ defined as*

$$\widehat{\Pi}'(\Pi_\mathcal{A}) = \{(q_1, q_2), (q_2, q_1) \in Q \times Q | (\exists q \in Q)(q \neq q_1 \wedge q \neq q_2)(\exists x \in X \text{ such that } q_1, q_2 \in x) [(q, q_1), (q, q_2) \in \Pi_\mathcal{A}]\} \cup \Pi, \quad (4.8)$$

is the relation of indistinguishable states due to a common state after continuations. ◊

In Definitions 6 and 7, whenever there is a state q that is indistinguishable with states q_1 and q_2 (i.e., both q and q_1 are related and q and q_2 are related), then q_1 and q_2 are also related. This means that, upon observation of a string that leads to state q, the actual current state can be q, q_1 or q_2. Definition 7 requires an additional condition: the pairs are combined only if both q_1 and q_2 does not belong to the set Q_{DS}.

Two states that cannot be reached from each other can also be indistinguishable. This happens when they are reached from a pair of indistinguishable states with continuations that have the same projection when erasing vulnerable and unobservable events. In other words, the uncertainty about the current state propagates to a new pair that is reached with continuations that have the same projection. This is captured by the following definition.

Definition 8 (Indistinguishable states due to identical observable continuations)**.** *For a given automaton G and a relation of pairs of indistinguishable states Π, the map $\Pi^{\rightarrow} : (Q \times Q) \rightarrow (Q \times Q)$ defined as*

$$\Pi^{\rightarrow}(\Pi_\alpha) := \{(q, q') \in Q \times Q | (\exists u, v \in (\Sigma_o \setminus \alpha)^*)(\exists (q_1, q_2) \in \Pi_\alpha)[P_{\Sigma \rightarrow \Sigma_o \setminus \alpha}(u) = P_{\Sigma \rightarrow \Sigma_o \setminus \alpha}(v)] \\ [q = \delta(q_1, u) \wedge q' = \delta(q_2, v)]\} \tag{4.9}$$

is the relation of indistinguishable states due to identical observable continuations. The operator $P_{\Sigma \rightarrow \Sigma_o \setminus \alpha}(u)$ is the natural projection of a given string u from alphabet Σ to $\Sigma_o \setminus \alpha$. ◊

According to Def. 8, if a pair of states (q_1, q_2) belongs to a relation of indistinguishable states Π_α and if there exists strings $u, v \in (\Sigma_o \setminus \Sigma_v)^*$ such that $\delta(q_1, u) = q$, $\delta(q_2, v) = q'$ and, in addition, $P_{\Sigma \rightarrow \Sigma_o \setminus \alpha}(u) = P_{\Sigma \rightarrow \Sigma_o \setminus \alpha}(v)$, then $(q, q') \in \Pi^{\rightarrow}(\Pi_\alpha)$. Note that $\Pi_\alpha \subseteq \Pi^{\rightarrow}(\Pi_\alpha)$, since $u = \varepsilon \in (\Sigma_o \setminus \Sigma_v)^*$ and $v = \varepsilon \in (\Sigma_o \setminus \Sigma_v)^*$ are always a possible continuation from a pair of indistinguishable states.

Another way to form new pairs of indistinguishable states is by considering that two attackers can cooperate with each other, even though they are not allowed to do so. This hypothetical assumption is useful because if two attackers acting cooperatively can make a pair of states indistinguishable, then it means that they can produce (by adding or removing events) a common observation starting from two different strings, which can potentially violate the P-observability for an attack set. For example, in Fig 4.3, states 0 and 2 are indistinguishable. To understand this, consider that string u was observed. This observation could have been the result of the insertion of event u at the initial state by attacker $A_{\{u,x\}}$ or the result of the erasure of event y by attacker $A_{\{y,z\}}$ after the occurrence of string uy. Another example can be given with respect to states 2 and 4. If string u is observed, then there is no way to know if string

ux happened and x was erased by attacker $A_{\{u,x\}}$ or string uy happened and y was erased by attacker $A_{\{y,z\}}$. More formally, the way two attackers can cooperate is formalized in Definition 9.

The following definition captures the relation of indistinguishable states after pairwise combined attacks, which is the second way two attackers can cooperate with each other.

Definition 9 (Indistinguishable states due to transitivity)**.** *The relation* $\Pi^{\circ}_{\mathcal{A}} \subseteq Q \times Q$ *defined as*

$$\Pi^{\circ}_{\mathcal{A}} = \{(q_1, q_2), (q_2, q_1) \in Q \times Q | (q_1, q), (q, q_2) \in \Pi_{\mathcal{A}}\}$$

where $\Pi_{\mathcal{A}}$ *is the relation of indistinguishable states considering all the attackers in the attack set* $\mathcal{A}$*.* $\Diamond$

In Def. 9, if (q_1, q) and (q, q_2) are pairs of indistinguishable states due to the action of two attackers, then the pairs (q_1, q_2) and (q_2, q_1) are also indistinguishable. Thus if a string that leads to state q is observed, then it is not possible to be sure if the current state is q_1, q or q_2. Note that for each pair $(q_1, q_2) \in \Pi_{\mathcal{A}}$, then the symmetric pair (q_2, q_1) will also be added to $\Pi^{\circ}_{\mathcal{A}}$, since $(q_1, q_1), (q_1, q_2) \in \Pi_{\mathcal{A}}$.

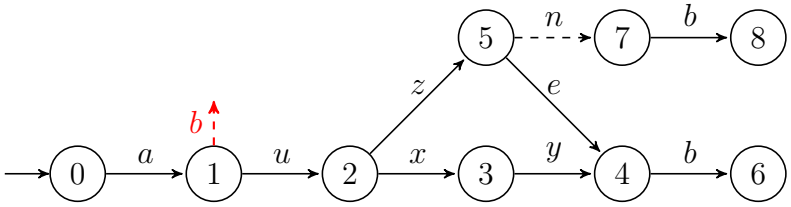
Using Definitions 5-9, one can find all possible pairs of indistinguishable states with respect to an attack set $\mathcal{A}$. The procedure is summarized in the next algorithm.

Algorithm 1: Indistinguishable states with respect to attack set $\mathcal{A}$.

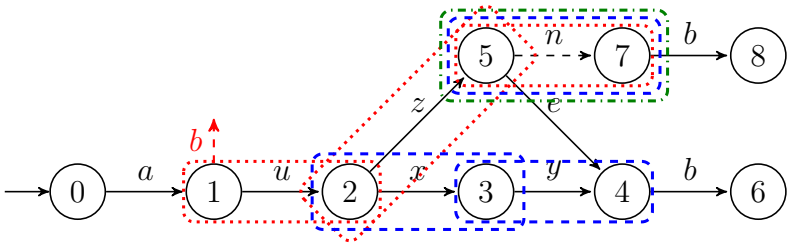
Input: $H = (Q^H, \Sigma, \delta^H, q_0^H), \mathcal{A}$
Result: $\Pi^2_{\mathcal{A}}$
1 **foreach** $A_\alpha \in \mathcal{A}$ **do**
2 | OBTAIN $\widehat{\Pi}(\Pi_\alpha)$
3 | $\Pi'_\alpha \leftarrow \Pi^{\rightarrow}(\widehat{\Pi}(\Pi_\alpha))$
4 $\Pi_{\mathcal{A}} \leftarrow \bigcup_{A_\alpha \in \mathcal{A}} \Pi'_\alpha$
5 $\Pi^2_{\mathcal{A}} \leftarrow \Pi^{\circ}_{\mathcal{A}} \cup \widehat{\Pi}'(\Pi_{\mathcal{A}})$
6 **return** $\Pi^2_{\mathcal{A}}$

The next examples highlight some characteristics of Alg. 1. Example 5 considers the case of an attack set $\mathcal{A}$ composed by three attackers, one of which represents the absence of the attack.

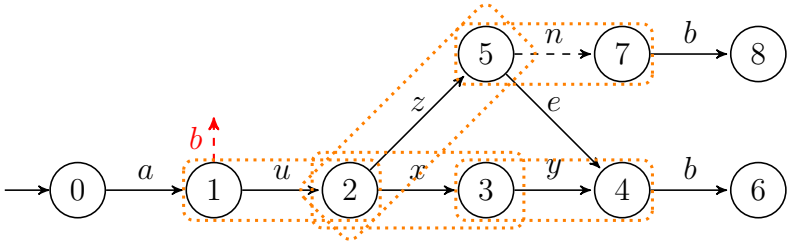
Example 5. *Consider the automaton* H *shown in Fig. 4.4(a), where* $\Sigma_v = \{u, x, y, z\}$*,* $\Sigma_{uo} = \{n\}$ *and* $\mathcal{A} = \{A_\emptyset, A_{\{u,z\}}, A_{\{x,y\}}\}$*. The red arrow represents a disablement. In order to find the relation of indistinguishable states* $\Pi^2_{\mathcal{A}}$*, Alg. 1 can be applied.*



(a) Automaton H



(b) lines 1-3



(c) line 4

Figure 4.4: Automata of Example 5.

After the execution of lines 1-3, the relations Π_α *obtained are:*

$$\begin{aligned}
\Pi'_{\emptyset} &= \{(5,7)\} \cup \{(q,q) | 0 \leq q \leq 8\} \\
\Pi'_{\{u,z\}} &= \{(1,2),(1,5),(1,7),(2,5),(2,7),(5,7)\} \cup \{(q,q) | 0 \leq q \leq 8\} \\
\Pi'_{\{x,y\}} &= \{(2,3),(2,4),(3,4),(5,7)\} \cup \{(q,q) | 0 \leq q \leq 8\}
\end{aligned}$$

A visual representation of these relations can be seen in Fig. 4.4(b), where rectangles around related pairs (q,q) *are omitted to reduce clutter.*

The next step takes the union of all relations Π'_α*, resulting in the relation* $\Pi_{\mathcal{A}}$*, which can be visually represented as in Fig. 4.4(c).*

$$\Pi_{\mathcal{A}} = \{(1,2),(1,5),(1,7),(2,3),(2,4),(2,5),(2,7),(3,4),(5,7)\} \cup \{(q,q) | 0 \leq q \leq 8\}$$

By looking at Fig. 4.4(b), one can see that the pairs $(3,5)$ *and* $(5,3)$ *must be added to* $\Pi^2_{\mathcal{A}}$ *since states 2 and 5 are in the same red (dotted) rectangle and states 2 and 3 are in the same blue (dashed) rectangle, i.e.,* $(5,2)$ *and* $(2,3)$ *are in* $\Pi_{\mathcal{A}}$*. Similar reasoning can be used to explain why*

the other pairs in $\Pi^2_{\mathcal{A}}$ *were added. Then, in line 5, the relations* $\Pi^\circ_{\mathcal{A}}$ *and* $\widehat{\Pi}'(\Pi_{\mathcal{A}})$ *are obtained:*

$$\Pi^\circ_{\mathcal{A}} = \{(1,3),(1,4),(2,1),(3,1),(3,2),(4,1),(4,2),(4,3),(5,1),(5,2),(7,1),(7,2),(7,5)\} \cup \Pi_{\mathcal{A}}.$$

$$\widehat{\Pi}'(\Pi_{\mathcal{A}}) = \{(1,7),(2,5),(3,4),(3,5),(3,7),(4,3),(4,5),(4,7),(5,2),(5,3),(5,4),(5,7),(7,1),(7,3),(7,4),(7,5)\} \cup \Pi_{\mathcal{A}}.$$

Note that for this example, the observer states are $X = \{\{0\},\{1,2,3,4,5,7\},\{4\},\{6\},\{6,8\}\}$. *Finally, the relation* $\Pi^2_{\mathcal{A}}$ *is obtained:*

$$\Pi^2_{\mathcal{A}} = \{(1,2),(1,3),(1,4),(1,5),(1,7),(2,1),(2,3),(2,4),(2,5),(2,7),(3,1),(3,2),(3,4),(3,5),(3,7),(4,1),(4,2),(4,3),(4,5),(4,7),(5,1),(5,2),(5,3),(5,4),(5,7),(7,1),(7,2),(7,3),(7,4),(7,5)\} \cup \{(q,q)|0 \leq q \leq 8\}$$

Note that, although states 4 and 7 are indistinguishable and there are continuations that have the same projection when erasing vulnerable and unobservable events from both of them, which reaches states 6 and 8, the pairs $(6,8)$ *and* $(8,6)$ *are not added to* $\Pi^2_{\mathcal{A}}$. *This happens because if string auxyb is observed, there is no way the string auznb could be the one that actually happened, since it would require the cooperation of two attackers (attacker* $A_{\{u,z\}}$ *would have to erase* z *and then attacker* $A_{\{x,y\}}$ *would have to insert* x *and* y*) and an assumption of the problem is that only one attacker attacks.*

□

Example 6 shows a case where two states become indistinguishable not because they are connected through a transition labeled with a vulnerable or non-observable event, but because those states can be reached from states that are already indistinguishable via continuations that have the same projection when erasing vulnerable and unobservable events. Such pair of states can be further combined in order to represent the hypothetical cooperation between attackers.

Example 6. *Consider the automaton* H *shown in Fig. 4.5, where* $\Sigma_v = \{u,x\}$ *and* $\mathcal{A} = \{A_{\{u\}}, A_{\{x\}}\}$. *The red arrow represents a disablement. In order to find the relation of indistinguishable states* $\Pi^2_{\mathcal{A}}$, *Alg. 1 can be applied.*

After the execution of lines 1-3 of Alg. 1, the relations Π'_α *obtained are as follows. For the relation* $\Pi'_{\{u\}}$, *pairs* $(0,4)$ *and* $(4,0)$ *were added in line 2 of Alg. 1 while pairs* $(1,5)$ *and* $(5,1)$ *were added in line 3 of Alg. 1.*

$$\Pi'_{\{u\}} = \{(0,4),(1,5)\} \cup \{(q,q)|0 \leq q \leq 5\}$$
$$\Pi'_{\{x\}} = \{(1,2)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

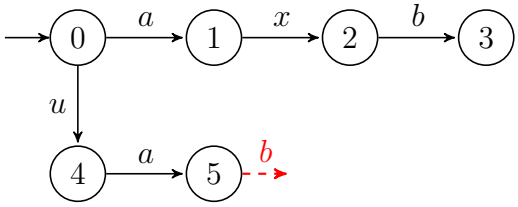


Figure 4.5: Automaton of Example 6.

The next step takes the union of all relations Π'_α*, resulting in the relation* $\Pi_\mathcal{A}$*.*

$$\Pi_\mathcal{A} = \{(0,4),(1,2),(1,5)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

Then, in line 5, the relations $\Pi^\circ_\mathcal{A}$ *and* $\widehat{\Pi}'(\Pi_\mathcal{A})$ *are obtained:*

$$\Pi^\circ_\mathcal{A} = \{(2,1),(4,0),(5,1)\} \cup \Pi_\mathcal{A}.$$
$$\widehat{\Pi}'(\Pi_\mathcal{A}) = \{(2,5),(5,2)\} \cup \Pi_\mathcal{A}$$

Note that for this example, the observer states are $X = \{\{0,4\},\{1,2,5\},\{3\}\}$*. Finally, the relation* $\Pi^2_\mathcal{A}$ *is obtained:*

$$\Pi^2_\mathcal{A} = \{(0,4),(1,2),(1,5),(2,1),(2,5),(4,0),(5,1),(5,2)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

For this example, note that states 2 and 5 are indistinguishable. Upon the observation of string a*, it is not possible to know if the current state is really state 1, or if string* ax *happened and* x *was erased by attacker* $A_{\{x\}}$ *or if string* ua *happened and attacker* $A_{\{u\}}$ *erased event* u*.* □

The next example presents a case where two states are indistinguishable and there are continuations that have the same projection when erasing vulnerable and unobservable events from both of them and yet, the states reached by this identical continuation are not indistinguishable.

Example 7. *Consider the automaton* H *shown in Fig. 4.6, where* $\Sigma_v = \{u,x\}$ *and* $\mathcal{A} = \{A_{\{u\}}, A_{\{x\}}\}$*. The red arrow represents a disablement. In order to find the relation of indistinguishable states* $\Pi^2_\mathcal{A}$*, Alg. 1 can be applied.*

After the execution of lines 1-3, the relations Π'_α *obtained are:*

$$\Pi'_{\{u\}} = \{(0,3)\} \cup \{(q,q)|0 \leq q \leq 5\}$$
$$\Pi'_{\{x\}} = \{(3,4)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

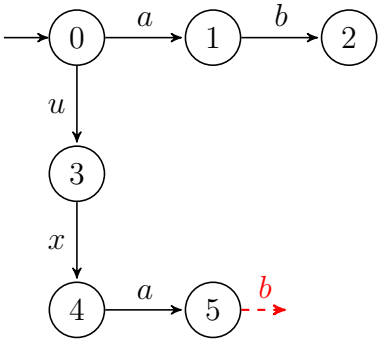


Figure 4.6: Automaton of Example 7.

The next step takes the union of all relations Π'_α*, resulting in the relation* $\Pi_\mathcal{A}$*.*

$$\Pi_\mathcal{A} = \{(0,3),(3,4)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

Then, in line 5, the relations $\Pi^\circ_\mathcal{A}$ *and* $\widehat{\Pi}'(\Pi_\mathcal{A})$ *are obtained:*

$$\Pi^\circ_\mathcal{A} = \{(3,0),(4,3)\} \cup \Pi_\mathcal{A}$$
$$\widehat{\Pi}'(\Pi_\mathcal{A}) = \Pi_\mathcal{A}.$$

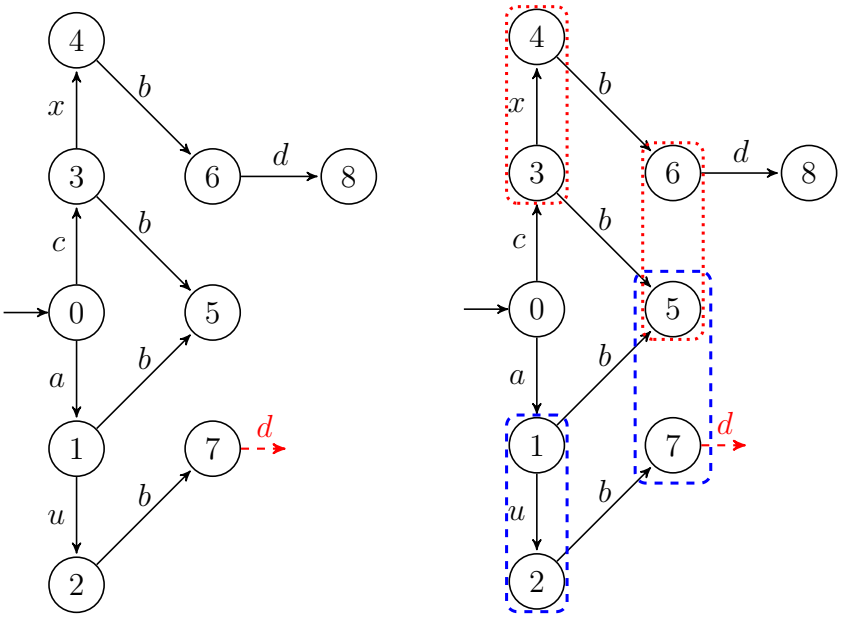
Note that for this example, the observer states are $X = \{\{0,3,4\},\{1,5\},\{2\}\}$*. Finally, the relation* $\Pi^2_\mathcal{A}$ *is obtained:*

$$\Pi^2_\mathcal{A} = \{(0,3),(0,4),(3,0),(3,4),(4,0),(4,3)\} \cup \{(q,q)|0 \leq q \leq 5\}$$

Note that in this example, although there exists an identical continuation starting from a pair of indistinguishable states, 0 and 4, states 1 and 5 are not indistinguishable. This is because the pair $(0,4)$ *was generated in the relation* $\Pi^\circ_\mathcal{A}$*, which does not take continuations that have the same projection into account. This is justified by the fact that the attackers are considered to be stealthy, which means that the attacker* $A_{\{u\}}$ *will not insert or delete event* u*, because if either* a *or* x *happens next, the string observed will not be in* $\mathcal{L}(H)$*, revealing the attacker's presence.* □

The next example presents a case where the set Q_{DS} is not empty. This fact will have impact on relation $\widehat{\Pi}_\mathcal{A}$.

Example 8. *Consider the automaton* H *shown in Fig. 4.7, where* $\Sigma_v = \{u,x\}$ *and* $\mathcal{A} = \{A_{\{u\}}, A_{\{x\}}\}$*. The red arrow represents a disablement. In order to find the relation of indistinguishable states* $\Pi^2_\mathcal{A}$*, Alg. 1 can be applied.*



(a) Automaton H (b) lines 1-3

Figure 4.7: Automata of Example 8.

After the execution of lines 1-3, the relations Π'_α *obtained are:*

$$\Pi'_{\{u\}} = \{(1,2),(5,7)\} \cup \{(q,q)|0 \leq q \leq 8\}$$
$$\Pi'_{\{x\}} = \{(3,4),(5,6)\} \cup \{(q,q)|0 \leq q \leq 8\}$$

The next step takes the union of all relations Π'_α*, resulting in the relation* $\Pi_\mathcal{A}$*.*

$$\Pi_\mathcal{A} = \{(1,2),(3,4),(5,6),(5,7)\} \cup \{(q,q)|0 \leq q \leq 8\}$$

In line 5 the relations $\Pi^\circ_\mathcal{A}$ *and* $\widehat{\Pi}'(\Pi_\mathcal{A})$ *are obtained:*

$$\Pi^\circ_\mathcal{A} = \{(2,1),(4,3),(6,5),(7,5)\} \cup \Pi_\mathcal{A}$$
$$\widehat{\Pi}'(\Pi_\mathcal{A}) = \Pi_\mathcal{A}$$

Note that for this example, the observer states are $X = \{\{0\},\{1,2\},\{3,4\},\{5,6\},\{5,7\},\{8\}\}$*. Since states 7 and 6 does not belong to the same state in the observer, then the pairs (5, 6) and (5, 7) are not going to be combined. As a result,*

$$\Pi^2_\mathcal{A} = \{(1,2),(2,1),(3,4),(4,3),(5,6),(5,7),(6,5),(7,5)\} \cup \{(q,q)|0 \leq q \leq 8\}$$

□

The next example shows a case where pairs of states are created by Def. 6, at line 2 of Alg. 1.

Example 9. *Consider the automaton H shown in Fig. 4.8, where $\Sigma_v = \{u, x, y\}$, and $\mathcal{A} = \{A_{\{u,x\}}, A_{\{y\}}\}$. The red arrow represents a disablement. In order to find the relation of indistinguishable states $\Pi^2_{\mathcal{A}}$, Alg. 1 can be applied.*

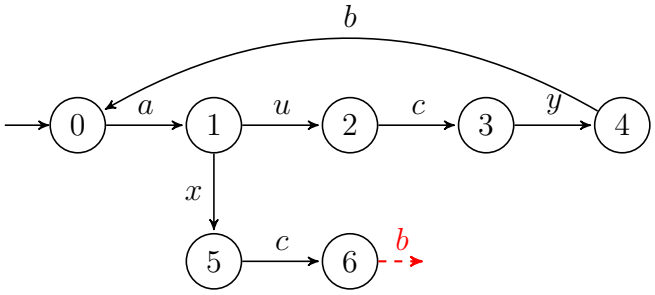


Figure 4.8: Automaton of Example 9.

After the execution of lines 1-3 of Alg. 1, the relations Π'_α obtained are:

$$\Pi'_{\{u,x\}} = \{(1,2),(1,5),(2,5),(3,6),(5,2),(6,3)\} \cup \{(q,q)|0 \leq q \leq 6\}$$
$$\Pi'_{\{y\}} = \{(3,4)\} \cup \{(q,q)|0 \leq q \leq 6\}$$

The pairs $(2,5)$ and $(5,2)$ were obtained in $\widehat{\Pi}(\Pi_{\{u,x\}})$ (line 2). Then, because states 2 and 5 have continuations that have the same projection when erasing vulnerable and unobservable events, the pairs $(3,6)$ and $(6,3)$ are obtained in $\Pi^{\rightarrow}(\widehat{\Pi}(\Pi_{\{u,x\}}))$ (line 3). The next step takes the union of all relations Π'_α, resulting in the relation $\Pi_{\mathcal{A}}$.

$$\Pi_{\mathcal{A}} = \{(1,2),(1,5),(2,5),(3,4),(3,6),(5,1),(5,2),(6,3)\} \cup \{(q,q)|0 \leq q \leq 6\}$$

In line 5 the relations $\Pi^\circ_{\mathcal{A}}$ and $\widehat{\Pi}'(\Pi_{\mathcal{A}})$ are obtained:

$$\Pi^\circ_{\mathcal{A}} = \{(1,5),(2,1),(2,5),(3,6),(4,3),(4,6),(5,1),(5,2),(6,3),(6,4)\} \cup \Pi_{\mathcal{A}}$$
$$\widehat{\Pi}'(\Pi_{\mathcal{A}}) = \{(1,2),(2,1),(2,5),(4,6),(5,2),(6,4)\} \cup \Pi_{\mathcal{A}}$$

Note that for this example, the observer states are $X = \{\{0\},\{1,2,5\},\{3,4,6\}\}$. Then, the relation $\Pi^2_{\mathcal{A}}$ is obtained:

$$\Pi^2_{\mathcal{A}} = \{(1,2),(1,5),(2,1),(2,5),(3,4),(3,6),(4,3),(4,6),(5,1),(5,2),(6,3),(6,4)\}$$
$$\cup\{(q,q)|0 \leq q \leq 6\}$$

Thus, if string axc is observed, it is not possible to be sure which one among axc, auc or aucy really happened. □

The next example shows a case where the attack set is composed by two attackers.

Example 10. *Consider the automaton H shown in Fig. 4.9, where $\Sigma_v = \{u, x, y, z\}$ and $\mathcal{A} = \{A_{\{u,x\}}, A_{\{y,z\}}\}$. The red arrow represents a disablement. In order to find the relation of indistinguishable states $\Pi^2_{\mathcal{A}}$, Alg. 1 can be applied.*

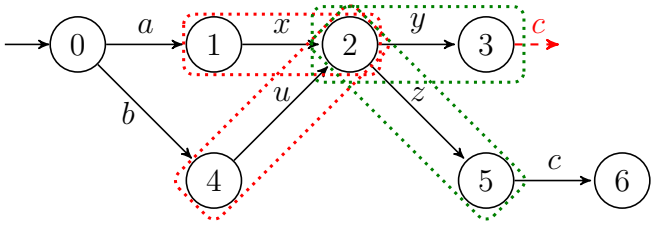


Figure 4.9: Automaton of Example 10.

After the execution of lines 1-3 of Alg. 1, the relations Π'_α obtained are:

$$\Pi'_{\{u,x\}} = \{(1,2),(4,2)\} \cup \{(q,q)|0 \leq q \leq 6\}$$
$$\Pi'_{\{y,z\}} = \{(2,3),(2,5),(3,5),(5,3)\} \cup \{(q,q)|0 \leq q \leq 6\}$$

The next step takes the union of all relations Π'_α, resulting in the relation $\Pi_{\mathcal{A}}$.

$$\Pi_{\mathcal{A}} = \{(1,2),(2,3),(2,5),(3,5),(4,2),(5,3)\} \cup \{(q,q)|0 \leq q \leq 6\}$$

In line 5 the relations $\Pi^\circ_{\mathcal{A}}$ and $\widehat{\Pi}'(\Pi_{\mathcal{A}})$ are obtained:

$$\Pi^\circ_{\mathcal{A}} = \{(1,3),(1,5),(2,1),(2,4),(3,1),(3,2),(3,4),(4,3),(4,5),(5,1),(5,2),(5,4)\} \cup \Pi_{\mathcal{A}}$$
$$\widehat{\Pi}'(\Pi_{\mathcal{A}}) = \Pi_{\mathcal{A}}$$

Note that for this example, the observer states are $X = \{\{0\},\{1,2,3,5\},\{2,3,4,5\},\{6\}\}$. The pairs $(1,4)$ and $(4,1)$ are not going to be added to the relation of indistinguishable states. The reason is that if string ax is observed, even though the attacker can erase event x and insert u, the resulting string au is not in $\mathcal{L}(H)$. This would reveal the presence of an attacker and hence is ruled out by the assumption that attackers are stealthy. On the other hand, the pairs $(2,3)$ and $(2,5)$ are going to be combined, resulting in $(5,3)$ and $(3,5)$ being added to the relation of indistinguishable states. The addition of the pairs $(3,5)$ and $(5,3)$ in $\Pi^2_{\mathcal{A}}$ reflects the fact that when bu is observed, it is not possible to know which one among the strings b, bu, buy and buz really happened. Thus,

$$\Pi^2_{\mathcal{A}} = \{(1,2),(1,3),(1,5),(2,1),(2,3),(2,4),(2,5),(3,2),(3,5),(4,2),(4,3),(4,5),(5,2),(5,3)\}$$
$$\cup\{(q,q)|0 \leq q \leq 6\}$$

□

The next example shows a case that has continuations that have the same projection when erasing vulnerable and unobservable events.

Example 11. *Consider the automaton H shown in Fig. 4.10, where $\Sigma_v = \{u, x\}$ and $\mathcal{A} = \{A_{\{u,x\}}\}$. The red arrow represents a disablement. In order to find the relation of indistinguishable states $\Pi^2_{\mathcal{A}}$, Alg. 1 can be applied.*

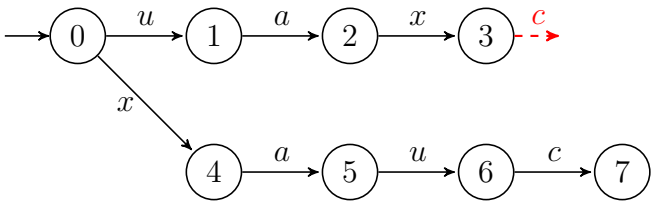


Figure 4.10: Automaton of Example 11.

After the execution of lines 1-3 of Alg. 1, the relations Π'_α obtained are:

$$\Pi'_{\{u,x\}} = \{(0,1),(0,4),(1,4),(2,3),(2,5),(2,6),(3,5),(3,6),(4,1),(5,2),(5,3),(5,6),(6,2),(6,3)\} \cup \{(q,q)|0 \leq q \leq 7\}$$

The next step takes the union of all relations Π'_α, resulting in the relation $\Pi_{\mathcal{A}}$. Since there is only one attacker, then

$$\Pi_{\mathcal{A}} = \Pi'_{\{u,x\}}$$

In line 5 the relations $\Pi^\circ_{\mathcal{A}}$ and $\widehat{\Pi}'(\Pi_{\mathcal{A}})$ are obtained:

$$\Pi^\circ_{\mathcal{A}} = \{(1,0),(3,2),(4,0),(6,5)\} \cup \Pi_{\mathcal{A}}$$
$$\widehat{\Pi}'(\Pi_{\mathcal{A}}) = \Pi_{\mathcal{A}}$$

Note that for this example, the observer states are $X = \{\{0,1,4\},\{2,3,5,6\},\{7\}\}$. Thus,

$$\Pi^2_{\mathcal{A}} = \{(0,1),(0,4),(1,0),(1,4),(2,3),(2,5),(2,6),(3,2),(3,5),(3,6),(4,0),(4,1),(5,2),(5,3),(5,6),(6,2),(6,3),(6,5)\} \cup \{(q,q)|0 \leq q \leq 7\}$$

Notice that upon observation of string uax it is not possible to know which string among ua, uax, xa and xau really happened. Also notice that this only happens because events u and x can be tampered by the same attacker.

□

Next, some useful lemmas are presented. They will be used later to prove the main result.

Lemma 1. *Let $G = (Q, \Sigma, \delta, q_0)$ be an automaton, $q, q' \in Q$ be states such that $(q, q') \in \Pi_\alpha$, obtained at line 2 of Alg. 1, for some attacker $A_\alpha \in \mathcal{A}$ and $P : \Sigma^* \to \Sigma_{uo}^*$ be a natural projection. Then, for all strings $w \in \mathcal{L}(G)$ and $v \in w(\alpha \cup \Sigma_{uo})^*$, such that $\delta(q_0, w) = q$ and $\delta(q_0, v) = q'$, it holds that*

1) $v \in P^{-1}(A_\alpha P(w))$;

2) $w \in P^{-1}(A_\alpha P(v))$.

⧫

Proof of Lemma 1. Since $v \in w(\alpha \cup \Sigma_{uo})^*$, to show that $v \in P^{-1}(A_\alpha P(w))$, it suffices to show that $w(\alpha \cup \Sigma_{uo})^* \subseteq P^{-1}(A_\alpha P(w))$. The string w can be written as $w = \sigma_1\sigma_2 \dots \sigma_n$, $\sigma_i \in \Sigma$, for $i = 1, 2, \dots, n$. Then $P(w) = P(\sigma_1)P(\sigma_2) \dots \ P(\sigma_n)$. To obtain $A_\alpha(P(\sigma_1) \dots P(\sigma_n))$, it is necessary to remove all events in α from $P(\sigma_1) \dots P(\sigma_n)$, resulting in $R_{\neg\alpha}(P(\sigma_1)) \dots R_{\neg\alpha}(P(\sigma_n))$. The next step is to find all strings that are equal to $R_{\neg\alpha}(P(\sigma_1)) \dots R_{\neg\alpha}(P(\sigma_n))$ if the events in α were also removed, i.e., $\alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^*$. Thus,

$$\begin{aligned} P^{-1}(A_\alpha P(w)) &= P^{-1}(A_\alpha(P(w))) \\ &= P^{-1}(A_\alpha(P(\sigma_1)P(\sigma_2) \dots P(\sigma_n))) \\ &= P^{-1}(\alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^*). \end{aligned} \tag{4.10}$$

In addition, $P(w)\alpha^* = P(\sigma_1)P(\sigma_2) \dots P(\sigma_n)\alpha^* \subseteq \alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^*$. Then it is possible to say that

$$\begin{aligned} P(w)\alpha^* \subseteq& \alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^* \\ P^{-1}(P(w)\alpha^*) \subseteq& P^{-1}(\alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^*). \end{aligned} \tag{4.11}$$

Taking the left side of (4.11),

$$P^{-1}(P(w)\alpha^*) = P^{-1}(P(w))P^{-1}(\alpha^*) \tag{4.12}$$

and as $w \in P^{-1}P(w)$, $(\alpha \cup \Sigma_{uo})^* \subseteq P^{-1}(\alpha^*)$, then

$$w(\alpha \cup \Sigma_{uo})^* \subseteq P^{-1}(P(w))P^{-1}(\alpha^*). \tag{4.13}$$

As $v \in w(\alpha \cup \Sigma_{uo})^*$, then it follows from (4.13) that $v \in P^{-1}(P(w))\ P^{-1}(\alpha^*)$. From (4.12), $v \in P^{-1}(P(w)\alpha^*)$ and, from (4.11), $v \in P^{-1}(\alpha^* R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^* R_{\neg\alpha}(P(\sigma_n))\alpha^*)$. Finally, (4.10) allows one to say that $v \in P^{-1}(A_\alpha P(w))$, proving 1).

For the second part, if $v \in w(\alpha \cup \Sigma_{uo})^*$ and $v \in \mathcal{L}(G)$, then $v = ws$, for some $s \in (\alpha \cup \Sigma_{uo})^*$ such that $ws \in \mathcal{L}(G)$. The strings w and s can be written as $w = \sigma_1\sigma_2 \dots \sigma_n$, with $\sigma_i \in \Sigma$, for $i = 1, 2, \dots n$ and $s = \gamma_1\gamma_2 \dots \gamma_m$, with $\gamma_j \in (\alpha \cup \Sigma_{uo})$, for $j = 1, 2, \dots m$. Thus, v can be written as $v = \sigma_1\sigma_2 \dots \sigma_n\gamma_1\gamma_2 \dots \gamma_m$. Applying the natural projection, one obtains

$$\begin{aligned} P(v) &= P(\sigma_1\sigma_2 \dots \sigma_n\gamma_1\gamma_2 \dots \gamma_m) \\ &= P(\sigma_1)P(\sigma_2) \dots P(\sigma_n)P(\gamma_1)P(\gamma_2) \dots P(\gamma_m). \end{aligned} \tag{4.14}$$

Now, the map A_α is applied over (4.14):

$$A_\alpha(P(v)) = A_\alpha(P(\sigma_1) \dots P(\sigma_n)P(\gamma_1) \dots P(\gamma_m)). \tag{4.15}$$

To obtain $A_\alpha(P(\sigma_1) \dots P(\sigma_n)P(\gamma_1) \dots P(\gamma_m))$, firstly all events in α are removed from $P(\sigma_1) \dots P(\sigma_n)P(\gamma_1) \dots P(\gamma_m)$, resulting in $R_{\neg\alpha}(P(\sigma_1)) \dots R_{\neg\alpha}(P(\sigma_n))R_{\neg\alpha}(P(\gamma_1)) \dots R_{\neg\alpha}(P(\gamma_m))$. For each $P(\gamma_j)$, there are two possibilities: *i)*$\gamma_j \in \Sigma_{uo}$ and $P(\gamma_j) = \varepsilon$ and; *ii)*$\gamma_j \in \alpha$ and $P(\gamma_j) = \gamma_j \in \alpha$. Either way, if the events in α are removed, it results in $R_{\neg\alpha}(P(\sigma_1)) \dots R_{\neg\alpha}(P(\sigma_n))$. The next step is to find the strings that are equal to $R_{\neg\alpha}(P(\sigma_1)) \dots R_{\neg\alpha}(P(\sigma_n))$ if the events in α were also removed, i.e., $\alpha^*R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^*R_{\neg\alpha}(P(\sigma_n))\alpha^*$. Thus, it is possible to write (4.15) as

$$A_\alpha(P(v)) = \alpha^*R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^*R_{\neg\alpha}(P(\sigma_n))\alpha^*. \tag{4.16}$$

As $P(w) = P(\sigma_1) \dots P(\sigma_n)$, then $P(w)$ is an element of $\alpha^*R_{\neg\alpha}(P(\sigma_1))\alpha^* \dots \alpha^*R_{\neg\alpha}(P(\sigma_n))\alpha^*$, which allows one to say that $P(w) \in A_\alpha(P(v)) = A_\alpha P(v)$. Applying the inverse projection on both sides, $P^{-1}(P(w)) \subseteq P^{-1}(A_\alpha P(v))$. Also, $w \in P^{-1}(P(w))$, allowing one to say that $w \in P^{-1}(A_\alpha P(v))$. This proves 2). □

Lemma 1 can be interpreted as follows. If $(q, q') \in \Pi_\alpha$, for some attacker $A_\alpha \in \mathcal{A}$, $q, q' \in Q$ and strings $v, w \in \mathcal{L}(G)$ such that $\delta(q_0, w) = q$ and $\delta(q_0, v) = q'$, then $v \in P^{-1}(A_\alpha P(w))$ means that v is obtained from w after the insertion of some events, while $w \in P^{-1}(A_\alpha P(v))$ means that w is obtained from v after the deletion of some events.

The next two lemmas are used in the main result's proof. The first one shows that for every pair $(q_1, q_2) \in \Pi_A^2$, then there exists a common observation $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$, with $\delta(q_0, w_1) = q_1$ and $\delta(q_0, w_2) = q_2$. The second one shows the opposite direction, that is, whenever it is possible to find a common observation $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$, this is captured by the inclusion of the pair (q_1, q_2) into the relation Π_A^2.

Lemma 2. *Let $G = (Q, \Sigma, \delta, q_0)$ be an automaton, $q_1, q_2 \in Q$ be states such that $(q_1, q_2) \in \Pi_{\mathcal{A}}^2$, obtained after the execution of Alg. 1, $w_1, w_2 \in \mathcal{L}(G)$ be strings such that $q_1 = \delta(q_0, w_1)$, $q_2 =$*

$\delta(q_0, w_2)$ and $P : \Sigma^* \to \Sigma_{uo}^*$ be a natural projection. Then, there exist attackers $A_{\alpha_i}, A_{\alpha_j} \in \mathcal{A}$ such that $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$. ♦

Proof of Lemma 2. There are multiple ways in which a pair (q_1, q_2) is formed during the execution of Alg. 1. These different ways will be shown in the following cases.

Case 1. For any pair of states (q_1, q_2) obtained at line 2 of Alg. 1, the pair can be obtained in one of the two following ways:

a) The pair was obtained by Def. 5. Then it is true that $(q_1, q_2) \in \Pi_{\alpha_i}$, for some attacker $A_{\alpha_i} \in \mathcal{A}$. Using Lemma 1, it is possible to write $w_1 \in P^{-1}(A_{\alpha_i}P(w_2))$. Because it is always true that $w_1 \in P^{-1}(A_{\alpha_i}P(w_1))$, it is possible to conclude that $P^{-1}(A_{\alpha_i}P(w_1)) \cap P^{-1}(A_{\alpha_i}P(w_2)) \cap \mathcal{L}(G) = P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$.

b) The pair was obtained by Def. 6. In this case, $\exists q \in Q$, such that $(q, q_1), (q, q_2) \in \Pi_{\alpha_i}$. According to Lemma 1, it is then possible to write $w_1 \in P^{-1}(A_{\alpha_i}P(w))$, with $\delta(q_0, w) = q$. The same reasoning allows one to write $w_2 \in P^{-1}(A_{\alpha_j}P(w))$. Again according to Lemma 1, it is also true that $w \in P^{-1}(A_{\alpha_i}P(w_1))$ and $w \in P^{-1}(A_{\alpha_i}P(w_2))$. Thus, it is possible to conclude that $P^{-1}(A_{\alpha_i}P(w_1)) \cap P^{-1}(A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) = P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$.

Case 2. The pair (q_1, q_2) is obtained by the relation $\Pi^{\rightarrow}(\widehat{\Pi}(\Pi_{\alpha_i}))$, at line 3 of Alg. 1. According to Def. 8, for every pair $(q_1, q_2) \in \Pi^{\rightarrow}(\widehat{\Pi}(\Pi_{\alpha_i}))$, the strings w_1 and w_2 can be written as $w_1 = w_1'u$ and $w_2 = w_2'v$, with $u, v \in (\Sigma_o \setminus \alpha_i)^*$. Consequently, the states q_1' and q_2', such that $\delta(q_0, w_1') = q_1'$ and $\delta(q_0, w_2') = q_2'$, form a pair $(q_1', q_2') \in \widehat{\Pi}(\Pi_{\alpha_i})$. There are two possibilities; there exists a state $q' \in Q$ such that: 1)$(q', q_1'), (q', q_2') \in \Pi_{\alpha_i}$ and $q' \neq q_1'$ and $q' \neq q_2'$; 2) $q' = q_1'$ or $q' = q_2'$ and either (q_1', q_1') and (q_1', q_2') or (q_2', q_2') and (q_2', q_1') are in Π_{α_i} Let w' be the string leading to state q', i.e., $\delta(q_0, w') = q'$. For the first possibility, since $(q', q_1'), (q', q_2') \in \Pi_{\alpha_i}$, then according to Def. 5, it must be the case that $w_1' \in w'(\alpha_i \cup \Sigma_{uo})^*$ and $w_2' \in w'(\alpha_i \cup \Sigma_{uo})^*$. Using Lemma 1, it is possible to say that $w' \in P^{-1}(A_{\alpha_i}P(w_1'))$ and $w' \in P^{-1}(A_{\alpha_i}P(w_2'))$. Still using Lemma 1, it is also possible to say that $w_1' \in P^{-1}(A_{\alpha_i}P(w'))$. In other words, it is possible to say that w_1' is an attacked version of w'. The same reasoning allows one to conclude that w' is also an attacked version of w_2'. Thus, it is possible to say that w_1' is an attacked version of w_2' and vice-versa. For the second possibility, as (q_1', q_2') or (q_2', q_1') belongs to Π_{α_i}, then according to Def. 5, it must be the case that $w_1' \in w_2'(\alpha_i \cup \Sigma_{uo})^*$ or $w_2' \in w_1'(\alpha_i \cup \Sigma_{uo})^*$. Either way, using Lemma 1, it is possible to say that $w_1' \in P^{-1}(A_{\alpha_i}P(w_1'))$ and $w_1' \in P^{-1}(A_{\alpha_i}P(w_2'))$. Thus, considering the two possibilities, it is possible to write

$$P(w_1') \in A_{\alpha_i}P(w_1') \cap A_{\alpha_i}P(w_2'), \tag{4.17}$$

Additionally, applying the attack map over the continuations u and v, one obtains:

$$A_{\alpha_i}P(u) = \alpha_i^* R_{\neg\alpha_i}(u)\alpha_i^*$$
$$A_{\alpha_i}P(v) = \alpha_i^* R_{\neg\alpha_i}(v)\alpha_i^*$$

Since $P_{\Sigma\to\Sigma_o\setminus\alpha}(u) = P_{\Sigma\to\Sigma_o\setminus\alpha}(v)$, then $R_{\neg\alpha_i}(u) = R_{\neg\alpha_i}(v)$, which means that $A_{\alpha_i}P(u) = A_{\alpha_i}P(v)$. Starting from (4.17) and since $A_{\alpha_i}P(w_1')u \subseteq A_{\alpha_i}P(w_1'u)$, then

$$\begin{aligned}
P(w_1') &\in A_{\alpha_i}P(w_1') \cap A_{\alpha_i}P(w_2') \\
P(w_1')u &\in A_{\alpha_i}P(w_1')u \cap A_{\alpha_i}P(w_2')u \\
P(w_1'u) &\in A_{\alpha_i}P(w_1'u) \cap A_{\alpha_i}P(w_2'u) \\
P(w_1'u) &\in A_{\alpha_i}P(w_1'u) \cap A_{\alpha_i}P(w_2'v) \\
P(w_1) &\in A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2) \\
w_1 &\in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2)).
\end{aligned} \tag{4.18}$$

As w_1 is a string such that $w_1 \in \mathcal{L}(G)$, then it is possible to say that $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$.

Case 3. At line 4, $\Pi_{\mathcal{A}}$ is redefined as the union of all relations Π_α obtained in lines 1-3 and no new pairs are created. Thus, it is still possible to say that $P^{-1}(A_{\alpha_i}P(w_1)) \cap P^{-1}(A_{\alpha_i}P(w_2)) \cap \mathcal{L}(G) = P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$ for a pair $(q_1, q_2) \in \Pi_{\mathcal{A}}$.

Case 4. Lastly, at line 5, the relations $\Pi^{\circ}_{\mathcal{A}}$ and $\widehat{\Pi}'(\Pi_{\mathcal{A}})$ are obtained and $\Pi^2_{\mathcal{A}}$ is defined as $\Pi^2_{\mathcal{A}} = \Pi^{\circ}_{\mathcal{A}} \cup \widehat{\Pi}'(\Pi_{\mathcal{A}})$. For each new pair (q_1, q_2) added by $\Pi^{\circ}_{\mathcal{A}}$, according to Def. 9, there exists q such that $(q_1, q), (q, q_2) \in \Pi_{\mathcal{A}}$. For every pair $(q_1, q) \in \Pi_{\mathcal{A}}$, it is true that (q_1, q) belongs to Π'_{α_i}, for some attacker $A_{\alpha_i} \in \mathcal{A}$. According to Case 2, for $\delta(q_0, w) = q$, $w \in P^{-1}(A_{\alpha_i}P(w) \cap A_{\alpha_i}P(w_1)) \cap \mathcal{L}(G)$. The same reasoning can be applied to the pair (q, q_2), resulting in $w \in P^{-1}(A_{\alpha_i}P(w) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$.

When the pair (q_1, q_2) is created by $\widehat{\Pi}'(\Pi_{\mathcal{A}})$, then there exists q such that $(q, q_1), (q, q_2) \in \Pi_{\mathcal{A}}$, $q \neq q_1$ and $q \neq q_2$. For every pair $(q, q_1) \in \Pi_{\mathcal{A}}$, it is true that (q, q_1) belongs to Π'_{α_i}, for some attacker $A_{\alpha_i} \in \mathcal{A}$. According to Case 2, for $\delta(q_0, w) = q$, $w \in P^{-1}(A_{\alpha_i}P(w) \cap A_{\alpha_i}P(w_1)) \cap \mathcal{L}(G)$. The same reasoning can be applied to the pair (q, q_2), resulting in $w \in P^{-1}(A_{\alpha_j}P(w) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$.

Thus, $w \in P^{-1}(A_{\alpha_i}P(w) \cap A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$ and $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$.

Since in all cases it is possible to say that $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$, this proves the lemma.

□

Lemma 2 can be interpreted as follows. If $(q_1, q_2) \in \Pi^2_{\mathcal{A}}$ obtained by the execution of Alg. 1, then it is possible to say that there exists a common string w that can be the result of the attack over strings w_1 and w_2 by two attackers. Note that the attackers can actually be the same. In other words, once w is observed, it is not possible to know which string among w, w_1 and w_2 really happened.

The next lemma shows that if two strings can generate a common observation because of the action of the attackers, then the states reached by these strings form a pair in the relation $\Pi^2_{\mathcal{A}}$.

Lemma 3. *Consider the strings $w_1, w_2 \in \mathcal{L}(G)$ such that $\delta(q_0, w_1) = q_1$ and $\delta(q_0, w_2) = q_2$. If $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) \neq \emptyset$, then $(q_1, q_2) \in \Pi^2_{\mathcal{A}}$, where $\Pi^2_{\mathcal{A}}$ is the relation obtained by the execution of Alg. 1.* ♦

Proof of Lemma 3. Let w be a string such that $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$ and $q \in Q$ be a state such that $\delta(q_0, w) = q$. Also, $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G) = P^{-1}(A_{\alpha_i}P(w_1)) \cap P^{-1}(A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$ and $w \in P^{-1}(A_{\alpha_i}P(w_1))$ and $w \in P^{-1}(A_{\alpha_j}P(w_2))$. Since w is an attacked version of w_1, it is also true that w_1 is an attacked version of w, i.e., $w_1 \in P^{-1}(A_{\alpha_i}P(w))$. The same is true regarding w and w_2, i.e., $w_2 \in P^{-1}(A_{\alpha_j}P(w))$.

Since w and w_1 can both be written as $w = w'u$ and $w_1 = w'_1v$, respectively, with $u, v \in (\Sigma_o \setminus \alpha_i)^*$ and $P_{\Sigma \to \Sigma_o \setminus \alpha}(u) = P_{\Sigma \to \Sigma_o \setminus \alpha}(v)$ being the longest common indistinguishable suffix (and note that the longest common indistinguishable suffix u and v may be equal to ε if the strings do not share a common suffix), then, it is now possible to write $w' \in P^{-1}(A_{\alpha_i}P(w'_1))$ and $w'_1 \in P^{-1}(A_{\alpha_i}P(w'))$. Consider a string s such that $w' \in s(\alpha_i \cup \Sigma_{uo})^*$ and $w'_1 \in s(\alpha_i \cup \Sigma_{uo})^*$. Then, it is possible to say that $w' \in s(\alpha_i \cup \Sigma_{uo})^* \subseteq P^{-1}(A_{\alpha_i}P(s))$ or $w'_1 \in s(\alpha_i \cup \Sigma_{uo})^* \subseteq P^{-1}(A_{\alpha_i}P(s))$. Using Def. 5, it is possible to write that $(q_s, q') \in \Pi_{\alpha_i}$ and $(q_s, q'_1) \in \Pi_{\alpha_i}$, where $\delta(q_0, w') = q'$, $\delta(q_0, w'_1) = q'_1$ and $\delta(q_0, s) = q_s$. The next step in Alg. 1 is to obtain the relation $\widehat{\Pi}(\Pi_{\alpha_i})$. According to Def. 6, the pairs (q', q'_1) and (q'_1, q') will be added to $\widehat{\Pi}(\Pi_{\alpha_i})$ if $q_s \neq q'$ and $q_s \neq q'_1$. Otherwise the pairs (q_s, q') and (q_s, q'_1) are already in Π_{α_i} (and consequently, in $\widehat{\Pi}_{\mathcal{A}}(\Pi_{\alpha_i})$). Note that if $q_s \in Q_{DS}$, then q' and q'_1 are also in Q_{DS}, since they are connected by vulnerable and unobservable events. At line 3, the relation $\Pi'_{\alpha_i} = \Pi^{\to}(\widehat{\Pi}(\Pi_{\alpha_i}))$ is obtained. If $q_s \neq q'$ and $q_s \neq q'_1$, then the pairs (q, q_1) and (q_1, q) are added to the relation Π'_{α_i}. Otherwise, the pairs (q, q) and (q, q_1) are added to the relation Π'_{α_i} (the pair (q, q) is already in $\widehat{\Pi}(\Pi_{\alpha_i})$).

Applying the same reasoning of the last paragraph to strings w and w_2, it is possible to conclude that either the pairs (q, q_2) and (q_2, q) or (q, q) and (q, q_2) belong to Π'_{α_j}, obtained at line 3 of Alg. 1.

In line 4 of Alg. 1, all the relations Π'_α are merged into the relation $\Pi_\mathcal{A}$. At line 5, the relations $\Pi^\circ_\mathcal{A}$ and $\widehat{\Pi}'(\Pi_\mathcal{A})$ are obtained. Next, a contradiction is employed to show that there is a state $x \in X$ in the observer of G such that $q_1, q_2 \in x$.

Assume that $q_1 \in x_1$, $q_2 \in x_2$ and $x_1 \neq x_2$. Since (q, q_1) and (q, q_2) are in $\Pi_\mathcal{A}$, it means that $q \in x_1$ and $q \in x_2$. To be possible for q_1 and q_2 to be at different states in the observer, it means that there is a non-vulnerable event that distinguish them. However, if that is the case, then either (q, q_1) or (q, q_2) will not be in $\Pi_\mathcal{A}$, which is a contradiction. Thus, it is possible to conclude that $x_1 = x_2$.

If $q \neq q_1$ and $q \neq q_2$, then the pairs (q_1, q_2) and (q_2, q_1) will be added to $\Pi^2_\mathcal{A}$ by $\widehat{\Pi}'(\Pi_\mathcal{A})$. However, if $q = q_1$ or $q = q_2$, then the pairs (q_1, q_2) and (q_2, q_1) will be added to $\Pi^2_\mathcal{A}$ by $\Pi^\circ_\mathcal{A}$. Thus, it is possible to conclude $(q_1, q_2) \in \Pi^2_\mathcal{A}$.

□

Lemma 3 can be interpreted as follows. If two attackers can act over strings w_1 and w_2 producing a common observation w, then this fact is captured by the inclusion of the pair (q_1, q_2) in the relation $\Pi^2_\mathcal{A}$.

The requirement of P-observability for an attack set is only relevant when dealing with strings $w, w' \in K$ that can generate the same observation after the attack and such that there exists an event $\sigma \in \Sigma$ such that $w\sigma \in K$ and $w'\sigma \in \mathcal{L}(G) \setminus K$ or $w'\sigma \in K$ and $w\sigma \in \mathcal{L}(G) \setminus K$. When $w\sigma \in \mathcal{L}(G) \setminus K$ or $w'\sigma \in \mathcal{L}(G) \setminus K$, it means that event σ should be disabled after w or w', respectively. In the same way, when $w\sigma \in K$ or $w'\sigma \in K$, it means that event σ should be enabled after w or w', respectively. Let $\xi : Q \to 2^\Sigma$ be a map defined as

$$\xi(q) := \{\sigma \in \Sigma | (\exists w \in \mathcal{L}(G))[\delta(q_0, w) = q \wedge w\sigma \in K]\} \tag{4.19}$$

that gives, for a state $q \in Q$, the set of *enabled* events at state q. Also, let the map $\phi : Q \to 2^\Sigma$ defined as

$$\phi(q) := \{\sigma \in \Sigma | (\exists w \in \mathcal{L}(G))[\delta(q_0, w) = q \wedge w\sigma \in \mathcal{L}(G) \setminus K]\} \tag{4.20}$$

be the map that gives, for a given state $q \in Q$, the set of *disabled* events at state q. Inspired by (Su and Wonham, 2004), where the authors present a binary relation of pairs of states that are consistent with respect to their control action and to their marking, the following definition is introduced.

Definition 10 (Relation of control inconsistent states)**.** *The binary relation $\mathcal{I} \subseteq Q \times Q$ defined by*

$$\mathcal{I} := \{(q_1, q_2) \in Q \times Q | \xi(q_1) \cap \phi(q_2) \neq \emptyset \vee \xi(q_2) \cap \phi(q_1) \neq \emptyset\} \tag{4.21}$$

is the relation of control inconsistent states. ◊

In words, according to Definition 10, a pair of states is in the relation of control inconsistent states $\mathcal{I}$ if the control action at these two states are conflicting. It is important to notice that the relation $\mathcal{I}$ is not transitive but is symmetric. For testing P-observability for an attack set, it is not necessary to consider all pairs of strings $w, w' \in K$, but only the pairs such that their control action is conflicting. In other words, only the states that are control inconsistent need to be considered. Next, Proposition 1 shows the relationship between the relations $\mathcal{I}$ and $\text{act}_{K \subseteq L}$.

Proposition 1. *Let $G = (Q, \Sigma, \delta, q_0)$ be an automaton, K be a desired language and $\mathcal{I}$ be the relation of control inconsistent states obtained according to Def. 10. Then, a pair (q_1, q_2) is in $\mathcal{I}$ only and only if $(w_1, w_2) \notin act_{K \subseteq L}$, where w_1 and w_2 are strings such that $\delta(q_0, w_1) = q_1$ and $\delta(q_0, w_2) = q_2$.* ⧫

Proof of Proposition 1. ($\Rightarrow$) The proof will be conducted by contradiction. Suppose $(q_1, q_2) \in \mathcal{I}$ and $(w_1, w_2) \in \text{act}_{K \subseteq L}$. According to Def. 10, if $(q_1, q_2) \in \mathcal{I}$, then $\exists \sigma \in \Sigma$ such that $\sigma \in \xi(q_1) \cap \phi(q_2)$ or $\sigma \in \xi(q_2) \cap \phi(q_1)$. According to (4.19), (4.20) and (3.5), σ is an event such that $(w_1, w_2) \notin \text{act}_{K \subseteq L}$, which is contradiction.

($\Leftarrow$) Suppose $(w_1, w_2) \in \text{act}_{K \subseteq L}$ and $(q_1, q_2) \in \mathcal{I}$. If $(w_1, w_2) \in \text{act}_{K \subseteq L}$, then $\nexists \sigma \in \Sigma$ such that $w_1\sigma \in K$ and $w_2\sigma \in \mathcal{L}(G) \setminus K$ or $w_2\sigma \in K$ and $w_1\sigma \in \mathcal{L}(G) \setminus K$. In this case, $\xi(q_1) \cap \phi(q_2) = \emptyset$ and $\xi(q_2) \cap \phi(q_1) = \emptyset$. Thus, $(q_1, q_2) \notin \mathcal{I}$, resulting in a contradiction. □

In other words, since the relations $\mathcal{I}$ and $\text{act}_{K \subseteq L}$ have opposite meanings, when a pair $(q_1, q_2) \in \mathcal{I}$, then their corresponding strings are not in $\text{act}_{K \subseteq L}$ or vice-versa.

Next, the main result of this chapter is presented, a new test for P-observability for an attack set.

Theorem 3. *Let G be a deterministic finite automaton that represents the behavior of a system, K the desired language, with $K \subseteq \mathcal{L}(G)$. The set $\Sigma_{uo} \subseteq \Sigma$ is the set of unobservable events, $\Sigma_v \subseteq \Sigma$ is the set of vulnerable events ($\Sigma_{uo} \cap \Sigma_v = \emptyset$) and $\mathcal{A} = \{A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_M}\}$ is the attack set, with $\alpha_i \subseteq \Sigma_v$, $i = 1, \ldots, M$. Let $\Pi^2_{\mathcal{A}}$ be the relation of indistinguishable states obtained after the execution of Alg. 1 and let $\mathcal{I}$ be the relation of control inconsistent states of the automaton that implements K. The language K will be P-observable for $\mathcal{L}(G)$ and $\mathcal{A}$ if and only if $\Pi^2_{\mathcal{A}} \cap \mathcal{I} = \emptyset$.* ⧫

Theorem 3 states that a given desired language K is P-observable for an attack set $\mathcal{A}$ via a stealthy attack if and only if there are no pairs of states common to the relations $\Pi^2_{\mathcal{A}}$ and $\mathcal{I}$. One can apply Theorem 3 to verify if a language K is P-observable for an attack set $\mathcal{A}$ of stealthy attacker by checking if $(q, q') \notin \Pi^2_{\mathcal{A}}$ holds for every pair (q, q') in the relation of control inconsistent states $\mathcal{I}$.

Proof of Theorem 3. ($\Rightarrow$) The proof is conducted by contradiction. Suppose K is P-observable and $\Pi^2_{\mathcal{A}} \cap \mathcal{I} \neq \emptyset$. Let $q_1, q_2 \in Q$ be states such that $(q_1, q_2) \in \Pi^2_{\mathcal{A}} \cap \mathcal{I}$ and $w_1, w_2 \in \mathcal{L}(G)$ be strings such that $\delta(q_0, w_1) = q_1$ and $\delta(q_0, w_2) = q_2$. If $(q_1, q_2) \in \mathcal{I}$, then $\exists \sigma \in \Sigma$ such that $w_1\sigma \in K$ and $w_2\sigma \in \mathcal{L}(G) \setminus K$ or $w_1\sigma \in \mathcal{L}(G) \setminus K$ and $w_2\sigma \in K$. This means that $(w_1, w_2) \notin \text{act}_{K \subset \mathcal{L}(G)}$.

If $(q_1, q_2) \in \Pi^2_{\mathcal{A}}$, then according to Lemma 2, $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_i}P(w_2)) \neq \emptyset$, which in turn means that $(w_1, w_2) \in R_{A_{\alpha_i}, A_{\alpha_j}}$. Hence, $(w_1, w_2) \in R_{A_{\alpha_i}, A_{\alpha_j}}$ but $(w_1, w_2) \notin \text{act}_{K \subset L}$, which contradicts the definition of P-observability for an attack set.

($\Leftarrow$) Again, by contradiction, suppose that K is not P-observable and $\Pi^2_{\mathcal{A}} \cap \mathcal{I} = \emptyset$. If K is not P-observable, it means that $\exists w_1, w_2 \in \Sigma^*$ and $\exists A_{\alpha_i}, A_{\alpha_j} \in \mathcal{A}$ such that $A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2) \neq \emptyset$ ($(w_1, w_2) \in R_{A_{\alpha_i}, A_{\alpha_j}}$) and also $\exists \sigma \in \Sigma$ such that $w_1\sigma \in K$ and $w_2\sigma \in \mathcal{L}(G) \setminus K$ or $w_1\sigma \in \mathcal{L}(G) \setminus K$ and $w_2\sigma \in K$ ($(w_1, w_2) \notin \text{act}_{K \subset \mathcal{L}(G)}$).

Let w_1 and w_2 be strings such that $\delta(q_0, w_1) = q_1$, $\delta(q_0, w_2) = q_2$. By Proposition 1, $(q_1, q_2) \in \mathcal{I}$. Furthermore, as $(w_1, w_2) \in R_{A_{\alpha_i}, A_{\alpha_j}}$, then $A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2) \neq \emptyset$ and $P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \neq \emptyset$, for some $A_{\alpha_i}, A_{\alpha_j} \in \mathcal{A}$. Let w be a string such that $w \in P^{-1}(A_{\alpha_i}P(w_1) \cap A_{\alpha_j}P(w_2)) \cap \mathcal{L}(G)$ and $q \in Q$ be a state such that $\delta(q_0, w) = q$. Then, according to Lemma 3, it is possible to say that $(q_1, q_2) \in \Pi^2_{\mathcal{A}}$ and $\Pi^2_{\mathcal{A}} \cap \mathcal{I} \neq \emptyset$, which is a contradiction.

All the presented cases allow one to conclude that $\Pi^2_{\mathcal{A}} \cap \mathcal{I} = \emptyset$, contradicting the initial hypothesis. Therefore, it is possible to say that K is P-observable if and only if $\Pi^2_{\mathcal{A}} \cap \mathcal{I} = \emptyset$. □

The next example uses Theorem 3 to test if a given language is P-observable for an attack set.

Example 12. *Consider Examples 5 to 11. For each example, the relation of control inconsistent states $\mathcal{I}_n$, with n being the example number, is given below.*

$$\begin{aligned}
\mathcal{I}_5 =&\{(1,4),(1,7),(4,1),(7,1)\}\\
\mathcal{I}_6 =&\{(2,5),(5,2)\}\\
\mathcal{I}_7 =&\{(1,5),(5,1)\}\\
\mathcal{I}_8 =&\{(6,7),(7,6)\}\\
\mathcal{I}_9 =&\{(4,6),(6,4)\}\\
\mathcal{I}_{10} =&\{(3,5),(5,3)\}\\
\mathcal{I}_{11} =&\{(3,6),(6,3)\}
\end{aligned}$$

The relations $\Pi^2_{\mathcal{A}}$ obtained for each example, are summarized next.

$$\begin{aligned}
\textit{Example 5}: \Pi^2_{\mathcal{A}} =&\{(1,2),(1,3),(1,4),(1,5),(1,7),(2,1),(2,3),(2,4),(2,5),(2,7),(3,1),(3,2),\\
&(3,4),(3,5),(3,7),(4,1),(4,2),(4,3),(4,5),(4,7),(5,3),(5,4),(5,1),(5,2),\\
&(5,7),(7,1),(7,2),(7,3),(7,4),(7,5)\}\cup\{(q,q)|0\leq q\leq 8\}.\\
\textit{Example 6}: \Pi^2_{\mathcal{A}} =&\{(0,4),(1,2),(1,5),(2,1),(2,5),(4,0),(5,1),(5,2)\}\cup\{(q,q)|0\leq q\leq 5\}.\\
\textit{Example 7}: \Pi^2_{\mathcal{A}} =&\{(0,3),(0,4),(3,0),(3,4),(4,3),(4,0)\}\cup\{(q,q)|0\leq q\leq 5\}.\\
\textit{Example 8}: \Pi^2_{\mathcal{A}} =&\{(1,2),(2,1),(3,4),(4,3),(5,6),(5,7),(6,5),(7,5)\}\cup\{(q,q)|0\leq q\leq 8\}.\\
\textit{Example 9}: \Pi^2_{\mathcal{A}} =&\{(1,2),(1,5),(2,1),(2,5),(3,4),(3,6),(4,3),(4,6),(5,1),(5,2),(6,3),(6,4)\}\cup\\
&\{(q,q)|0\leq q\leq 6\}.\\
\textit{Example 10}: \Pi^2_{\mathcal{A}} =&\{(1,2),(1,5),(2,1),(2,3),(2,4),(2,5),(3,2),(3,4),(3,5),(4,2),(4,3),(4,5),\\
&(5,1),(5,2),(5,3),(5,4)\}\cup\{(q,q)|0\leq q\leq 6\}.\\
\textit{Example 11}: \Pi^2_{\mathcal{A}} =&\{(1,2),(1,3),(1,5),(2,1),(2,3),(2,4),(2,5),(3,2),(3,5),(4,2),(4,3),(4,5),\\
&(5,2),(5,3)\}\cup\{(q,q)|0\leq q\leq 7\}
\end{aligned}$$

Then, according to Theorem 3, it is possible to have the following conclusions:

Example 5 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_5 \neq \emptyset$: *The language K is not P-observable for the attack set $\mathcal{A}$.*

Example 6 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_6 \neq \emptyset$: *The language K is not P-observable for the attack set $\mathcal{A}$.*

Example 7 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_7 = \emptyset$: *The language K is P-observable for the attack set $\mathcal{A}$.*

Example 8 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_8 = \emptyset$: *The language K is P-observable for the attack set $\mathcal{A}$.*

Example 9 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_9 \neq \emptyset$: *The language K is not P-observable for the attack set $\mathcal{A}$.*

Example 10 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_{10} \neq \emptyset$: *The language K is not P-observable for the attack set $\mathcal{A}$.*

Example 11 – $\Pi^2_{\mathcal{A}} \cap \mathcal{I}_{11} \neq \emptyset$: *The language K is not P-observable for the attack set $\mathcal{A}$.*

□

In the next sections the algorithms developed to perform the P-observability test are presented, as well as their time complexity analysis.

4.4 Algorithms

The first algorithm is Algorithm 2, that obtains the relation of indistinguishable states Π_α with respect to attacker A_α. The algorithm receives an automaton $H = (Q^H, \Sigma, \delta^H, q_0^H)$, such that $\mathcal{L}(H) = K$, and the set α of events that attacker $A_\alpha \in \mathcal{A}$ can manipulate.

In Alg. 2, Ξ represents a FIFO queue and the call to the function at line 5 sets the attribute of the states, which is the number of outgoing transitions from each state. This attribute is represented by $q.out$, for $q \in Q^H$.

The operation at line 1 applies the map defined in (4.1) over all transitions in δ, as follows:

$$R_{\neg(\Sigma\setminus\alpha)}(\delta^H) := \{(q, \sigma, q') \in \Delta^H | \sigma \in \alpha\}. \tag{4.22}$$

In words, the operation $R_{\neg(\Sigma\setminus\alpha)}(\delta)$ removes from δ all the transitions labeled with non-vulnerable events, i.e., events in $\Sigma \setminus \alpha$, resulting in δ_α. The idea behind Alg. 2 is to first obtain a modified automaton $H' = (Q, \Sigma, \delta_\alpha, q_0)$ and for each state $q \in Q$, a breadth-first search (BFS) is performed. Once state $q' \in Q$ is reached from q, the pair (q, q') is added to Π_α. The set π keeps track of all states already visited in a particular BFS, guaranteeing that each state is visited only once.

The next algorithm finds the pairs of states that can be reached through identical continuations from states that are already indistinguishable.

Algorithm 2: Indistinguishable states Π_α with respect to attacker A_α

Input: $H = (Q^H, \Sigma, \delta^H, q_0^H), \alpha$
Result: Π_α
$\delta_\alpha \leftarrow R_{\neg(\Sigma\setminus\alpha)}(\delta^H)$
$\Xi \leftarrow \emptyset$ // Queue
$\pi \leftarrow \emptyset$
$\Pi \leftarrow \emptyset$
Initialize_State_Attributes()
foreach $q \in Q^H$ **do**
  Ξ.Enqueue(q)
  while $\Xi \neq \emptyset$ **do**
    $q_c \leftarrow \Xi$.Dequeue()
    if $(q, q_c) \notin \Pi$ **then**
      $\Pi \leftarrow \Pi \cup \{(q, q_c)\}$
    if $q_c \notin \pi$ **then**
      $\pi \leftarrow \pi \cup \{q_c\}$
    if $q_c.out > 0$ **then**
      foreach $(q_c, \sigma, q') \in \Delta_\alpha$ **do**
        if $q' \notin \pi$ **then**
          Ξ.Enqueue(q')
  $\pi \leftarrow \emptyset$
return Π_α

Algorithm 3: Identical continuations

Input: $H = (Q^H, \Sigma, \delta^H, q_0^H), \Pi$
Result: $\Pi^{\rightarrow}(\Pi)$
$\Pi^{\rightarrow}(\Pi) \leftarrow \Pi$
$\Sigma_M \leftarrow \{\sigma \in \Sigma | (\exists (q, \sigma, q_1), (q', \sigma, q_2) \in \Delta)(\sigma \in (\Sigma_o \setminus \Sigma_v))[q \neq q' \wedge q_1 \neq q_2]\}$
$\pi \leftarrow \emptyset$
while $|\pi| \neq |\Pi^{\rightarrow}(\Pi)|$ **do**
  $\Pi^{\rightarrow}(\Pi) \leftarrow \Pi^{\rightarrow}(\Pi) \cup \pi$
  $\pi \leftarrow \Pi^{\rightarrow}(\Pi)$
  foreach $\sigma \in \Sigma_M$ **do**
    foreach $(q, q') \in \Pi^{\rightarrow}(\Pi)$ **do**
      if $\delta(q, \sigma)! \wedge \delta(q', \sigma)!$ **then**
        $\pi \leftarrow \pi \cup \{(\delta(q, \sigma), \delta(q', \sigma))\}$
return $\Pi^{\rightarrow}(\Pi)$

Algorithm 3 receives as input the automaton H and the relation of indistinguishable states. Then, line 1 calculates the set Σ_M, which has all events such that they appear in more than one transition in H and those transitions have different origins and destinations. The next part of the algorithm is a `while` loop, which is executed as long as new pairs of indistinguishable states can be created. At each iteration of the `while` loop, for each event in $\sigma \in \Sigma_M$ and for each pair in the relation $(q, q') \in \Pi_\alpha$, the algorithm checks if $\delta(q, \sigma)$ and $\delta(q', \sigma)$ are both defined, and if so, the pair $(\delta(q, \sigma), \delta(q', \sigma))$ is added to π.

Algorithm 4 is used to check if a given desired language is P-observable for an attack set. It receives an automaton H such that $\mathcal{L}(H) = K$, the relation of pairs of control inconsistent states $\mathcal{I}$ and the attack set $\mathcal{A}$. It returns TRUE if K is P-observable or FALSE, otherwise. Initially, the relations of indistinguishable states Π_α, for all $A_\alpha \in \mathcal{A}$ are obtained and added to $\Pi_\mathcal{A}$. Secondly, the relation of indistinguishable states after pairwise combined attacks $\Pi^2_\mathcal{A}$ is determined. Finally, for each pair $(q_1, q_2) \in \mathcal{I}$, the inclusion of (q_1, q_2) in Π^2_α is checked. If at some point the inclusion holds, then the algorithm finishes by returning FALSE, since the language K is not P-observable. Otherwise, the algorithm returns TRUE.

Algorithm 4: P-observability test

```
   Input: H = (Q^H, Σ, δ^H, q_0^H), I, A
   Result: {TRUE, FALSE}
1  Π²_A ← INDISTINGUISHABLE_STATES(H, A)
2  foreach (q1, q2) ∈ I do
3  |  if (q1, q2) ∈ Π²_A then
4  |  |  return FALSE
5  return TRUE
```

In Alg. 4 it is assumed that the relation of control inconsistent states is already provided. The relation of indistinguishable states $\Pi^2_\mathcal{A}$ is obtained by the execution of Alg. 1. Algorithm 5 shows how $\mathcal{I}$ can be obtained. The algorithm receives two automata $G = (Q^G, \Sigma, \delta^G, q_0^G)$ and $H = (Q^H, \Sigma, \delta^H, q_0^H)$, representing the plant and the desired language, respectively. It is assumed that $Q^H \subseteq Q^G$. To obtain $\xi(q)$, for $q \in Q^H$, all events $\sigma \in \Sigma$ such that $(q, \sigma, q') \in \Delta^H$, for some $q' \in Q^H$, are taken. On the other hand, $\phi(q)$, for $q \in Q^H$, is obtained by picking all $\sigma \in \Sigma$ such that $(q, \sigma, q') \in \Delta^G$ and $(q, \sigma, q') \notin \Delta^H$, for some $q' \in Q^G$. Then, for all pairs of states $q, q' \in Q^H$, it is verified if $\phi(q) \cap \xi(q') \neq \emptyset$ or $\phi(q') \cap \xi(q) \neq \emptyset$.

Next, the complexity of the algorithms is discussed.

Algorithm 5: Inconsistent states

Input: $G = (Q^G, \Sigma, \delta^G, q_0^G)$, $H = (Q^H, \Sigma, \delta^H, q_0^H)$
Result: $\mathcal{I}$
1 $\mathcal{I} \leftarrow \emptyset$
2 **foreach** $q \in Q^H$ **do**
3 | **foreach** $q' \in Q^H \setminus \{q\}$ **do**
4 | | **if** $\phi(q) \cap \xi(q') \neq \emptyset \vee \phi(q') \cap \xi(q) \neq \emptyset$ **then**
5 | | | $\mathcal{I} \leftarrow \mathcal{I} \cup \{(q, q')\}$
6 **return** $\mathcal{I}$

4.5 Time complexity

In this subsection the time complexity of the presented algorithms is discussed. The sets Q^H and Q^G are referred to as Q and the transition functions δ^H and δ^G as δ (the corresponding transitions set as Δ). At line 1 of Alg. 2, in order to select only the transitions labeled with events in α, all transitions have to be checked, resulting in the time complexity $O(|\Delta|)$. Line 5 also has complexity $O(|\Delta|)$, since in order to find the number of outgoing transitions from a given state, all transitions have to be considered. The `foreach` loop of lines 6-25 is executed once for each state $q \in Q$. At each iteration, a BFS is performed, considering state q as the source state. Then, all transitions in Δ_α are checked once, resulting in a complexity of $O(|Q||\Delta|)$. It is important to notice that a given state is never visited twice during the BFS. Thus, the overall complexity of Alg. 2 is $O(|Q||\Delta|)$.

Algorithm 3 starts by obtaining the set Σ_M. In order to build this set, all transitions have to be compared to each other, which leads to time complexity of $O(|\Delta|^2)$. The maximum number of executions of the `while` loop of lines 4-10 is $O(|Q|^2)$, while the `foreach` loops of lines 7-10 and 8-10 execute at most $|\Sigma|$ and $|Q|^2$ times, respectively. Thus, the overall time complexity of Alg. 3 is $O(|\Delta|^2 + |Q|^2|\Sigma||Q|^2) = O(|\Sigma||Q|^4)$.

Now it is possible to find the time complexity of Alg 1. The `foreach` loop of lines 1-3 is executed once for each attacker in $\mathcal{A}$. The complexity of finding the relation of indistinguishable states Π_α is $O(|Q||\Delta|)$. To implement the operation $\widehat{\Pi}(\Pi_\alpha)$, each relation Π_α can be represented as a $|Q| \times |Q|$ matrix. Then, all elements of the matrix representing Π_α need to be visited once, in the worst case. Thus, obtaining $\widehat{\Pi}(\Pi_\alpha)$ has a worst case complexity of $O(|Q|^2)$. Then, to find their identical continuations, the time complexity is $O(|\Sigma||Q|^4)$. Consequently, the complexity of this `foreach` loop is $O(|\mathcal{A}|(|Q||\Delta| + |Q|^2 + |\Sigma||Q|^4))$. The union operation at line 4, can be implemented as an adapted sum of matrices. This means that all elements of the matrices need to be visited once, resulting in a complexity of $O(|Q|^2)$ for each union operation. In order to find the set Q_{MP} at line 5, the observer automaton need to be obtained, which has exponential complexity on the number of the states.

In line 6, the composition of the relations can be done with complexity of $O(|Q|^3)$, as a matrix multiplication. Hence, the algorithm has a overall time complexity of $O(|\mathcal{A}|(|Q||\Delta| + |Q|^2 + |\Sigma||Q|^4 + |Q|^2) + |Q|^2 + 2^{|Q|}) = O(2^{|Q|})$.

The P-observability for an attack set test, performed by Alg. 4, has as the worst case scenario, all states being control inconsistent with all other states. This means that $\mathcal{I}$ can have up to $|Q|^2$ pairs and, consequently, the `foreach` loop of lines 2-4 will be executed at most $|Q|^2$ times. At line 3, the verification of the inclusion of (q_1, q_2) in $\Pi^2_{\mathcal{A}}$ can be done in constant time. This means that the `foreach` loop of lines 2-4 has a worst case complexity of $O(|Q|^2)$. Finally, the overall complexity of Alg. 4 is $O(|\mathcal{A}|(|\Sigma||Q|^4)+|Q|^2) = O(|\mathcal{A}|(|\Sigma||Q|^4))$.

For Alg. 5, the `foreach` loop of lines 2-8 executes $|Q|$ times while the `foreach` loop of lines 3-7 executes $|Q| - 1$ times. To find the set of enabled events all transitions in Δ^H have to be analyzed, while to find the set of disabled events, for each transition in Δ, it is verified if the same transition is not in Δ^H. The intersection operation has a cost of $O(|\Sigma|)$, since the set of enabled and disabled events have at most $|\Sigma|$ elements. Thus, the overall complexity of this algorithm is $O(|Q|^2(|\Delta|+|\Sigma|+|\Delta|^2)) = O(|Q|^2(|\Sigma|+|\Delta|^2))$.

In the next section, the P-observability test is applied to a problem extracted from the literature.

4.6 Case study

Computer networks have many vulnerabilities in hardware or software and, although taking advantage of a single vulnerability normally doesn't cause significant damage, a malicious user can exploit a sequence of vulnerabilities before achieving a particular goal, e.g., getting root privilege into a server. Such attacks are called multi-step attacks Akinyemi et al. (2018).

Multi-step attacks can be described as *attack graphs*, that show the possible sequences of malicious actions that an attacker can follow to invade the network and gain certain privileges. Such graphs can be obtained by employing vulnerability scanner software Fadlallah et al. (2016). Figure 4.11 is an example of an attack graph, taken from Matthews et al. (2020). It has three node types. Diamonds represent a state that an attacker can be in, such as a certain level of privilege with respect to a host. Ellipses represent actions such as exploiting a vulnerability or connecting to a host, while rectangles represent conditions that have to be true so that the action can be executed. Node 0 represents the initial state, where an attacker located outside the network hasn't taken advantage of any vulnerability yet, while node 1

represents that the attacker has gained root privileges in a database server. For a detailed description of each node's meaning, the reader is referred to Matthews et al. (2020).

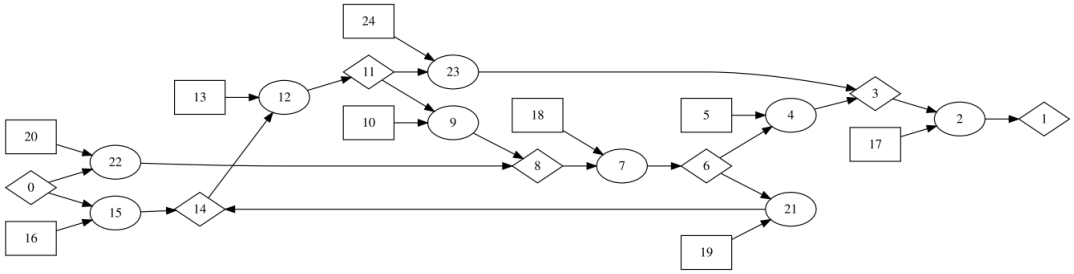


Figure 4.11: Attack graph. From Matthews et al. (2020).

The attack graph of Fig. 4.11 was converted into the automaton shown in Fig. 4.12. Each diamond node in the attack graph has generated a state in the automaton and each ellipsis was converted to a transition in the automaton. The rectangle nodes were not included in the automaton and it is assumed that the conditions needed for an action to happen are always met. The numbers from Fig. 4.11 were kept in order to provide easier correspondence between both figures. All events in Fig. 4.12 are uncontrollable.

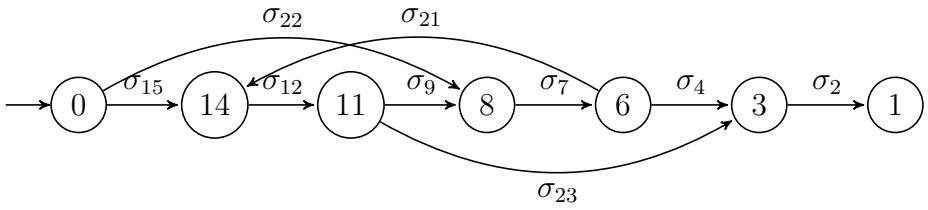


Figure 4.12: Automaton representing the attack graph of Fig. 4.11.

In order to prevent an attacker from accessing the database, a security module that monitors the network can be installed. This security module can control the connection to the database server, by granting or denying an user's access to it. These actions are represented by controllable events μ and λ, respectively, as shown by the automaton G of Fig. 4.13. The security module is equipped with security logs that allow the events representing the use of the vulnerabilities to be observed.

While the system is at state 0, which represents a secure state, denial of access must be avoided, since it can forbid legal users from accessing the database. In contrast, if the system is at state 1, at which a malicious user has root privileges in the database server, the access to the database must be forbidden. These situations are represented by states A and B, respectively. The control of the security module can be implemented by means of a supervisor. The desired language K is generated by the automaton shown in Fig. 4.14, where only the

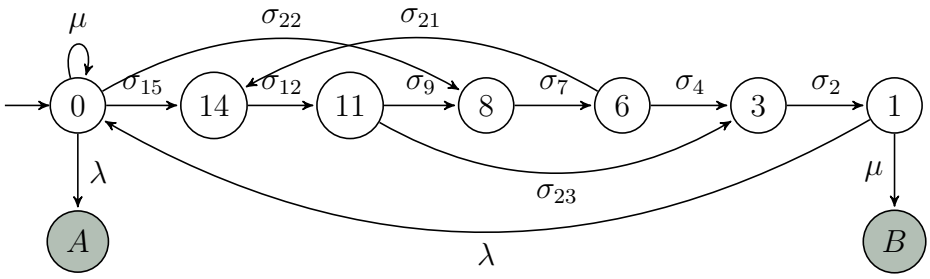


Figure 4.13: Modified automaton representing the computer network.

uncolored states from Fig. 4.13 are kept. The dashed arrows coming out of states 0 and 1 and their labels represent disabled events.

Different malicious agents can hack into the security logs and add or erase the occurrence of the σ events. According to Theorem 1, it is possible to design a supervisor that enforces the language K if and only if K is controllable with respect to G and P-observable for a given attack set $\mathcal{A}$.

The first step is to find the relation of control inconsistent states $\mathcal{I}$, using Alg. 5. The result obtained is $\mathcal{I} = \{(0,1),(1,0)\}$. This can be seen at Fig. 4.14, where event μ is enabled at state 0 while it is disabled at state 1. No other state has disabled events, making it impossible to find other pairs of control inconsistent states. Consider the attack set $\mathcal{A}_1 = \{A_\emptyset, A_{\{\sigma_{15},\sigma_{12},\sigma_{22}\}}, A_{\{\sigma_{22},\sigma_7\}}, A_{\{\sigma_{23},\sigma_4,\sigma_2\}}\}$. Then, as in Example 4, the proposed visual interpretation was applied over the automaton. By inspecting Fig. 4.14, one can conclude that the language K is not P-observable for the attack set $\mathcal{A}_1$, since attacker $A_{\{\sigma_{22},\sigma_7\}}$ can make states 0 and 6 indistinguishable, while attacker $A_{\{\sigma_{23},\sigma_4,\sigma_2\}}$ can make states 6 and 1 indistinguishable, and consequently, turning 0 and 1 indistinguishable if the two attackers cooperate with each other. Another possibility is the one in which attacker $A_{\{\sigma_{15},\sigma_{12},\sigma_{22}\}}$ makes states 0 and 11 indistinguishable, while attacker $A_{\{\sigma_{23},\sigma_4,\sigma_2\}}$ makes states 11 and 1 indistinguishable. As expected, if one furnished the automaton of Fig. 4.14 and attack set $\mathcal{A}_1$ to Alg. 4, it returns FALSE. The fact that the desired language K is not P-observable for attack set $\mathcal{A}_1$ means that a supervisor that enforces K cannot be designed, according to Theorem 1.

One question that may arise is how to determine the attack set for which to test P-observability. It could depend on the experience of the person responsible for the network's security management, for example. If such knowledge is not available, one can apply the P-observability test proposed, not for a single attack set, but for different scenarios. Using this idea, the P-observability test was applied for all attack sets of the type $\mathcal{A}_k = \{A_{\alpha_k}\}$, for all $\alpha_k \subseteq 2^{\Sigma_v}$, with $\Sigma_v = \{\sigma_{15},\sigma_{12},\sigma_9,\sigma_7,\sigma_4,\sigma_2,\sigma_{23},\sigma_{22},\sigma_{21}\}$. After performing the tests, it was noticed that whenever $\alpha_k \supseteq \Phi_i$, for $i = 1,\ldots,4$ and

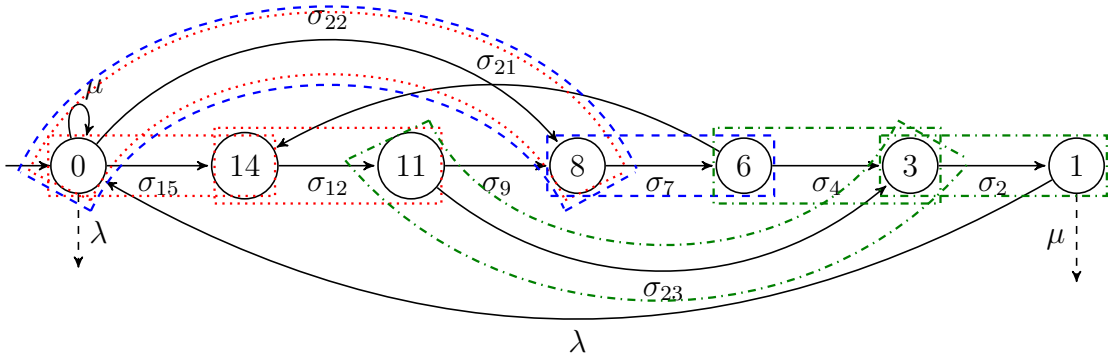


Figure 4.14: Automaton representing the desired language K. The dashed transitions represent disabled events. Dotted (red), dashed (blue) and dotted-dashed (green) borders represent attackers $A_{\{\sigma_{15},\sigma_{12},\sigma_{22}\}}$, $A_{\{\sigma_{22},\sigma_7\}}$ and $A_{\{\sigma_{23},\sigma_4,\sigma_2\}}$, respectively.

$$\Phi_1 = \{\sigma_{22}, \sigma_7, \sigma_4, \sigma_2\};$$

$$\Phi_2 = \{\sigma_{15}, \sigma_{12}, \sigma_{23}, \sigma_2\};$$

$$\Phi_3 = \{\sigma_{15}, \sigma_{12}, \sigma_9, \sigma_7, \sigma_4, \sigma_2\};$$

$$\Phi_4 = \{\sigma_{22}, \sigma_7, \sigma_{21}, \sigma_{12}, \sigma_{23}, \sigma_2\},$$

the language K was not P-observable for the attack set $\mathcal{A}_k$. Each set Φ_i is composed of events belonging to a direct path between states 0 and 1. In other words, any path between states 0 and 1 is composed by events from one of the sets Φ_i. This means that if an attacker is able to act on all events that belong to a direct path between two control inconsistent states, then the language K is not P-observable. The tests were performed considering only one attacker per attack set but different combinations of attack sets can be built from each α_k. For instance, considering $\alpha_k = \Phi_1 = \{\sigma_{22}, \sigma_7, \sigma_4, \sigma_2\}$, the following attack sets can be considered:

$$\mathcal{A}_2 = \{A_{\{\sigma_{22},\sigma_7,\sigma_4,\sigma_2\}}\};$$

$$\mathcal{A}_3 = \{A_{\{\sigma_{22},\sigma_7\}}, A_{\{\sigma_4,\sigma_2\}}\};$$

$$\mathcal{A}_4 = \{A_{\{\sigma_{22},\sigma_4\}}, A_{\{\sigma_7,\sigma_2\}}\}.$$

In Fig. 4.15, the single attacker in the attack set $\mathcal{A}_2$ can act on all the events in Φ_1. It is easy to see that the attacker can make states 0 and 1 indistinguishable, thus making the language K not P-observable.

In Fig. 4.16, there are two attackers in the attack set $\mathcal{A}_3$. Attacker $A_{\{\sigma_{22},\sigma_7\}}$ is able to make states 0 and 6 indistinguishable, while attacker $A_{\{\sigma_4,\sigma_2\}}$ can make states 6 and 1 indistinguishable. Thus, if one allows the two attackers to cooperate, which is only a hypothesis, they can make states 0 and 1 indistinguishable and it is possible to conclude that

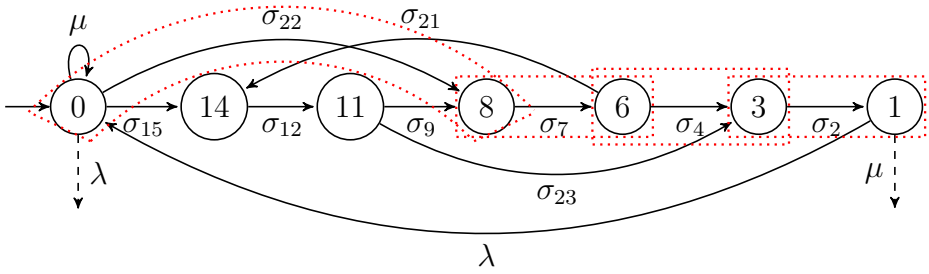


Figure 4.15: Attack set $\mathcal{A}_2 = \{A_{\{\sigma_{22},\sigma_7,\sigma_4,\sigma_2\}}\}$. The dotted (red) borders represent attacker $A_{\{\sigma_{22},\sigma_7,\sigma_4,\sigma_2\}}$.

the language K is not P-observable for the attack set $\mathcal{A}_3$. In this case, the problem appears once a string y such that $\delta(q_0, y) = 6$ is observed, e.g., $y = \sigma_{22}\sigma_7$. As state 6 is simultaneously indistinguishable with states 0 and 1, upon observing y, a supervisor cannot decide which control action to take.

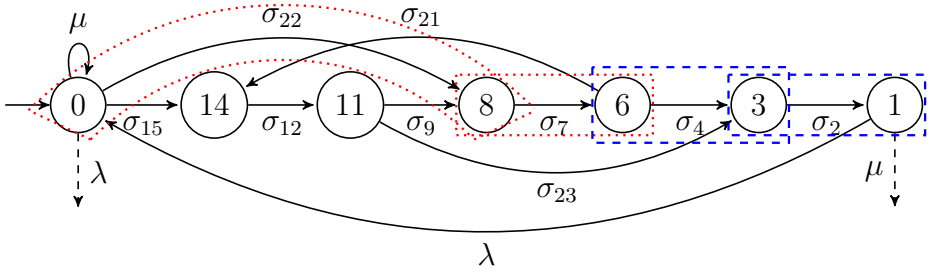


Figure 4.16: Attack set $\mathcal{A}_3 = \{A_{\{\sigma_{22},\sigma_7\}}, A_{\{\sigma_4,\sigma_2\}}\}$. The dotted (red) and dashed (blue) borders represent attacker $A_{\{\sigma_{22},\sigma_7\}}$ and $A_{\{\sigma_4,\sigma_2\}}$, respectively.

Finally, Fig. 4.17 shows the effect of the attack set $\mathcal{A}_4$ over the automaton that implements the language K. For this attack set, K is P-observable, since if $y = \sigma_{22}$ is observed, for instance, one can only conclude that the system is at states 0, 8 or 6. States 3 and 1 are excluded from this estimate because in order to make state 0 indistinguishable with them, the two attackers would need to actually cooperate with each other, e.g., by erasing events σ_7 and σ_4, which is not allowed. This can be seen at Fig. 4.17 where it is only possible to reach state 1 starting from state 0 by inserting vulnerable events that alternately can be manipulated by different attackers, e.g., there are 3 changes of attackers between states 0 and 1.

From another perspective, if Alg. 1 is applied, one obtains

$$\Pi'_{\{\sigma_{22},\sigma_4\}} = \{(0,0),(0,8),(1,1),(3,3),(3,6)(6,3),(6,6),(8,0),(8,8),(11,11),(14,14)\},$$
$$\Pi'_{\{\sigma_7,\sigma_2\}} = \{(0,0),(1,1),(1,3),(3,1),(3,3),(6,6),(6,8),(8,6),(8,8),(11,11),(14,14)\},$$

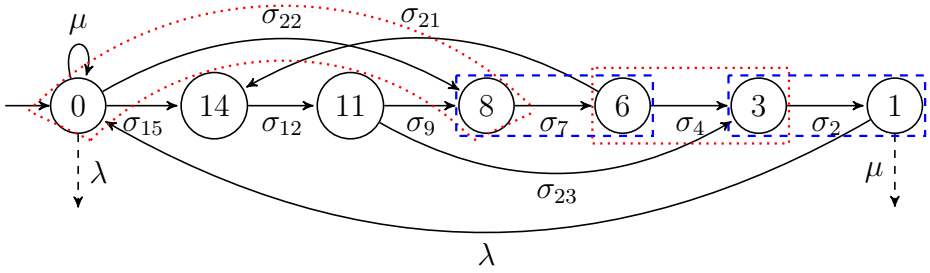


Figure 4.17: Attack set $\mathcal{A}_4 = \{A_{\{\sigma_{22},\sigma_4\}}, A_{\{\sigma_7,\sigma_2\}}\}$. The dotted and dashed borders represent attacker $A_{\{\sigma_{22},\sigma_4\}}$ and $A_{\{\sigma_7,\sigma_2\}}$, respectively.

$$\Pi_{\mathcal{A}_4} = \{(0,0),(0,8),(1,1),(1,3),(3,1),(3,3),(3,6),(6,3),(6,6),(6,8),(8,0),(8,6),(8,8),(11,11),(14,14)\}.$$

The set Q_{MP} is empty. Lastly the relation of indistinguishable states is obtained.

$$\Pi^2_{\mathcal{A}_4} = \{(0,0),(0,6),(0,8),(1,1),(1,3),(1,6),(3,1),(3,3),(3,6),(3,8),(6,0),(6,1),(6,3),(6,6),(6,8),(8,0),(8,3),(8,6),(8,8),(11,11),(14,14)\}.$$

As expected, $\Pi^2_{\mathcal{A}_4}$ does not have the pair $(0,1)$ or $(1,0)$, confirming formally the conclusion that K is P-observable for the attack set $\mathcal{A}_4$.

Thus, the way in which the events in Φ_i are distributed among the attackers in an attack set also has influence in the P-observability of the language K. Furthermore, by inspecting the sets Φ_i, it was noticed that event σ_2 is a common element among them. By making sure that σ_2 is not vulnerable, then the language K will be P-observable for any attack set, where the new set of vulnerable events is $\Sigma'_v = \Sigma_v \setminus \{\sigma_2\}$. Strategies to transform a vulnerable event in a non-vulnerable one depend on the context and can include the addition of redundant sensors or the employment of sensors that rely on more secure protocols for communication.

Chapter 5

Persistent attacks in Discrete-Event Systems

In this chapter a new model of deception attacks at the supervisory control layer of a control system is introduced and the problem of synthesis of *successful persistent deception attacks* is presented. Although this thesis does not propose a method for defending against attacks, understanding how attackers are designed will aid in developing defense strategies.

5.1 Problem formulation

The problem tackled in this chapter consists of the design of a persistent attacker, which is an attacker that infects a control system and wants to remain undetected. It will not necessarily act whenever it can, but it will eventually tamper with the communication between devices, in order to accomplish its goal. Since the attacker is persistent, as a consequence, it is also stealthy, or otherwise, its presence will be revealed. In this sense, an attacker is considered to be stealthy if: (i) the behavior of the system under attack from the point of view of the IDS (Intrusion Detection System) cannot be distinguished from the behavior with no attack and; (ii) the behavior of the system under attack from the point of view of a human operator will be similar to the behavior of the system with no attack. To be similar means that machines in the system should not start or stop operating for no apparent reason and the machines should not be prevented from starting work when they are supposed to.

5.1.1 System setup

Before presenting the attack model, the setup of the system under consideration is presented, which will have impact on the mathematical description of the attacker.

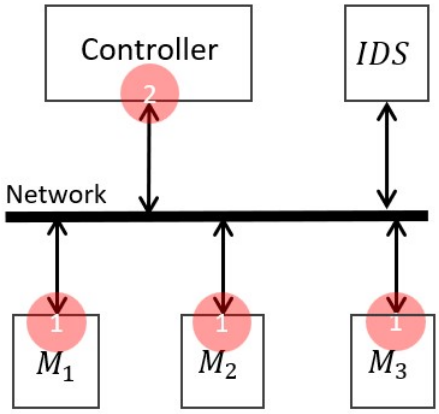


Figure 5.1: System with IDS. The red circles represent the possible attack locations.

Consider a system composed by multiple subsystems, which are coordinated by a centralized controller, as shown in Fig. 5.1. The subsystems and controller are connected to a bus network, through which they exchange information. Because of the network's topology, whenever a message is sent by one of the devices connected to it, all other devices will receive the message. Each device has the ability to check if the message was addressed to itself and process the information in the positive case. Otherwise, the message is ignored.

In this work the ideas of Balemi et al. (1993); Dietrich et al. (2002); Vieira et al. (2017), among others, are adopted. The controller has the role of generating commands to be sent to the subsystems while the subsystems generate responses that are sent to the controller. A command can be associated with a controllable event, while a response is associated with an uncontrollable event. As a consequence, any subsystem will ignore any uncontrollable event received while the controller ignores any controllable event that it might receive.

The monolithic supervisor or the local modular supervisors are encoded inside the controller, that coordinates the subsystems by deciding which controllable events should be triggered. This decision takes into account the control action of the supervisors and an event priority rule. The details about how the controller is implemented are out of the scope of this chapter, since only the controller's behavior as seen by the network is of interest. Such details will be presented in the next chapter.

The IDS is a special device that monitors the network, verifying if the events that are being exchanged respect the legal behavior of the system. If it detects an abnormal behavior, it triggers an alarm for a human operator. It is assumed that the IDS is immune to attacks.

An attack can happen in location 1 or 2, represented by the red circles in Fig. 5.1. In this work, only attacks at location 1 will be considered. Furthermore, it is assumed that only

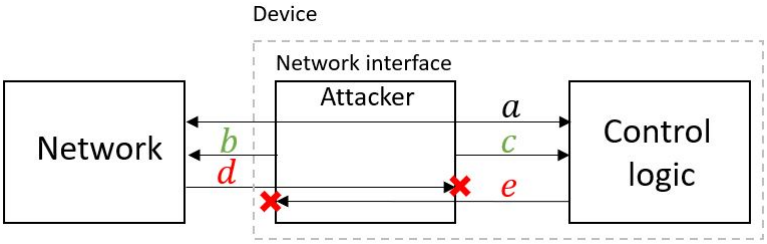


Figure 5.2: Possible actions of the attacker

one of the subsystems is under attack.

5.1.2 Actions of the attacker

Once an attacker has infiltrated one of the subsystems, the possible actions that the attacker can perform are shown in Fig. 5.2, where

- a: the attacker has not interfered in the communication;
- b: the attacker inserted an event in the network;
- c: the attacker inserted an event in the device observation;
- d: the attacker erased an event that came from the network;
- e: the attacker erased an event that was sent by the device.

Consider that the attacker has infiltrated subsystem G_i and let Σ_i denote the alphabet of events related to it. The actions of the attacker can be represented by the following sets of events:

- Σ_i: (action a of Fig. 5.2) events that are not tampered with by the attacker;
- $\Sigma_i^{+N} := \{\sigma^{+N} | \sigma \in \Sigma_i\}$: (action b of Fig. 5.2) events that are inserted in the network. The attacker is able to insert any event in Σ_i. In this case, all devices in the network, except the one that is infected by the attacker, can see the event;
- $\Sigma_{i,uc}^{-N} := \{\sigma^{-N} | \sigma \in \Sigma_i \cap \Sigma_{uc}\}$: (action e of Fig. 5.2) events that are erased, preventing them to get to the network. Note that the attacker can only erase uncontrollable events in Σ_i since these are the only events generated by the subsystem. In this case, only the device that is infected by the attacker can see the event;
- $\Sigma_{i,c}^{+G} := \{\sigma^{+G} | \sigma \in \Sigma_i \cap \Sigma_c\}$: (action c of Fig. 5.2) events that are inserted in the device's observation. Note that the attacker can only insert controllable events in Σ_i since these are the only events expected to be received by the subsystem. Other events will be ignored by it. In this case, only the infected plant sees the inserted event;

- $\Sigma_{i,c}^{-G} := \{\sigma^{-G} | \sigma \in \Sigma_i \cap \Sigma_c\}$: (action d of Fig. 5.2) events that are erased from the device's observation. Note that the attacker can only erase controllable events in Σ_i since these are the only events expected to be received by the subsystem. The event has traveled through the network and was seen by all other devices, except the plant under attack.

The set of all events, the ones that are legitimate and the ones that represent the actions of the attacker, is represented by

$$\Sigma_{ai} := \Sigma_i \cup \Sigma_i^{+N} \cup \Sigma_{i,uc}^{-N} \cup \Sigma_{i,c}^{+G} \cup \Sigma_{i,c}^{-G}$$

and is called the *attacked alphabet*. Note that an element $\sigma^{-N} \in \Sigma_{i,uc}^{-N}$ or $\sigma^{-G} \in \Sigma_{i,c}^{-G}$ represent that the legitimate event σ has occurred and it was erased by the attacker. In the remainder of the text, strings defined in Σ_{ai} are denoted by w or v while strings defined in Σ_i are denoted by s, including superscripts and subscripts.

Independently of which subsystem is under attack, there are two different points of view: the one from the infected device and the one from all other devices in the network, including the supervisor, that share the same observation. To obtain only the events from the attacked alphabet that are seen by the plant, the map, called the *plant projection*, $P^G : \Sigma_{ai} \to (\Sigma_i \cup \{\varepsilon\})$ can be applied:

$$P^G(\sigma^a) := \begin{cases} \sigma \text{ if } \sigma^a \in \Sigma_{ai} \setminus (\Sigma^{+N} \cup \Sigma_{i,c}^{-G}) \\ \varepsilon \text{ if } \sigma^a \in (\Sigma^{+N} \cup \Sigma_{i,c}^{-G}). \end{cases} \tag{5.1}$$

In a similar manner, to obtain only the events from the attacked alphabet that are seen from the network's point of view, the *network projection* $P^N : \Sigma_{ai} \to (\Sigma_i \cup \{\varepsilon\})$ can be applied:

$$P^N(\sigma^a) := \begin{cases} \sigma \text{ if } \sigma^a \in \Sigma_{ai} \setminus (\Sigma_{i,uc}^{-N} \cup \Sigma_{i,c}^{+G}) \\ \varepsilon \text{ if } \sigma^a \in (\Sigma_{i,uc}^{-N} \cup \Sigma_{i,c}^{+G}). \end{cases} \tag{5.2}$$

Now, the plant and network projection are extended so they can be applied over strings. To obtain the observation of string w, from the infected device's point of view, one can apply the plant projection $P^G : \Sigma_{ai}^* \to \Sigma_i^*$, defined as follows:

$$P^G(\varepsilon) := \varepsilon \tag{5.3}$$

$$P^G(w\sigma^a) := \begin{cases} P^G(w)\sigma \text{ if } \sigma^a \in \Sigma_{ai} \setminus (\Sigma^{+N} \cup \Sigma_{i,c}^{-G}) \\ P^G(w) \text{ if } \sigma^a \in (\Sigma^{+N} \cup \Sigma_{i,c}^{-G}). \end{cases} \tag{5.4}$$

That is, events in Σ^{+N} and $\Sigma_{i,c}^{-G}$ cannot be seen by the infected device. To obtain the observation as it is seen by all other devices in the network, the *network projection*, $P^N : \Sigma_{ai}^* \to$

Σ_i^*, defined as:

$$P^N(\varepsilon) := \varepsilon \tag{5.5}$$

$$P^N(w\sigma^a) := \begin{cases} P^N(w)\sigma \text{ if } \sigma^a \in \Sigma_{ai} \setminus (\Sigma_{i,uc}^{-N} \cup \Sigma_{i,c}^{+G}) \\ P^N(w) \text{ if } \sigma^a \in (\Sigma_{i,uc}^{-N} \cup \Sigma_{i,c}^{+G}), \end{cases} \tag{5.6}$$

can be applied. If a string $w \in \Sigma_{ai}^*$ represents the sequence of actions of the attacker, then a string $s = P^G(w)$ is the sequence of events observed by the plant under attack while $s' = P^N(w)$ is the sequence of events observed by the network.

5.1.3 Attack model

An attacker A is an agent that implements an attack function f_A, which in turn, is described by 1) the subsystem G_i under attack and; 2) a set of rules that gives the available options of actions that the attacker can take according to the evolution of G_i. More details on the attack function will be given later. An attacker that has infected the network interface of a given subsystem is able to:

- Erase a controllable event from the subsystem's observation or not tamper with it when such event is sent by the controller;
- Erase an uncontrollable event from the network's observation or not tamper with it when such event is generated by the subsystem under attack and;
- Insert any event into the network's observation or insert controllable events into the subsystem's observation or do nothing, when no legitimate event is generated by the system.

An attacker that is able to act on all events is an attacker that is able to perform multiple types of attacks including the ones that cause the system to block and the ones that can even produce physical damage in the machines, depending on the physical system. Such attacks can achieve the attacker's goal with the cost of being exposed to a human operator, that will likely perform a scan of the system in order to find the attacker. In contrast, in this work, the interest is in the types of attacks that allow the attacker to remain hidden even after achieving a successful attack, so that it can keep attacking without being revealed. Such attackers are called *persistent attackers*. A persistent attacker is considered to be successful if an attacker A is able to cause the production of off-specification products or to introduce small delays in the process, while the discrete behavior of the controlled system under attack is indistinguishable from the behavior of the controlled system with no attack, as seen by the IDS. In this work, it is assumed that the attacker knows the plants and supervisors, as well as the physical process. Let A be an attacker acting over a subsystem G_i, which is represented by

A/G_i. The attacker modifies the original behavior $\mathcal{L}(G_i)$ to a behavior of G_i under attack of A, denoted by $\mathcal{L}(A/G_i)$. The concept of a persistent attacker is formalized in the next definition.

Definition 11 (Persistent attacker)**.** *An attacker A acting over a subsystem G_i with associated attack function f_A is called a persistent attacker if $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$.* $\Diamond$

According to Def. 11, an attacker is persistent if for every string $w \in \mathcal{L}(A/G_i)$, then $P^N(w) \in \mathcal{L}(G_i)$, which means that the behavior of the subsystem under attack is indistinguishable from the behavior with no attack, when observed from the network's point of view.

Furthermore, it is desirable to also minimize the possibility of a human operator detecting an abnormal behavior of the system, which can reveal the attacker's presence. For this reason, the attacker won't insert or remove controllable events in the device's observation. Tampering with controllable events will certainly cause an explicit abnormal behavior of the system, drawing the undesirable attention of a human operator.

Under the presumption that controllable events are not going to be attacked, tampering with uncontrollable events may also cause the controller to not respect specifications. This situation is shown through the next example.

Example 13. *Consider the system shown in Fig. 5.3. It represents two machines, M_1 and M_2, which are connected through an unity buffer. The models of machines M_1 and M_2 are shown in Figs. 5.4(a) and 5.4(b), respectively. For this problem, $\Sigma_c = \{a_1, a_2\}$ and $\Sigma_{uc} = \{b_1, b_2\}$. The automaton of Fig. 5.4(c) models the specification, which has the goal of avoiding underflow and overflow in the buffer.*

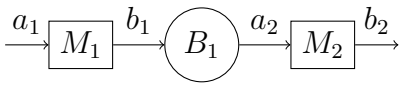
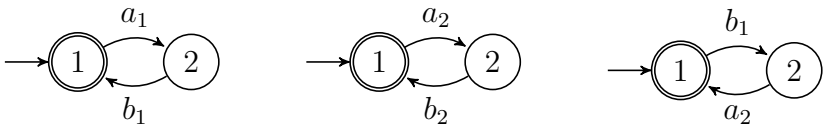


Figure 5.3: Simplified small factory with unity buffer.



(a) M_1 (b) M_2 (c) E_1

Figure 5.4: Models and specification of the simplified small factory problem.

Now suppose that machine M_1 is under attack. Consider the sequence of the attacker's actions shown in the first column of Table 5.1. At each line a new event has happened and the second and third columns show the observation of the controller and of the machine M_1,

respectively. The last column shows the current states of M_1, M_2 and E_1, respectively, for each new event. Note that after the occurrence of event a_1 the attacker inserts event b_1, (action represented by b_1^{+N}) into the network. This causes the controller to trigger event a_2 before the occurrence of the legitimate event b_1. Thus, the specification is violated, resulting in underflow in the buffer. This can also be seen by analyzing the last column, in which the state of machine M_2 goes from 1 to 2 before the state of machine M_1 goes back to 1, characterizing the underflow in the buffer. Depending on the physical system, this situation can cause some damage in the machines since the second machine will start operating with no product.

Table 5.1: Sequence of strings that cause underflow in the buffer.

w	$P^N(w)$	$P^{M_1}(w)$	q_{M_1}, q_{M_2}, q_E
ε	ε	ε	1, 1, 1
a_1	a_1	a_1	2, 1, 1
$a_1 b_1^{+N}$	$a_1 b_1$	a_1	2, 1, 2
$a_1 b_1^{+N} a_2$	$a_1 b_1 a_2$	a_1	2, 2, 1
$a_1 b_1^{+N} a_2 b_1^{-N}$	$a_1 b_1 a_2$	$a_1 b_1$	1, 2, 1
$a_1 b_1^{+N} a_2 b_1^{-N} b_2$	$a_1 b_1 a_2 b_2$	$a_1 b_1$	1, 1, 1

□

The situation described in Example 13 will likely draw the attention of a human operator. To avoid this problem, the attacker is not allowed to insert an uncontrollable event in the network's observation if this event causes a state change in the supervisor while the state in the plant has not changed. However, as will be seen later, attacks on events which cause state changes can result in a persistent attack under certain circumstances.

First, three types of attack functions are introduced, which take into account the restrictions over attacks on controllable and uncontrollable events and provide a set of rules that will allow one to obtain persistent attackers, as will be seen later. For some attack functions, it is important for the attacker to keep track of whether its last action with respect to a given event was an insertion or a deletion. In particular, if an event σ causes a state change, then the attacker can only insert σ^{+N} if the last action regarding this event was its erasure, represented by σ^{-N}. To allow the attacker to check which was the last action, the map $M : \Sigma_i \to \Sigma_{ai}^*$ is defined as follows:

$$M(\sigma) := \sigma^{-N}(\sigma^{+N}\sigma^{-N})^* \tag{5.7}$$

The map defined in (5.7) receives an event $\sigma \in \Sigma_i$ and returns a language composed only of strings that finish with event $\sigma^{-N} \in \Sigma_{i,uc}^{-N}$ interleaved with events $\sigma^{+N} \in \Sigma_{i,uc}^{+N}$. Now, if $w \in \Sigma_{ai}^*$ represents a string of past actions of the attacker and one wants to know if the last action of the attacker with regard to an event σ was an insertion or deletion, the first step is to take a string w and keep from it only events σ^{-N} and σ^{+N}. This is done by obtaining $P_{\Sigma_{ai}\to\{\sigma^{-N},\sigma^{+N}\}}(w)$. Finally, if $P_{\Sigma_{ai}\to\{\sigma^{-N},\sigma^{+N}\}}(w) \in M(\sigma)$, it means that

the last action of the attacker regarding event σ was a deletion. On the other hand, if $P_{\Sigma_{ai}\to\{\sigma^{-N},\sigma^{+N}\}}(w) \notin M(\sigma)$, then the last action of the attacker with respect to event σ was an insertion or the attacker has not acted over event σ yet. The map $R^1_{G_i} : Q \to \Sigma_i^1$ is also defined as $R^1_{G_i}(q) = \{s \in \Sigma_i^1 | \delta_i(q,s)!\}$, which gives all the strings s of length 1 that can be executed from the state q in G_i.

Now it is possible to characterize three attack functions that can be used by a persistent attacker, according to the next definition.

Definition 12 (Attack functions)**.** *Consider a subsystem $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$, a string $w \in \Sigma_{ai}^*$ representing the past actions of the attacker that leads G_i to state $q = \delta_i(q_0, P^G(w))$ and that allows any device in the network to make a state estimate $\tilde{q} = \delta_i(q_0, P^N(w))$ and a string $s \in R^1_{G_i}(q)$. The following three functions $f_A : \Sigma_{ai}^* \times (\Sigma_i^1 \cup \{\varepsilon\}) \to (2^{\Sigma^1_{ai}} \setminus \emptyset) \cup \{\varepsilon\}$ are called attack functions:*

1. *Passive mode, denoted by f_A^P:*

$$f_A^P(w,s) = \{s\} \tag{5.8}$$

2. *Delay mode, denoted by f_A^D:*

$$f_A^D(w,s) = \begin{cases} \{\sigma\} & \text{if } s = \sigma \in \Sigma^1_{i,c}; \\ \{\sigma^{-N}\} & \text{if } s = \sigma \in \Sigma^1_{i,uc}; \\ \{\varepsilon, \lambda^{+N}\} & \text{if } s = \varepsilon. \end{cases} \tag{5.9}$$

$\forall \lambda^{+N} \in (\Sigma^{+N}_{i,uc})^1$, *such that (5.10) and* (5.11) *always hold:*

$$\lambda \in \Gamma(\tilde{q}) \tag{5.10}$$

$$\delta_i(\tilde{q}, P^N(\lambda^{+N})) = \tilde{q} \vee [\delta_i(\tilde{q}, P^N(\lambda^{+N})) \neq \tilde{q} \wedge P_{\Sigma_{ai}\to\{\lambda^{+N},\lambda^{-N}\}}(w) \in M(\lambda)]. \tag{5.11}$$

3. *Forward mode, denoted by f_A^F:*

$$f_A^F(w,s) = \begin{cases} \{\sigma\} & \text{if } s = \sigma \in \Sigma^1_i; \\ \{\varepsilon, \lambda^{+N}\} & \text{if } s = \varepsilon. \end{cases} \tag{5.12}$$

$\forall \lambda^{+N} \in (\Sigma^{+N}_{i,uc})^1$, *such that*

$$\delta_i(q, P^N(\lambda^{+N})) = q. \tag{5.13}$$

$\Diamond$

Definition 12 introduces three attack functions. In the first one, called passive mode and defined by (5.8), the attacker does not interfere with the events and it is used when the attacker only wants to observe the occurrence of events in the subsystem, without changing them.

In the second one, defined by (5.9) and called delay mode, the attacker introduces delays in the execution of the subsystem by having the ability to erase from the network's observation any uncontrollable event σ that is generated by G_i. Also, the attacker is able to insert any event $\lambda^{+N} \in (\Sigma_{i,uc}^{+N})^1$ if λ^{+N} meets the conditions in (5.10) and (5.11). The condition expressed by (5.10) states that λ has to be feasible at state $\tilde{q}$. Note that $\tilde{q}$ is not necessarily equal to the actual current state q of G_i. In fact, q and $\tilde{q}$ are different if the attacker has erased an event that causes a state change in G_i. Thus, if a device in the network thinks that G_i is at state $\tilde{q}$, then the attacker can only insert events that are feasible at that state or otherwise the attacker is revealed. Condition (5.11) refines even more the condition expressed by (5.10). Event λ can be inserted in the network in two cases. The first one occurs when λ appears as a self-loop at state $\tilde{q}$. The second one handles the case where λ is not in self-loop. In this case, the attacker can only insert λ into the network if the last action taken regarding event λ was its erasure, which can be tested by verifying that $P_{\Sigma_{ai} \to \{\lambda^{+N}, \lambda^{-N}\}}(w)$ belongs to $M(\lambda)$.

Finally, the third attack function, which is called forward mode and is defined by (5.12), allows the attacker to anticipate the occurrence of events by inserting them into the network. The restriction, expressed by (5.13), is that the attacker can only anticipate events that are in self-loops in G_i at state q.

Remark 4. *A string $w \in \Sigma_{ai}^*$ cannot be executed directly in G_i, since the alphabets Σ_{ai} and Σ_i are different. Thus, to obtain the effect of w over G_i, one has to first translate w into a string that can be executed in G_i. This is done by applying the plant projection over w. Hence, $q = \delta_i(q_0, P^G(w))$ is the current state of G_i. In contrast, $\tilde{q} = \delta_i(q_0, P^N(w))$ gives the current state estimate that any device in the network can do about G_i after having observed $P^N(w)$.* □

Next, an example that shows an application of the attack functions is presented.

Example 14. *Consider the subsystem G_i, shown in Fig. 5.5, where $\Sigma_{i,c} = \{a\}$ and $\Sigma_{i,uc} = \{b, c, d\}$. Tables 5.2 to 5.4 show the possible actions of the attacker for each attack function, considering some possible different strings w.*

Table 5.2 illustrates the passive mode acting over G_i. One can observe that, independently of w, the output of the attack function is exactly the second argument it receives (which is a string representing what just happened in G_i), which means that the attacker is only observing the execution of the system.

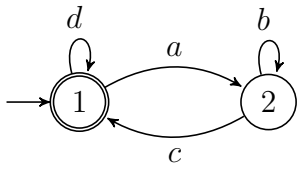


Figure 5.5: Automaton G_i of Example 14.

Table 5.2: Passive mode over G_i of Fig. 5.5.

w	q	$\tilde{q}$	s	$f_A^P(w,s)$
	1	1	ε	$\{\varepsilon\}$
ε	1	1	d	$\{d\}$
	1	1	a	$\{a\}$
	1	1	ε	$\{\varepsilon\}$
d	1	1	d	$\{d\}$
	1	1	a	$\{a\}$
	2	2	ε	$\{\varepsilon\}$
da	2	2	b	$\{b\}$
	2	2	c	$\{c\}$
	1	1	ε	$\{\varepsilon\}$
dac	1	1	d	$\{d\}$
	1	1	a	$\{a\}$

Under the action of the delay mode, illustrated in Table 5.3, whenever an uncontrollable event happens in G_i, the only option for the attacker is to erase it. Starting from the initial state of G_i, where $w = \varepsilon$, while no event occurs, the attacker is able to do nothing or to insert event d in the network. If d happens, then the attacker has to erase it. On the other hand, if a (controllable) is received by G_i, then the attacker cannot interfere with it. Now suppose event d happened and the attacker chose to erase it, which means that $w = d^{-N}$. The attacker is still able to insert d into the network, erase event d if it eventually occurs again or to not interfere in the case that event a was received by G_i. Note that because event d is in self-loop at state 1, whenever the state estimate $\tilde{q} = 1$, the attacker is able to insert d into the network.

Next, consider that $w = d^{-N}d^{+N}a$, which means that G_i is at state 2, where events b and c are feasible. Because b is in self-loop at that state, the attacker can always insert it to the network. Notice that this is not the case for event c, which causes a state change. At this point, the only action the attacker can take with respect to event c is to erase it if it occurs.

For $w = d^{-N}d^{+N}ab^{-N}c^{-N}$, which represents that events b and c happened and were erased by the attacker, the possible actions are to insert those events into the network or do nothing, while nothing happens in G_i. Although the state estimate is $\tilde{q} = 2$, the actual state q of G_i is 1, where event d can happen. If this is the case, the attacker has to erase event d, or otherwise its presence is revealed.

Furthermore, while $\tilde{q} = 2$, the attacker can insert event b in the network multiple times. This is the case when $w = d^{-N}d^{+N}ab^{-N}c^{-N}b^{+N}$, where the attacker can insert events c and b at any time. However, after inserting c, which causes the state estimate to be updated to $\tilde{q} = 1$, the attacker can no longer insert event b into the network. Also, notice that event c can only be inserted once after it is erased, since it is an event that causes a state change in G_i.

Table 5.3: Delay mode over G_i of Fig. 5.5.

w	q	$\tilde{q}$	s	$f_A^D(w,s)$
ε	1	1	ε	$\{\varepsilon, d^{+N}\}$
	1	1	d	$\{d^{-N}\}$
	1	1	a	$\{a\}$
d^{-N}	1	1	ε	$\{\varepsilon, d^{+N}\}$
	1	1	d	$\{d^{-N}\}$
	1	1	a	$\{a\}$
$d^{-N}d^{+N}$	1	1	ε	$\{\varepsilon, d^{+N}\}$
	1	1	a	$\{a\}$
$d^{-N}d^{+N}a$	2	2	ε	$\{\varepsilon, b^{+N}\}$
	2	2	b	$\{b^{-N}\}$
	2	2	c	$\{c^{-N}\}$
$d^{-N}d^{+N}ab^{-N}$	2	2	ε	$\{\varepsilon, b^{+N}\}$
	2	2	b	$\{b^{-N}\}$
	2	2	c	$\{c^{-N}\}$
$d^{-N}d^{+N}ab^{-N}c^{-N}$	1	2	ε	$\{\varepsilon, b^{+N}, c^{+N}\}$
	1	2	d	$\{d^{-N}\}$
$d^{-N}d^{+N}ab^{-N}c^{-N}b^{+N}$	1	2	ε	$\{\varepsilon, b^{+N}, c^{+N}\}$
	1	2	d	$\{d^{-N}\}$
$d^{-N}d^{+N}ab^{-N}c^{-N}b^{+N}c^{+N}$	1	1	ε	$\{\varepsilon, d^{+N}\}$
	1	1	a	$\{a\}$
	1	1	d	$\{d^{-N}\}$

Finally, Table 5.4 illustrates the actions of the attacker when it is operating under the forward mode. Notice that whenever an event σ is in self-loop at the current state q, the attacker is able to insert it to the network or do nothing. That is the reason that event d can be inserted while G_i is at state 1 and event b can be inserted when G_i is at state 2, but the attacker is never able to insert event c, since this event causes a state change. When event c is generated by G_i, the attacker cannot interfere with it.

It is important to highlight that among the three attack functions given in Def. 12, only the delay mode causes the current state q of G_i and its estimate $\tilde{q}$ to be different at some point. For the other two attack functions, q and $\tilde{q}$ are always equal. □

Next, it is shown how one can obtain the behavior of G_i when it is operating under the

Table 5.4: Forward mode over G_i of Fig. 5.5.

w	q	$\tilde{q}$	s	$f_A^F(w,s)$
	1	1	ε	$\{\varepsilon, d^{+N}\}$
ε	1	1	d	$\{d\}$
	1	1	a	$\{a\}$
	1	1	ε	$\{\varepsilon, d^{+N}\}$
d^{+N}	1	1	d	$\{d\}$
	1	1	a	$\{a\}$
	2	2	ε	$\{\varepsilon, b^{+N}\}$
$d^{+N}a$	2	2	b	$\{b\}$
	2	2	c	$\{c\}$
	2	2	ε	$\{\varepsilon, b^{+N}\}$
$d^{+N}ab^{+N}$	2	2	b	$\{b\}$
	2	2	c	$\{c\}$
	1	1	ε	$\{\varepsilon, d^{+N}\}$
$d^{+N}ab^{+N}b^{+N}c$	1	1	d	$\{d\}$
	1	1	a	$\{a\}$

action of attacker A, which is denoted by $\mathcal{L}(A/G_i)$.

Definition 13 (Behavior under attack)**.** *Given a subsystem $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$ and an attack function $f_A : \Sigma_{ai}^* \times (\Sigma_i^1 \cup \{\varepsilon\}) \to (2^{\Sigma_{ai}^1} \setminus \emptyset) \cup \{\varepsilon\}$, the behavior of the subsystem G_i under attack of A with attack function f_A, is denoted by $\mathcal{L}(A/G_i) \subseteq \Sigma_{ai}^*$ and can be obtained as follows:*

i. $\varepsilon \in \mathcal{L}(A/G_i)$

ii. If $w \in \mathcal{L}(A/G_i)$, $s \in \Sigma_i^1 \cup \{\varepsilon\}$ *such that* $P^G(w)s \in \mathcal{L}(G_i)$ *and* $v^a \in f_A(w,s)$, *then* $wv^a \in \mathcal{L}(A/G_i)$;

iii. No other strings belong to $\mathcal{L}(A/G_i)$.

$\Diamond$

Definition 13 shows how one can build the behavior under attack $\mathcal{L}(A/G_i)$, starting with ε and by adding new strings that are formed by concatenating a string $w \in \mathcal{L}(A/G_i)$ with a new string $v^a \in f_A(w,s)$, where $s \in \Sigma_i^1 \cup \{\varepsilon\}$ is a single-event string that is feasible at the current state of G_i or is the empty string ε.

The first result shows that an attacker A that uses any of the attack functions of Def. 12 to guide its actions is a persistent attacker.

Theorem 4. *An attacker A associated with an attack function of Def. 12 is a persistent attacker (Def. 11), that is, $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$.* $\blacklozenge$

Proof. To prove Theorem 4, it is necessary to show that $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$ is true when attacker A is associated with each of the three attack functions of Def. 12. That is, for every string $w \in \mathcal{L}(A/G_i)$, it is necessary to show that $P^N(w) \in \mathcal{L}(G_i)$. This will be done inductively on the length of w.

Base case: According to Def. 13,

$$w = \varepsilon \in \mathcal{L}(A/G_i).$$

As $P^N(\varepsilon) = \varepsilon$, this allows one to say that

$$P^N(\varepsilon) \in \mathcal{L}(G_i), \tag{5.14}$$

which is valid for all three attack functions.

Induction step: As the induction hypothesis, assume that for a string $w \in \mathcal{L}(A/G_i)$, such that $|w| = k$, it holds that $P^N(w) \in \mathcal{L}(G_i)$. Also, according to Def. 13ii, a new string $w' = wv^a \in \mathcal{L}(A/G_i)$, of length $k+1$, is obtained by concatenating w with a string $v^a \in f_A(w,s)$, with $s \in \Sigma_i^1 \cup \{\varepsilon\}$ and $P^G(w)s \in \mathcal{L}(G_i)$. Notice that it is not possible to consider for the induction step the case where $\varepsilon \in f_A(w,s)$, since that would lead to a string w' such that $|w'| = |w| = k$. However, if $\varepsilon \in f_A(w,s)$, then $w' = w$ and it holds that $P^N(w') \in \mathcal{L}(G_i)$, according to the induction hypothesis.

For the passive mode, defined by (5.8) in Def. 12:

$$\begin{aligned} v^a &\in f_A^P(w,s) = \{s\} \\ v^a &= s \end{aligned} \tag{5.15}$$

Furthermore, according to (5.8), the string w is only composed by events in Σ_i, which allows one to say that

$$P^G(w) = P^N(w). \tag{5.16}$$

Also,

$$\begin{aligned} P^N(w') &= P^N(wv^a) \\ &= P^N(w)P^N(v^a) \\ &= P^N(w)P^N(s) && \text{from (5.15)} \\ &= P^N(w)s && \text{since it is always true that } s \in \Sigma_i^1 \cup \{\varepsilon\} \\ &= P^G(w)s && \text{from (5.16).} \end{aligned} \tag{5.17}$$

Since $P^G(w)s \in \mathcal{L}(G_i)$, then, from (5.17), one can say that $P^N(w') \in \mathcal{L}(G_i)$, proving that $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$ for the passive mode.

For the delay mode, recall that $P^N(w) \in \mathcal{L}(G_i)$. Since $v^a \in f_A^D(w,s)$, with $P^G(w)s \in \mathcal{L}(G_i)$, as defined in Def. 12, there are three possibilities:

a) $f_A^D(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma_{i,c}^1$. Then $v^a = \sigma$, $P^N(v^a) = \sigma$ and $P^N(w') = P^N(wv^a) = P^N(w)\sigma$. Because σ is a controllable event, it is possible to say that $P^N(w)\sigma \in \mathcal{L}(G_i)$, since it is assumed that the controller will only trigger an event σ if it is feasible after $P^N(w)$;

b) $f_A^D(w,s) = \{\sigma^{-N}\}$, if $s = \sigma \in \Sigma_{i,uc}^1$. Then $v^a = \sigma^{-N}$, $P^N(v^a) = \varepsilon$ and $P^N(w') = P^N(wv^a) = P^N(w) \in \mathcal{L}(G_i)$;

c) $f_A^D(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and $P^N(v^a) = \lambda$. Additionally, according to (5.10), $\lambda \in \Gamma(\tilde{q})$ and $\tilde{q} = \delta_i(q_0, P^N(w))$, which allows one to say that $P^N(w)\lambda \in \mathcal{L}(G_i)$.

For each of the above three cases, it is possible to say that $P^N(w') \in \mathcal{L}(G_i)$, allowing one to conclude that $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$ for the delay mode.

Finally, for the forward mode, defined in Def. 12, there are two possibilities:

a) $f_A^F(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma_i^1$. Then, $v^a = \sigma$, $P^N(v^a) = \sigma$ and $P^N(w') = P^N(wv^a) = P^N(w)\sigma \in \mathcal{L}(G_i)$;

b) $f_A^F(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and $P^N(v^a) = \lambda$. Also, according to (5.13), one can say that $\delta_i(q_0, P^N(w\lambda^{+N}))$ is defined, which allows one to say that $P^N(w\lambda^{+N}) = P^N(w)\lambda \in \mathcal{L}(G_i)$.

Hence, one can conclude that $P^N(w') = P^N(ww^a) \in \mathcal{L}(G_i)$ and consequently, $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$ for the forward mode and, therefore, $P^N(\mathcal{L}(A/G_i)) \subseteq \mathcal{L}(G_i)$ for all attack functions of Def. 12. □

It is important to highlight that Theorem 4 states only that if an attacker is associated with an attack function of Def. 12, then it is a persistent attacker. The converse does not need to be true. In fact, a persistent attacker may have an attack function that is a combination of the attack functions of Def. 12. A systematic method for obtaining all possible combinations of the attack functions of Def. 12 will be presented later. First, it is shown how an attack function is obtained from an automaton.

5.1.4 Attack function represented as an automaton

Let $G_A = (Q_A, \Sigma_{ai}, \delta_A, q_0, Q_m)$ be an automaton that represents an attack function f_A acting over $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$, with $Q \subseteq Q_A$ and q_0 and Q_m be the same in both automata. Then f_A is an attack function such that $\forall w \in \mathcal{L}(G_A)$, $q = \delta_A(q_0, w)$ and $\forall s \in P^G(R_{G_A}^1(q)) \cup \{\varepsilon\}$:

$$f_A(w,s) = \begin{cases} R^1_{G_i}(q) \cap (\Sigma^{+N}_{i,uc})^1 \cup \{\varepsilon\} & \text{if } s = \varepsilon \\ R^1_{G_i}(q) \cap \Sigma^1_{i,uc} \cap (\Sigma^{-N}_{i,uc})^1 & \text{if } s = \sigma \in \Sigma^1_{i,uc} \\ \{\sigma\} & \text{if } s = \sigma \in \Sigma^1_{i,c} \end{cases} \tag{5.18}$$

Equation (5.18) shows how one can extract an attack function f_A by considering each state $q \in Q_A$ and strings $s \in P^G(R^1_{G_A}(q))$ and the empty string ε. Notice that the second argument of the attack function is defined only for strings $s \in \Sigma^1_i \cup \{\varepsilon\}$. That is the reason why the plant projection is applied over $R^1_{G_A}(q)$, so only events in Σ_i are kept. There are three different cases to consider. Case 1) happens when $s = \varepsilon$ and the attacker can do the actions corresponding to events in $\Sigma^{+N}_{i,uc}$ that are feasible at state q under consideration. Additionally, the attacker can always choose to do nothing when there is no occurrence of events in G_i, which is represented by $\varepsilon \in f_A(w,\varepsilon)$. Case 2) handles the case in which $s = \sigma \in \Sigma^1_{i,uc}$ and the attacker can do the actions corresponding to all feasible events at state q that belong to $\Sigma_{i,uc}$ or $\Sigma^{-N}_{i,uc}$. Finally, in case 3) the controllable events are considered, where the only option for the attacker is to leave the event untouched.

Next an example showing how to obtain the attack function from an automaton G_A is presented.

Example 15. *Consider the automaton G_A shown in Fig. 5.6. In order two obtain the attack function f_A represented by G_A, one has to consider strings that lead G_i to each of its states. Starting with $w = \varepsilon$, then $q = \delta_A(q_0, w) = 1$ and $R^1_{G_A}(q) = \{g\}$.*

Now one has to find f_A by considering w and all strings $s \in P^G(R^1_{G_A}(q)) \cup \{\varepsilon\} = \{\varepsilon, g\}$, according to (5.18)*:*

- $f_A(\varepsilon, \varepsilon) = \{\varepsilon\}$
- $f_A(\varepsilon, g) = \{g\}$

which means that the attacker can only observe the occurrence of events at this state. For state $q = 2$, one possible string w is $w = g$ and the set of possible continuations of length 1 at this state is $R^1_{G_A}(q) = \{h, h^{-N}\}$. Thus, one has to obtain f_A for every $s \in P^G(R^1_{G_A}(q)) \cup \{\varepsilon\} = \{\varepsilon, h\}$:

- $f_A(g, \varepsilon) = \{\varepsilon\}$
- $f_A(g, h) = \{h, h^{-N}\}$

At state $q = 2$, when event h happens, the attacker can choose between not interfering with the event or erasing it. Finally, for state $q = 3$, one can consider a string w such as $w = gh^{-N}$. In

this case, $R^1_{G_A}(q) = \{h^{+N}\}$ *and one has to consider every string* $s \in P^G(R^1_{G_A}(q)) \cup \{\varepsilon\} = \{\varepsilon\}$. *Thus:*

- $f_A(gh^{-N}, \varepsilon) = \{\varepsilon, h^{+N}\}$.

At state $q = 3$, *the attacker can choose between doing nothing or inserting event* h *into the network. After the execution of a string* $w' = gh$ *or* $w'' = gh^{-N}h^{+N}$, *the current state* q *of* G_A *is again the initial state and the attack function repeats itself as if* $w = \varepsilon$.

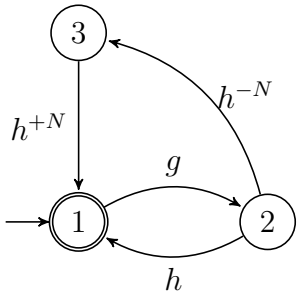


Figure 5.6: Automaton G_A representing an attack function f_A - Example 15

□

As can be seen in Example 15, an attack function can have a finite representation when it is represented as an automaton. Next, it is shown how to obtain the automata corresponding to the attack functions of Def. 12.

Passive mode: An automaton G_i^P that represents the passive mode f_A^P over subsystem G_i is obtained by considering that $G_i^P \simeq G_i$.

Delay mode: An automaton G_i^D that represents the delay mode f_A^D over subsystem G_i is obtained by applying Algorithm 6. Notice that the convention of representing the transition function δ as a set Δ is employed. Any changes in Δ are automatically reflected in the transition function δ.

In Alg. 6, every transition in G_i labeled with events in $\Sigma_{i,uc}$ is replaced by at least two new transitions while the original one is removed. For this to be possible, a new state q_σ is added to Q_A for every event in $\sigma \in \Sigma_{i,uc}$ (line 2). Two different cases are treated. The first one (lines 4-8) considers the case where the transition with event $\sigma \in \Sigma_{i,uc}$ causes a state change. In such a case, the original transition $(q, \sigma, q') \in \Delta_A$ is removed (line 5) and two new transitions are created, $(q, \sigma^{-N}, q_\sigma)$ and $(q_\sigma, \sigma^{+N}, q')$ (line 6). The algorithm also checks for the existence of transitions labeled with events $\lambda \in \Sigma_{i,uc}$ originating in state q' (line 7). If this is the case, then the attacker should be able to also erase event λ at state q_σ, or otherwise, its presence could be revealed. For this reason, a transition $(q_\sigma, \lambda^{-N}, q_\sigma)$ is added (line 8). Note that the transition added is a self-loop. This is because while the

Algorithm 6: Automaton representation of f_A^D

Input: $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$
Result: $G_i^D = (Q_A, \Sigma_{ai}, \delta_A, q_0, Q_m)$

1 $G_i^D \leftarrow G_i$
2 $Q_A \leftarrow Q_A \cup \{q_\sigma\}, \forall \sigma \in \Sigma_{i,uc}$
3 **foreach** $(q, \sigma, q') \in \Delta_A$, *with* $\sigma \in \Sigma_{i,uc}$ **do**
4 **if** $q \neq q'$ **then**
5 $\Delta_A \leftarrow \Delta_A \setminus \{(q, \sigma, q')\}$
6 $\Delta_A \leftarrow \Delta_A \cup \{(q, \sigma^{-N}, q_\sigma), (q_\sigma, \sigma^{+N}, q')\}$
7 **if** $\exists (q', \lambda, q'') \in \Delta_A$, *with* $\lambda \in \Sigma_{i,uc}$ *and* $q'' \in Q_A$ **then**
8 $\Delta_A \leftarrow \Delta_A \cup \{(q_\sigma, \lambda^{-N}, q_\sigma)\}$
9 **else**
10 $\Delta_A \leftarrow \Delta_A \setminus \{(q, \sigma, q')\}$
11 $\Delta_A \leftarrow \Delta_A \cup \{(q, \sigma^{+N}, q), (q, \sigma^{-N}, q_\sigma), (q_\sigma, \sigma^{-N}, q_\sigma), (q_\sigma, \sigma^{+N}, q)\}$
12 **if** $\exists (q, \lambda, q'') \in \Delta_A$, *with* $\lambda \in \Sigma_{i,uc}$ *and* $q'' \in Q_A$ **then**
13 $\Delta_A \leftarrow \Delta_A \cup \{(q_\sigma, \lambda^{-N}, q_\lambda), (q_\lambda, \sigma^{+N}, q_\lambda)\}$

attacker has not inserted event σ, its only option is to erase event λ if it wants to remain hidden.

The second case (lines 9-13) takes into account the situation where the event $\sigma \in \Sigma_{i,uc}$ is in self-loop. When this happens, the self-loop is removed (line 10) and four new transitions are added: $(q, \sigma^{+N}, q), (q, \sigma^{-N}, q_\sigma)$, $(q_\sigma, \sigma^{-N}, q_\sigma)$ and $(q_\sigma, \sigma^{+N}, q)$. These transitions allow the attacker to erase event σ any number of times and to insert in the network any number of occurrences of event σ. The decision about each action to take first is up to the attacker. Since this is the delay mode, it is expected that the attacker erases events before inserting them. After that, it is checked if there is another transition labeled with an event $\lambda \in \Sigma_{i,uc}$ originating in state q (line 12). If this is the case, then two new transitions are added: $(q_\sigma, \lambda^{-N}, q_\lambda)$ and $(q_\lambda, \sigma^{+N}, q_\lambda)$ (line 13). These last transitions allow the attacker to erase an event λ that is also feasible at the current state of G_i while σ is not inserted in the network.

Forward mode: An automaton G_i^F that represents the forward mode f_A^F over subsystem G_i is obtained by applying Algorithm 7.

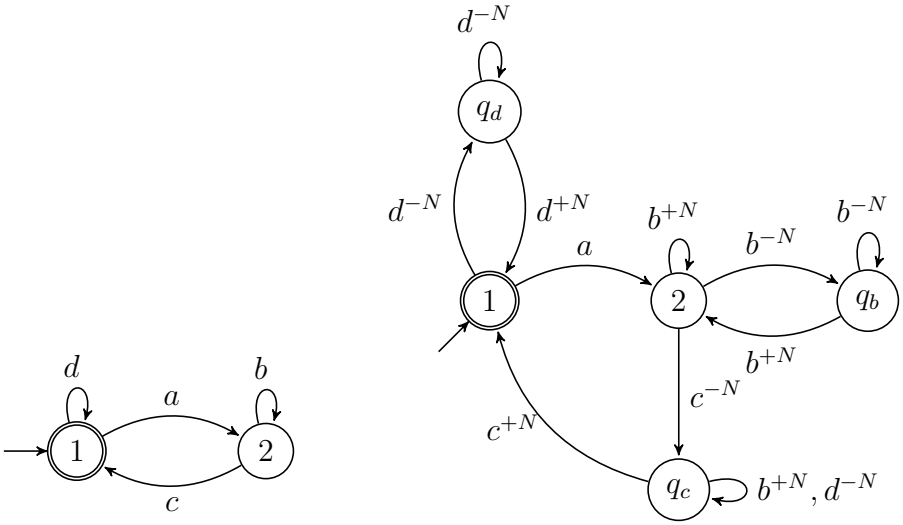
In Alg. 7, lines 2-4 are executed once for each event in $\sigma \in \Sigma_{i,uc}$. First, the algorithm checks if σ is in self-loop in G_i (line 3). If this is the case, then a new self-loop with event σ^{+N} is added (line 4).

Next it is presented two examples of how to obtain the automata G_i^D and G_i^F, representing the delay mode and forward mode, respectively, acting over subsystem G_i.

Example 16. *Consider the subsystem G_i shown in Fig. 5.7(a), where $\Sigma_c = \{a\}$ and $\Sigma_{uc} = \{b, c, d\}$. If Algorithm 6 is applied over G_i, the result is the automaton of Fig. 5.7(b).*

Algorithm 7: Automaton representation of f_A^F

Input: $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$
Result: $G_i^F = (Q_A, \Sigma_{ai}, \delta_A, q_0, X_A^m)$
1 $G_i^F \leftarrow G_i$
2 **foreach** $(q, \sigma, q') \in \Delta_A$, *with* $\sigma \in \Sigma_{i,uc}$ **do**
3 **if** $q = q'$ **then**
4 $\Delta_A \leftarrow \Delta_A \cup \{(q, \sigma^{+N}, q)\}$



(a) Automaton G_i (b) G_i^D

Figure 5.7: Automata of Example 16.

It is important to highlight that G_i has transitions labeled with uncontrollable events causing a state change as well as self-loops. In the delay mode the main goal of the attacker is to erase uncontrollable events as soon as they happen and then insert false occurrences of them in the network, which results in a delay. The first step is to set $G_i^D = G_i$. Then all transitions labeled with events in $\Sigma_{i,uc}$ are removed, which means that the attacker will always erase the uncontrollable events generated by G_i. The next step is, for every $\sigma \in \Sigma_{i,uc}$, to create a state q_σ and add it to Q_A. Then each event σ is treated individually.

For event d, that is in self-loop, transitions are added so that they allow the attacker to erase multiple occurrences of event d and then insert false occurrences of it in the network. The number of insertions does not need to match the number of occurrences that were erased, since the event is in self-loop. The same analysis can be done for event b, that is also in self-loop at state 2. However, because there is another uncontrollable event that is also feasible at state 2 of G_i, additional transitions are added. To explain the reasoning behind the necessity of such extra transitions, imagine the situation where event b occurred and was

erased and the attacker wants to insert it again into the network before the occurrence of event c. As event c is uncontrollable, it can occur before the attacker has inserted event b and the attacker should be able to erase it. That is why a transition from state q_b to q_c with event c^{-N} was added in G_i^D. But once event c is erased, the attacker should still be able to insert event b in the network, which justifies the self-loop with b^{+N} at state q_c. Note that event b can be inserted in the network only while event c is not inserted, or else the attacker is revealed.

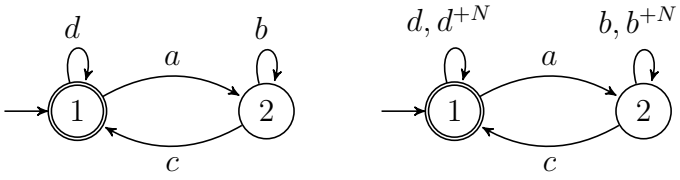
Regarding event c, since that is an event that causes a state change, the attacker can only erase and insert it back to the network once for every time the legitimate event c occurs. Moreover, as long as the attacker has not yet inserted back event c, event d can occur since in this case G_i is already at state 1. If event d is not erased at this point, the attacker is also revealed, because as the devices in the network have not observed event c, they think that G_i is still at state 2, where event d is not feasible. For this reason, a self-loop with event d^{-N} is added at state q_c.

Finally, notice that one does not need to worry about event a, which is controllable, happening while G_i is at a state where it is not feasible. This is because it is assumed that the controller will only trigger a controllable event when it is feasible at the controller's current state estimate. □

The next example shows how to obtain an automaton representing the forward mode.

Example 17. *Consider the subsystem G_i shown in Fig. 5.8(a), where $\Sigma_c = \{a\}$ and $\Sigma_{uc} = \{b, c, d\}$. If Algorithm 7 is applied over G_i, the result is the automaton of Fig. 5.8(b).*

In this example, the goal of the attacker is to insert false occurrences of events before the real ones happen in G_i, while remaining undetected. This is only possible for uncontrollable events that are in self-loop. For the other events, the only option for the attacker is to let them go untouched when they happen.



(a) Automaton G_i (b) G_i^F

Figure 5.8: Automata of Example 17.

□

As stated by Theorem 4, an attacker that uses one of the attack functions of Def. 12, is a persistent attacker. However, there are other attack functions that can be employed by

an attacker and that ensure that it will be a persistent attacker. However, a simple parallel composition of the automata generated by Alg. 6 and 7 does not generate an attack function that can be used by a persistent attacker. The next section shows how to obtain a combination of the attack functions of Def. 12.

5.2 Design method for a persistent attacker

In order to obtain the combination of all three attack functions of Def. 12, two automata are built, called estimators. The estimators proposed here were inspired by the estimator proposed by Zhang et al. (2021). The estimators are automata that help the attacker to know which actions the attacker can take and what is the impact of the actions on the system. The first one is the *plant estimator*, which shows what the attacker knows the subsystem under attack is seeing. The second one, called the *network estimator*, represents what the attacker knows about the subsystem under attack as it is seen by the network and the devices connected to it. Particularly, the network estimator captures the situations that can reveal the attacker's presence. Then, these two estimators are combined and after some refinement, the automaton that represents the combination of the attack functions is obtained. The main result of this section establishes that the resulting automaton includes the behavior of all three attack functions of Def 14. The proof of this result relies on the fact that the behavior under attack is included in the plant and network estimators and is preserved after their combination, independently of which attack function of Def. 14 is being used by the attacker. Lemmas 4 and 5 show that the behavior under attack is captured by the estimators.

5.2.1 Plant estimator

In this subsection an algorithm for building the plant estimator is provided, which is a structure that the attacker can use to determine the current state estimate according to the observation of the subsystem under attack. Events in $\Sigma_{i,uc}^{+N}$ will not affect the plant since they are not seen by it. This is the reason they appear in self-loops in all states of the estimator. Events in $\Sigma_{i,uc}^{-N}$ are only possible when the legitimate event is possible and because they hide the real occurrence of an event, this action of the attacker makes the estimator reach a new state, that represents that the system is under attack. The plant estimator is obtained by applying Alg. 8.

The next examples show how to obtain the plant estimator from G_i by applying Alg. 8.

Example 18. *Consider the automaton of Fig. 5.9(a) where* $\Sigma_c = \{a, b\}$ *and* $\Sigma_{uc} = \{c\}$. *In order to obtain* $\Theta_{G_i}^{G}$ *according to Alg. 8, the first step is to consider* $\Theta_{G_i}^{G} = G_i$ *(line 2).*

Algorithm 8: Plant estimator $\Theta^G_{G_i}$

Input: $G_i = (Q, \Sigma_i, \delta_i, q_0, Q_m)$
Result: $\Theta^G_{G_i} = (X_G, \Sigma_{ai}, \delta_G, x_0, X^m_G)$

```
1  Σ_ai ← Σ_i ∪ Σ^{+N}_{i,uc} ∪ Σ^{-N}_{i,uc}
2  Θ^G_{G_i} ← G_i
3  foreach σ^{-N} ∈ Σ^{-N}_{i,uc} do
4  |  foreach x ∈ X_G do
5  |  |  if (x, σ, x) ∈ Δ_G then
6  |  |  |  X_G ← X_G ∪ {x_σ}
7  |  |  |  Δ_G ← Δ_G ∪ {(x, σ^{-N}, x_σ)}
8  |  |  |  foreach (x, λ, x'') ∈ Δ_G do
9  |  |  |  |  Δ_G ← Δ_G ∪ {(x_σ, λ, x'')}
10 |  |  else if (x, σ, x') ∈ Δ_G then
11 |  |  |  Δ_G ← Δ_G ∪ {(x, σ^{-N}, x')}
12 foreach σ^{+N} ∈ Σ^{+N}_{i,uc} do
13 |  foreach x ∈ X_G do
14 |  |  if (δ_G(x, σ) = x) ∨ (δ_G(x, σ)! ∧ x ∉ Q) ∨ (δ_G(x, σ) not defined) then
15 |  |  |  Δ_G ← Δ_G ∪ {(x, σ^{+N}, x)}
```

Then, the foreach loop of lines 3-11 considers every event $\sigma^{-N} \in \Sigma^{-N}_{i,uc}$. Since the only event in $\Sigma_{i,uc}$ is event c, then $\Sigma^{-N}_{i,uc} = \{c^{-N}\}$. The inner foreach loop (lines 4-11) considers each state $x \in X_G$ and checks if a transition with the legitimate event σ, which in this case is event c, is defined. As event c is in self-loop, then lines 5-9 are executed with state $x = 2$. First, a new state x_c is created and added to X_G (line 6), as well as a new transition (x, c^{-N}, x_c) (line 7). The estimator obtained up to this point is shown in Fig. 5.9(b).

Next, the foreach of lines 8 and 9 will add one transition originating in state x_c for each transition that originates at state 2, which is the state that a transition with the legitimate event c is originally defined. Since state 2 is the origin of three transitions, it means that three new transitions are going to be added originating in state x_c. The new transitions have the state x_c as origin and the same destination as the original ones. After adding the new transitions, the estimator obtained so far is shown in Fig. 5.9(c).

Finally, in the foreach loop of lines 12-15, self-loops with events in $\Sigma^{+N}_{i,uc}$ are added to the estimator. Note that $\Sigma^{+N}_{i,uc} = \{c^{+N}\}$ since c is the only uncontrollable event in G_i. The self-loops are added to the states of the estimator if at least one of the three conditions of line 14 is met. In this example, the second condition is met for state 2 and the last one for states 1 and x_c. The final automaton is shown in Fig. 5.9(d).

□

In the plant estimator, the intent of adding self-loops with events in $\Sigma^{+N}_{i,uc}$ is to not restrict the ability of the attacker to insert events, except in the case where the event is inserted before

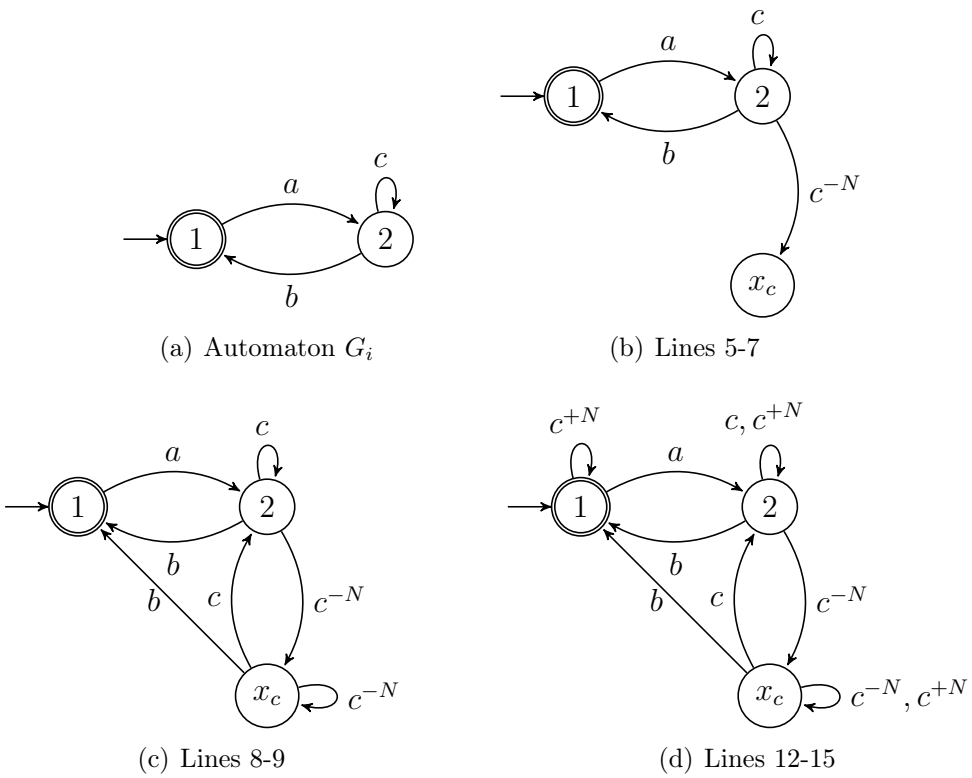


Figure 5.9: Automata of Example 18. Construction of $\Theta^G_{G_i}$ by Alg. 8.

the legitimate event that causes a state change is erased. This situation will be represented in the next example.

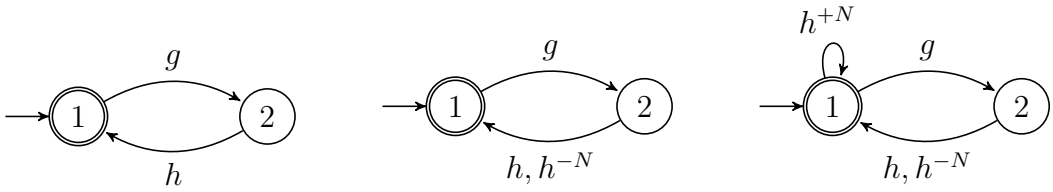
Example 19. *Consider the automaton of Fig. 5.10(a) where $\Sigma_c = \{g\}$ and $\Sigma_{uc} = \{h\}$. In order to obtain $\Theta^G_{G_i}$ according to Alg. 8, the first step is to consider $\Theta^G_{G_i} = G_i$ (line 2).*

Then, since $\Sigma_{i,uc} = \{h\}$, the foreach loop of lines 3-11 will be executed with $\Sigma^{-N}_{i,uc} = \{h^{-N}\}$. Now, the corresponding legitimate event causes a state change from state 2 to 1, which will be treated by lines 10 and 11. In this case, only a new transition from state 2 to state 1 labeled with event h^{-N} will be added to the estimator. The resulting automaton up to this point is shown in Fig. 5.10(b).

Next the self-loops with events in $\Sigma^{+N}_{i,uc} = \{h^{+N}\}$ are added. In this example, the last condition of line 14 is met for state 1 and none of the conditions is met for state 2. Thus, the self-loop with event h^{+N} is added only at state 1. The final estimator is shown in Fig. 5.10(c).

□

In Example 19, because the uncontrollable event h causes a state change, the self-loop with event h^{+N} was only added to state 1 in order to prevent the attacker from inserting the event at state 2 before the legitimate one is erased, which would eventually cause the attacker



(a) Automaton G_i (b) Lines 3-11 (c) Lines 12-15

Figure 5.10: Automata of Example 19. Construction of $\Theta^G_{G_i}$ by Alg. 8.

to be revealed. The next result, Lemma 4, shows that the behavior under attack is captured by the plant estimator.

Lemma 4. *Let G_i be a subsystem and let $\Theta^G_{G_i}$ be the plant estimator obtained by Alg. 8. Also, consider that an attacker A is acting over G_i with one of the attack functions defined in Def. 12. Then, it holds that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i})$.* ♦

Proof. To prove that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i})$ for the attack functions of Def. 12, one has to consider every string $w \in \mathcal{L}(A/G_i)$ and show that $w \in \mathcal{L}(\Theta^G_{G_i})$, for each of the three attack functions. This will be done inductively on the length of w.

Base case: According to Def. 13,

$$w = \varepsilon \in \mathcal{L}(A/G_i).$$

It is also true that

$$\varepsilon \in \mathcal{L}(\Theta^G_{G_i}), \tag{5.19}$$

which is valid for all three attack functions.

Induction step: As the induction hypothesis, assume that for a string $w \in \mathcal{L}(A/G_i)$, such that $|w| = k$, it holds that $w \in \mathcal{L}(\Theta^G_{G_i})$. Also, according to Def. 13ii, a new string $w' = wv^a \in \mathcal{L}(A/G_i)$, of length $k+1$, is obtained by concatenating w with a string $v^a \in f_A(w, s)$, with $s \in \Sigma^1_i \cup \{\varepsilon\}$ and $P^G(w)s \in \mathcal{L}(G_i)$. Also, consider a state $x \in X_G$, such that $x = \delta_G(x_0, w)$.

For the passive mode, according to (5.8) in Def 12:

$$\begin{aligned} v^a &\in f^P_A(w, s) = \{s\} \\ v^a &= s. \end{aligned} \tag{5.20}$$

Thus, $w' = wv^a = ws \in \mathcal{L}(A/G_i)$ and since, according to (5.8), the string w is only composed of events in Σ_i, one can say that

$$P^G(w) = w \tag{5.21}$$

If $P^G(w)s \in \mathcal{L}(G_i)$, then, from (5.21),

$$w' \in \mathcal{L}(G_i) \tag{5.22}$$

Because Alg. 8 starts by considering that $\Theta^G_{G_i} = G_i$ (line 2) and no transitions are removed from $\Theta^G_{G_i}$, then one can say that

$$\mathcal{L}(G_i) \subseteq \mathcal{L}(\Theta^G_{G_i}). \tag{5.23}$$

From (5.22) and (5.23), one can say that $w' \in \mathcal{L}(\Theta^G_{G_i})$, for the passive mode.

Regarding the delay mode, according to (5.9), $v^a \in f^D_A(w,s)$, with $P^G(w)s \in \mathcal{L}(G_i)$, there are three possibilities:

a) $f^D_A(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_{i,c}$. Thus $v^a = \sigma$ and since $P^G(w)s \in \mathcal{L}(G_i)$, then $w' = w\sigma \in \mathcal{L}(\Theta^G_{G_i})$, because Alg. 8 starts by considering that $\Theta^G_{G_i} = G_i$ and no transitions are removed from $\Theta^G_{G_i}$;

b) $f^D_A(w,s) = \{\sigma^{-N}\}$, if $s = \sigma \in \Sigma^1_{i,uc}$. Then $v^a = \sigma^{-N}$ and Alg. 8 adds a transition with event σ^{-N} whenever $\sigma \in \Gamma(\delta_G(x_0, w))$ (lines 3-11). Thus it is possible to say that $w' = w\sigma^{-N} \in \mathcal{L}(\Theta^G_{G_i})$.

c) $f^D_A(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and λ is an event such that (5.10) and (5.11) (of Def. 12) are met. In (5.10), λ is an event such that $\lambda \in \Gamma(\delta_i(q_0, P^N(w)))$, which means that λ is feasible at the current state estimate. Additionally, λ needs to be in self-loop at the current state estimate or, if it is not, then a previous occurrence must have been erased, conditions expressed by (5.11). In lines 12-15, Alg. 8 adds a transition with event λ^{+N} at state x if λ is in self-loop at state x, or if a transition with event λ is defined from a state that is reached after the erasure of λ or if no transition with event λ is defined at state x. Thus one can say that if $w' = wv^a = w\lambda^{+N} \in \mathcal{L}(A/G_i)$, then $wv^a \in \mathcal{L}(\Theta^G_{G_i})$.

Hence, it is possible to conclude that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i})$ for the delay mode.

Finally, for the forward mode, according to (5.12), $v^a \in f^F_A(w,s)$, with $P^G(w)s \in \mathcal{L}(G_i)$, there are two possibilities:

a) $f^F_A(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_i$. Hence $v^a = \sigma$ and since $P^G(w)s \in \mathcal{L}(G_i)$, then $w' = w\sigma \in \mathcal{L}(\Theta^G_{G_i})$, because Alg. 8 starts by considering that $\Theta^G_{G_i} = G_i$ and no transitions are removed from $\Theta^G_{G_i}$;

b) $f^F_A(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and λ is an event such that $q = \delta_i(q, P^N(\lambda^{+N}))$, a condition expressed by (5.13). Because Alg. 8 adds a transition

with event λ^{+N} at state x whenever λ is in self-loop at state x (lines 12-15), it is possible to say that $w' = wv^a \in \mathcal{L}(\Theta^G_{G_i})$.

Hence, one can conclude that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i})$ for the forward mode. Consequently, it is possible to say that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i})$ for any of the attack functions of Def. 12.

□

5.2.2 Network Estimator

The network estimator shows what the attacker knows about the plant if it is seen from the point of view of a device in the network (except the one that is infected). Events in $\Sigma^{-N}_{i,uc}$ will not affect this estimator since the legitimate event was erased by the attacker and therefore was not seen by the network. Events in $\Sigma^{+N}_{i,uc}$ will potentially cause transitions in the supervisor and because of that, the attacker needs to keep track of this information. This is done by making the estimator reach a new state if the legitimate event is in self-loop. If it is not in self-loop, then an additional transition will represent this information. Moreover, if the attacker inserts an event that is not supposed to happen at a given state, it will make the estimator reach a new state x_E. This state represents that the presence of the attacker was exposed. Note that it is assumed that the supervisor will never trigger a controllable event if it is not feasible at the current supervisor's state estimate. The network estimator is obtained by applying Alg. 9.

Next, two examples are presented, showing how to obtain the network estimator, by applying Alg.9.

Example 20. *Consider the automaton G_i of Fig. 5.11(a) where $\Sigma_c = \{a, b\}$ and $\Sigma_{uc} = \{c\}$. In order to obtain the network estimator $\Theta^N_{G_i}$, one has to apply Alg. 9. The first step is to consider $\Theta^N_{G_i} = G_i$ (line 2).*

Then, the foreach loop of lines 3-11 considers every event $\sigma^{+N} \in \Sigma^{+N}_{i,uc}$. Since the only event in $\Sigma_{i,uc}$ is event c, then $\Sigma^{+N}_{i,uc} = \{c\}$. The inner foreach loop (lines 4-11) considers each state $x \in X_N$ and checks if a transition with the legitimate event σ, which in this case is event c, is defined. As event c is in self-loop, then lines 5-9 are executed with state $x = 2$. First, a new state x_c is created (line 6), as well as a new transition (x, c^{+N}, x_c) (line 7). The estimator obtained up to this point is shown in Fig. 5.11(b).

Next, the foreach of lines 8 and 9 will add one transition originating in state x_c for each transition that originated from state 2, which is the state that a transition with the legitimate event c is originally defined. Since state 2 is the origin of three transitions, it means that three new transitions are going to be added originating from state x_c. The new transitions have

Algorithm 9: Network estimator Θ_N

Input: $G = (Q, \Sigma_i, \delta_i, x_0, X_m)$
Result: $\Theta^N_{G_i} = (X_N, \Sigma_{ai}, \delta_N, x_0, X^m_N)$

```
1  Σ_ai ← Σ_i ∪ Σ^{+N}_{i,uc} ∪ Σ^{−N}_{i,uc}
2  Θ^N_{G_i} ← G_i
3  foreach σ^{+N} ∈ Σ^{+N}_{i,uc} do
4  |  foreach x ∈ X_N do
5  |  |  if (x, σ, x) ∈ δ_N then
6  |  |  |  X_N ← X_N ∪ {x_σ}
7  |  |  |  Δ_N ← Δ_N ∪ {(x, σ^{+N}, x_σ)}
8  |  |  |  foreach (x, λ, x'') ∈ Δ_N do
9  |  |  |  |  Δ_N ← Δ_N ∪ {(x_σ, λ, x'')}
10 |  |  else if (x, σ, x') ∈ δ_N then
11 |  |  |  Δ_N ← Δ_N(x, σ^{+N}, x')}
12 X_N ← X_N ∪ {x_E}
13 foreach x ∈ X_N do
14 |  foreach σ ∈ Σ_{i,uc} do
15 |  |  if δ_N(x, σ) is not defined then
16 |  |  |  Δ_N ← Δ_N ∪ {(x, σ, x_E)}
17 |  |  |  Δ_N ← Δ_N ∪ {(x, σ^{+N}, x_E)}
18 foreach σ^{−N} ∈ Σ^{−N}_{i,uc} do
19 |  foreach x ∈ X_N do
20 |  |  Δ_N ← Δ_N ∪ {(x, σ^{−N}, x)}
```

the state x_c as origin and the same destination as the original ones. After adding the new transitions, the estimator obtained so far is shown in Fig. 5.11(c).

Then, the algorithm adds a new state x_E to X_N (line 12) and proceeds to add new transitions connecting this new state in the foreach loop of lines 13-17. Thus, for each state $x \in X_G$ and each event $\sigma \in \Sigma_{i,uc}$, it checks if $\delta_N(x, \sigma)$ is defined. If the transition is not defined, then two new transitions are added (lines 16 and 17): (x, σ, x_E) and (x, σ^{+N}, x_E). Note that state x_E is reached when an uncontrollable event that was not originally defined in a given state x happens. Also notice that self-loops with events c and c^{+N} are also added to state x_E, since event c is not originally defined in it. The estimator obtained so far is shown in Fig.5.11(d)

Finally, in the foreach loop of lines 18-20, self-loops with events in $\Sigma^{-N}_{i,uc}$ are added to the estimator. Note that $\Sigma^{+N}_{i,uc} = \{c^{-N}\}$ since c is the only uncontrollable event in G_i. The final automaton is shown in Fig. 5.9(d).

□

In contrast to Example 20, the next example shows how to build the network estimator of an automaton G_i that has a transition labeled with an uncontrollable that causes a state change.

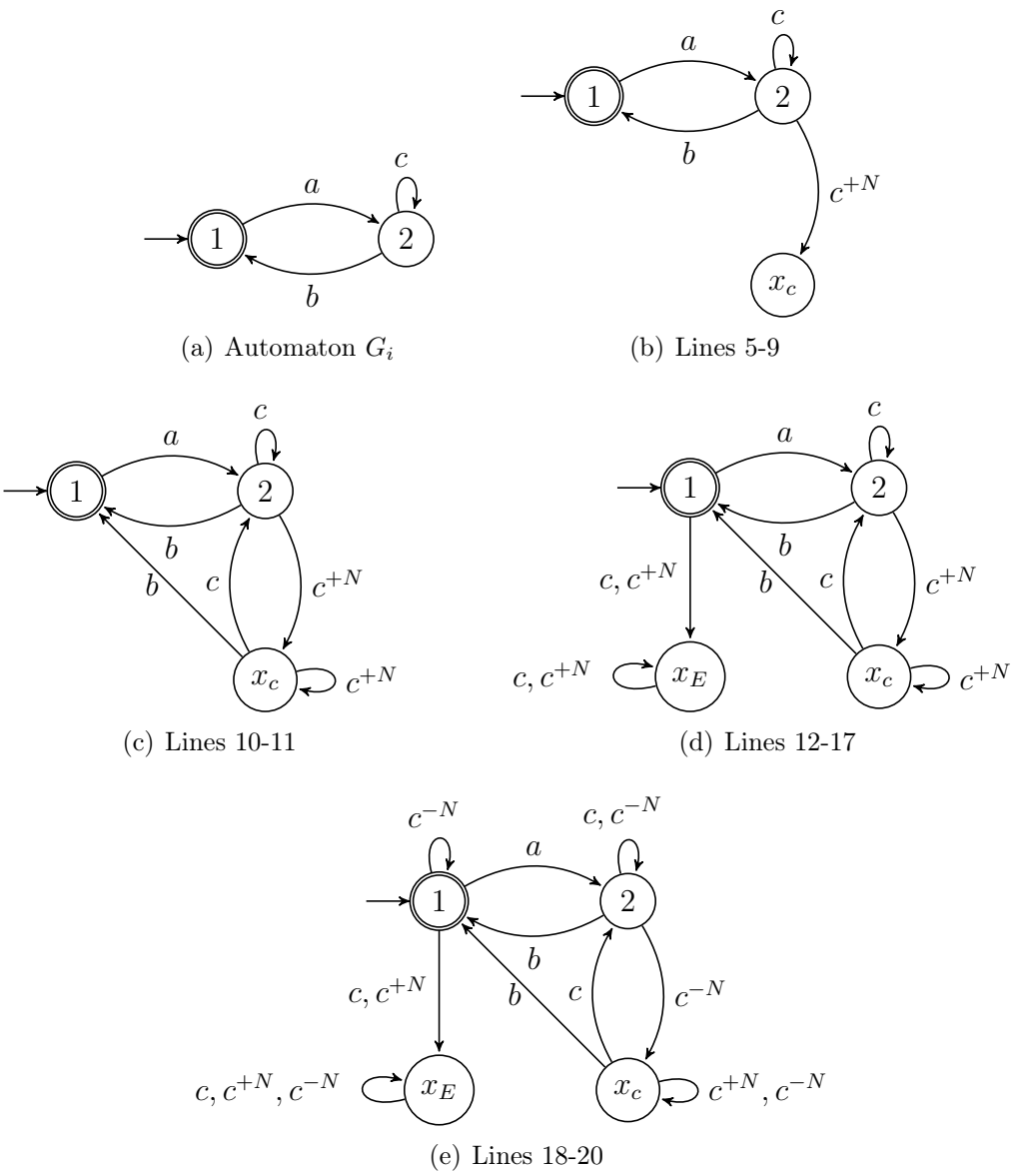


Figure 5.11: Automata of Example 20. Construction of $\Theta^N_{G_i}$ by Alg. 9.

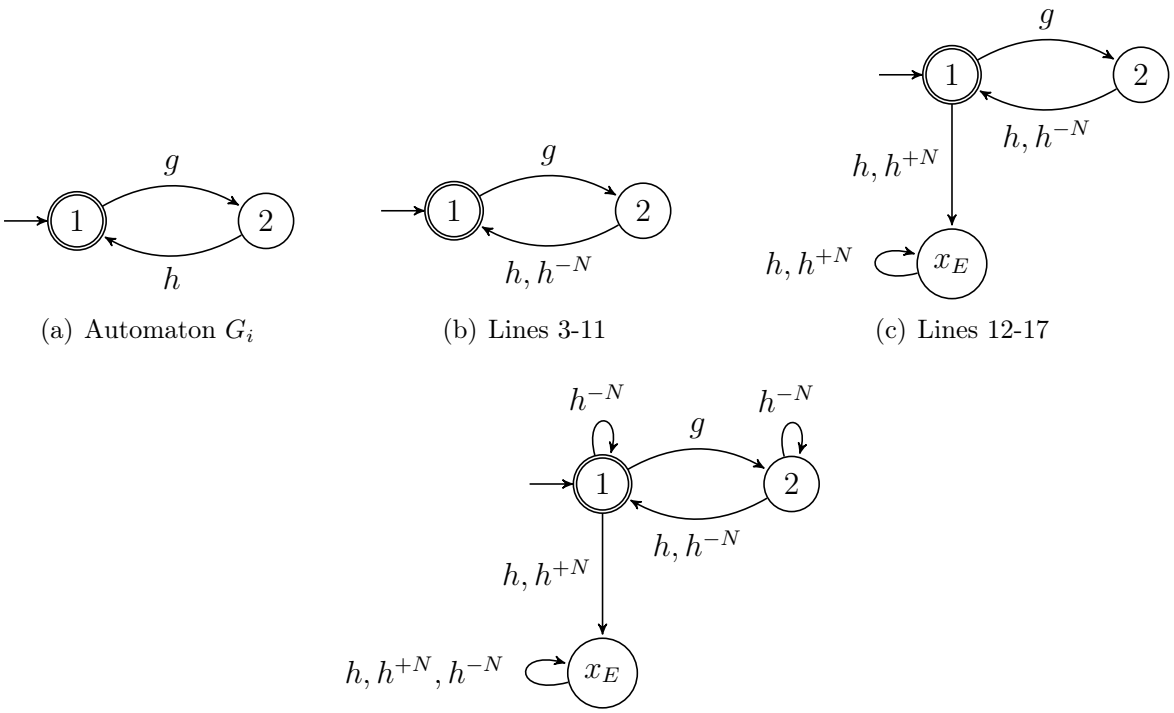
Example 21. *Consider the automaton of Fig. 5.12(a) where* $\Sigma_c = \{g\}$ *and* $\Sigma_{uc} = \{h\}$*. In order to obtain* $\Theta^N_{G_i}$ *according to Alg. 9, the first step is to consider* $\Theta^N_{G_i} = G_i$ *(line 2).*

Then, since $\Sigma_{i,uc} = \{h\}$*, the foreach loop of lines 3-11 will be executed considering* $\Sigma^{-N}_{i,uc} = \{h^{-N}\}$*. Now, the corresponding legitimate event causes a state change from state 2 to 1, which will be treated by lines 10 and 11. In this case, only a new transition from state 2 to state 1 labeled with event* h^{-N} *will be added to the estimator. The resulting automaton up to this point is shown in Fig. 5.12(b).*

The next step is to add a new state x_E *to* x_N *(line 12) and then add the transitions to connect it to the automaton. This is done in the foreach loop of lines 13-17. The resulting automaton*

is shown in Fig. 5.12(c).

Finally, in the foreach loop of lines 18-20, self-loops with events in $\Sigma_{i,uc}^{-N}$ *are added to the estimator. Note that* $\Sigma_{i,uc}^{+N} = \{h^{-N}\}$ *since* h *is the only uncontrollable event in* G_i*. The final automaton is shown in Fig. 5.12(d).*



(a) Automaton G_i (b) Lines 3-11 (c) Lines 12-17

(d) Lines 18-20

Figure 5.12: Automata of Example 21. Construction of $\Theta_{G_i}^N$ by Alg. 9.

□

The next lemma shows that the behavior under attack is captured by the network estimator.

Lemma 5. *Let* G_i *be a subsystem and let* $\Theta_{G_i}^N$ *be the network estimator obtained by Alg.9. Also, consider that an attacker* A *is acting over* G_i *with one of the attack functions defined in Def. 12. Then, it holds that* $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta_{G_i}^N)$. ♦

Proof. To prove that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta_{G_i}^N)$ for the attack functions of Def. 12, one has to consider every string $w \in \mathcal{L}(A/G_i)$ and show that $w \in \mathcal{L}(\Theta_{G_i}^N)$, for each of the three attack functions. This will be done inductively on the length of w.

Base case: According to Def. 13,

$$w = \varepsilon \in \mathcal{L}(A/G_i).$$

It is also true that

$$\varepsilon \in \mathcal{L}(\Theta^N_{G_i}), \tag{5.24}$$

which is valid for all three attack functions.

Induction step: As the induction hypothesis, assume that for a string $w \in \mathcal{L}(A/G_i)$, such that $|w| = k$, it holds that $w \in \mathcal{L}(\Theta^N_{G_i})$. Also, according to Def. 13ii, a new string $w' = wv^a \in \mathcal{L}(A/G_i)$, of length $k+1$, is obtained by concatenating w with a string $v^a \in f_A(w, s)$, with $s \in \Sigma^1_i \cup \{\varepsilon\}$ and $P^G(w)s \in \mathcal{L}(G_i)$. Also, consider a state $x \in X_G$, such that $x = \delta_G(x_0, w)$.

For the passive mode, according to (5.8) in Def 12:

$$\begin{aligned} v^a &\in f^P_A(w, s) = \{s\} \\ v^a &= s. \end{aligned} \tag{5.25}$$

Thus, $w' = ws \in \mathcal{L}(A/G_i)$ and since, according to (5.8), the string w is only composed of events in Σ_i, it is possible to say that

$$P^G(w) = w \tag{5.26}$$

If $P^G(w)s \in \mathcal{L}(G_i)$, then from (5.26),

$$w' \in \mathcal{L}(G_i) \tag{5.27}$$

Because Alg. 9 starts by considering that $\Theta^N_{G_i} = G_i$ (line 2) and no transitions are removed from $\Theta^N_{G_i}$, then it is possible to say that

$$\mathcal{L}(G_i) \subseteq \mathcal{L}(\Theta^N_{G_i}). \tag{5.28}$$

From (5.27) and (5.28), one can say that $w' \in \mathcal{L}(\Theta^N_{G_i})$, for the passive mode.

Regarding the delay mode, according to (5.9), $v^a \in f^D_A(w, s)$, with $P^G(w)s \in \mathcal{L}(G_i)$, there are three possibilities:

a) $f^D_A(w, s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_{i,c}$. Thus $v^a = \sigma$ and since $P^G(w)s \in \mathcal{L}(G_i)$, then $w' = w\sigma \in \mathcal{L}(\Theta^N_{G_i})$, because Alg. 9 starts by considering that $\Theta^N_{G_i} = G_i$ and no transitions are removed from $\Theta^N_{G_i}$;

b) $f^D_A(w, s) = \{\sigma^{-N}\}$, if $s = \sigma \in \Sigma^1_{i,uc}$. Then $v^a = \sigma^{-N}$ and Alg. 9 adds a self-loop with event σ^{-N} in every state of the estimator (lines 18-20). Thus it is possible to say that $w' = w\sigma^{-N} \in \mathcal{L}(\Theta^N_{G_i})$.

c) $f^D_A(w, s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and λ is an event such that (5.10) and (5.11) are met. In (5.10), λ is an event such that $\lambda \in \Gamma(\delta_i(q_0, P^N(w)))$,

which means that λ is feasible at the current state estimate. Additionally, λ needs to be in self-loop at the current state estimate or if it is not, then a previous occurrence must have been erased, conditions expressed by (5.11). In lines 3-11, Alg. 9 adds a transition with event λ^{+N} at state x if a transition with event λ is defined $\Theta^N_{G_i}$. Thus it is possible to say that if $w' = wv^a = w\lambda^{+N} \in \mathcal{L}(A/G_i)$, then $wv^a \in \mathcal{L}(\Theta^N_{G_i})$.

Hence, one can conclude that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^N_{G_i})$ for the delay mode.

Finally, for the forward mode, according to (5.12), $v^a \in f^F_A(w,s)$, with $P^G(w)s \in \mathcal{L}(G_i)$, there are two possibilities:

a) $f^F_A(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_i$. Hence $v^a = \sigma$ and since $P^G(w)s \in \mathcal{L}(G_i)$, then $w' = w\sigma \in \mathcal{L}(\Theta^N_{G_i})$, because Alg. 9 starts by considering that $\Theta^N_{G_i} = G_i$ and no transitions are removed from $\Theta^N_{G_i}$;

b) $f^F_A(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Since $|v^a| = 1$, then $v^a = \lambda^{+N}$ and λ is an event such that $q = \delta_i(q, P^N(\lambda^{+N}))$, condition expressed by (5.13). Because Alg. 9 adds a transition with event λ^{+N} at state x whenever λ is defined at state x (lines 3-11), one can say that $w' = wv^a \in \mathcal{L}(\Theta^N_{G_i})$.

Hence, it is possible to conclude that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^N_{G_i})$ for the forward mode. Consequently, one can say that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^N_{G_i})$ for any of the attack functions of Def. 12.

□

The next subsection shows how to use the plant and network estimators to obtain a combination of the attack functions of Def. 14.

5.2.3 Non-exposing attack structure

The next step is to combine the plant and network estimators in order to obtain all possible behaviors of the system under attack. This is done by parallel composition and the resulting automaton is called the *attack structure*, $\Theta_{G_i} = \Theta^G_{G_i} || \Theta^N_{G_i}$. Depending on the actions of the attacker, states labeled with a pair of the type $(\_, x_E)$ may be reachable in the attack structure. These states represent that the attacker is revealed, since the behavior of the system under attack, as seen by the IDS, is not in the generated language of G_i. Such states are called *exposing states*, which are formalized by the next definition.

Definition 14. *[Exposing states] A state $x = (x_1, x_2) \in X_G \times X_N$ is an exposing state if $x_2 = x_E$.* ◊

The next result shows that if an exposing state is reached in Θ_{G_i} by a string w', then the same string w', when seen from the network's point of view, is not in $\mathcal{L}(G_i)$.

Lemma 6. *Let G_i be a subsystem and $\Theta_{G_i} = \Theta^G_{G_i} || \Theta^N_{G_i} = (X, \Sigma_{ai}, \delta_A, x_0, X_M)$ its corresponding attack structure. Consider a string $w \in \mathcal{L}(\Theta_{G_i})$ such that $P^N(w) \in \mathcal{L}(G_i)$. If $w' = wv^a \in \mathcal{L}(\Theta_{G_i})$, with $v^a \in \Sigma^1_{ai}$, is a string such that $\delta_A(x_0, w') = (\_, x_E)$, then $P^N(w') \notin \mathcal{L}(G_i)$.* ♦

Proof. If $w' = wv^a \in \mathcal{L}(\Theta_{G_i}) = \mathcal{L}(\Theta^G_{G_i} || \Theta^N_{G_i})$, then it means that $wv^a \in \mathcal{L}(\Theta^G_{G_i}) \cap \mathcal{L}(\Theta^N_{G_i})$, since the estimators share the same alphabet. Because the exposing state is only defined in the network estimator, then in order to analyze the case when $\delta_A(x_0, w') = (\_, x_E)$, it suffices to consider only the case when $\delta_N(x_0, w') = x_E$.

Applying the network projection over wv^a, one has

$$P^N(w') = P^N(wv^a) = P^N(w)P^N(v^a) = \begin{cases} P^N(w) & \text{if } v^a \in (\Sigma^{-N}_{i,uc})^1 \quad (5.29) \\ P^N(w)v & \text{if } v^a \in (\Sigma_i \cup \Sigma^{+N}_{i,uc})^1 \quad (5.30) \end{cases}$$

From (5.29), an exposing state is never reached, since in the construction of the network estimator in Alg. 9, lines 13-17, an exposing state is reached with events in Σ_i or in Σ^{+N}.

On the other hand, from (5.30) and from the lemma statement, it is true that $\delta_A(x_0, wv^a) = (\_, x_E)$. According to lines 13-17 of Alg. 9, the condition to add a transition with label $v^a \in (\Sigma_i \cup \Sigma^{+N}_{i,uc})^1$ to the exposing state x_E is that in the partially-built network estimator, $\delta_N(x, v)$ is not defined, with $x = \delta_N(x_0, w)$ and $v \in \Sigma^1_{i,uc}$, which in turn, means that $\delta_i(q_0, P^N(wv))$ is not defined as well, since the algorithm starts by considering that $\Theta^N_{G_i} = G_i$ (line 2) and no transitions are removed from $\Theta^N_{G_i}$.

If $\delta_i(q_0, P^N(wv))$ is not defined, then it is possible to say that $P^N(w') = P^N(wv^a) = P^N(w)v = P^N(wv) \notin \mathcal{L}(G_i)$. In other words, if $w' \in \mathcal{L}(A/G_i)$ and $\delta_{GN}(x_0, w') = (\_, x_E)$, then $P^N(w') \notin \mathcal{L}(G_i)$. □

To avoid the problem of reaching exposing states, one can remove these states from the attack structure, as well as the associated transitions. Removing a transition in the attack structure means that the attacker is choosing to not execute a given action or, in other words, it means that a given action is being disabled. From the attacker's point of view, events in $\Sigma^{+N}_{i,uc}$, $\Sigma^{-N}_{i,uc}$ and $\Sigma_{i,uc}$ are controllable, since the attacker chooses when to execute the associated actions. Note that an event in $\Sigma_{i,uc}$ also represents an action of the attacker, corresponding to the option of not tampering with the occurred event. The other option of the attacker is to erase the event from the network's observation, represented by the corresponding event in $\Sigma^{-N}_{i,uc}$. On the other hand, events in $\Sigma_{i,c}$ are uncontrollable from the attacker's point of view.

Thus, the problem of avoiding exposing states can be thought as the problem of finding a *non-exposing sublanguage* in a way that exposing states are never reached. Alg. 10 summarizes this procedure. It starts by obtaining the parallel composition of the plant and network

estimators, resulting in automaton Θ_{G_i}. Next, it removes from Θ_{G_i} the exposing states, by applying the Trim operation, and the algorithm returns an automaton Ψ_{G_i} which is called the *non-exposing attack structure*. The removal of exposing states and its associated transitions by the Trim operation is possible because once they are reached, marked states become inaccessible and, since x_E is not a marked state in $\Theta^N_{G_i}$, exposing states will not be marked states in Ψ_{G_i} either.

Algorithm 10: Non-exposing attack structure

Input: $\Theta^G_{G_i}, \Theta^N_{G_i}$
Result: Ψ_{G_i}
1 $\Theta_{G_i} \leftarrow \Theta^G_{G_i} || \Theta^N_{G_i} = ((X_G \times X_N), \Sigma_{ai}, \delta_{GN}, (x^G_0, x^N_0), (X^m_G \times X^m_N))$
2 $\Psi_{G_i} \leftarrow \text{TRIM}(\Theta_{G_i})$
3 **return** Ψ_{G_i}

The next example shows to obtain the non-exposing attack structure of the automaton G_i from Examples 18 and 20.

Example 22. *Continuing from Examples 18 and 20, the attack structure $\Theta_{G_i} = \Theta^G_{G_i} || \Theta^N_{G_i}$ is shown in Fig. 5.13, while the non-exposing attack structure Ψ_{G_i} is shown in Fig. 5.14. Note that the exposing state is reached if the attacker chooses to insert an occurrence of event c to the network, while the Θ_{G_i} is at the initial state. In order to avoid reaching such a state, the attacker should not have the possibility of inserting that event at this point. The exposing state and associated transitions are removed after the Trim operation, meaning that inserting event c at the initial state is no longer a possibility.*

□

The non-exposing attack structure represents all the behavior of the subsystem under attack that keeps the attacker undetected. Next, it is shown that the behavior of the three attack functions of a persistent attacker given by Definition 12 is captured by the non-exposing attack structure generated by Alg. 10.

Theorem 5. *Let A be an attacker with an attack function according to Def. 12 and $\mathcal{L}(\Psi_{G_i})$ be the generated language of the automaton returned by Alg. 10. Then $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Psi_{G_i})$.* ♦

Proof. One has to prove that for any string $w \in \mathcal{L}(A/G_i)$, it is also true that $w \in \mathcal{L}(\Psi_{G_i})$. This must be done for all three attack functions of Def. 12.

Because the alphabets of $\Theta^G_{G_i}$ and $\Theta^N_{G_i}$ are the same, then

$$\begin{aligned} \mathcal{L}(\Theta_{G_i}) &= \mathcal{L}(\Theta^G_{G_i} || \Theta^N_{G_i}) \\ &= \mathcal{L}(\Theta^G_{G_i}) \cap \mathcal{L}(\Theta^N_{G_i}). \end{aligned} \tag{5.31}$$

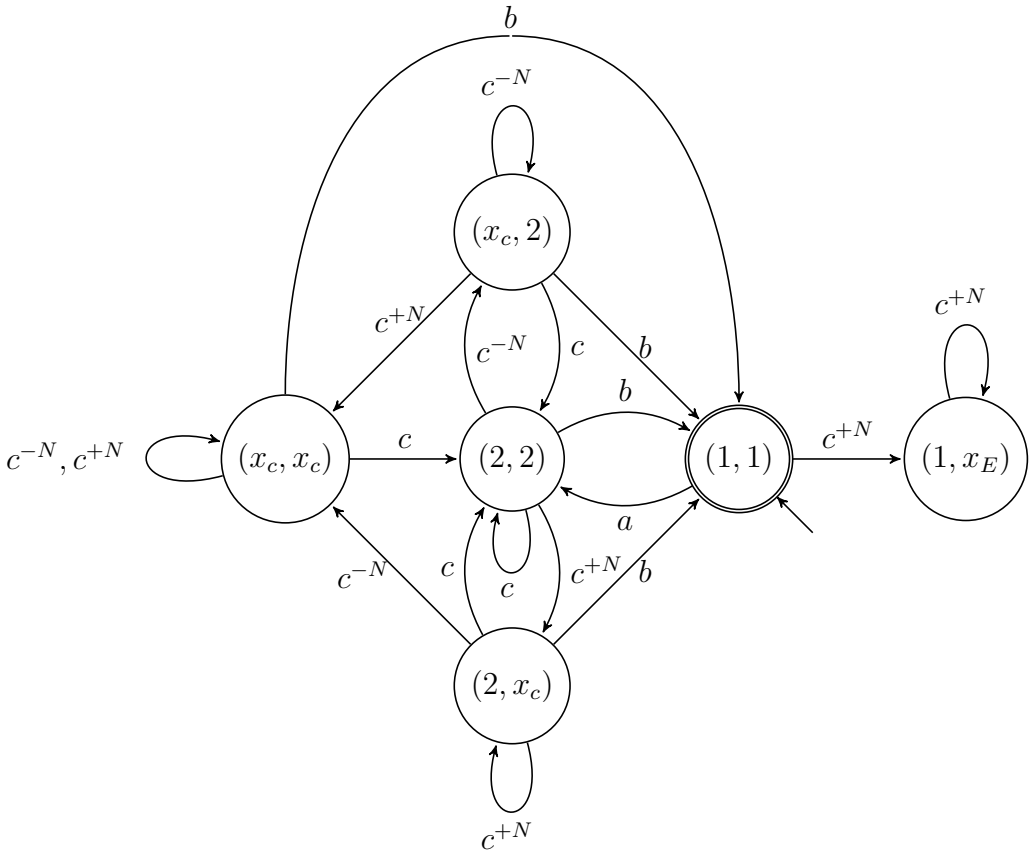


Figure 5.13: Attack structure $\Theta_{G_i} = \Theta^G_{G_i} || \Theta^N_{G_i}$ of Example 22.

According to Lemmas 4 and 5, it is possible to say that

$$\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^G_{G_i}) \tag{5.32}$$

and

$$\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta^N_{G_i}). \tag{5.33}$$

From (5.31), (5.32) and (5.33), one can say that

$$\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Theta_{G_i}). \tag{5.34}$$

Since $\mathcal{L}(\Psi_{G_i})$ is obtained from $\mathcal{L}(\Theta_{G_i})$ by removing exposing states, according to Algorithm 10, then it is true that

$$\mathcal{L}(\Psi_{G_i}) \subseteq \mathcal{L}(\Theta_{G_i}) \tag{5.35}$$

The relationship between the languages of interest is shown in Fig. 5.15. It is still to be shown that $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Psi_{G_i})$, i.e., the gray area in Fig. 5.15 does not exist. This will be done by contradiction. Assume that there exists a string $w' = wv^a \in \mathcal{L}(A/G_i)$ such that $w' \notin \mathcal{L}(\Psi_{G_i})$. If $w' \notin \mathcal{L}(\Psi_{G_i})$, and from (5.34), it is possible to say that w' is a string such that

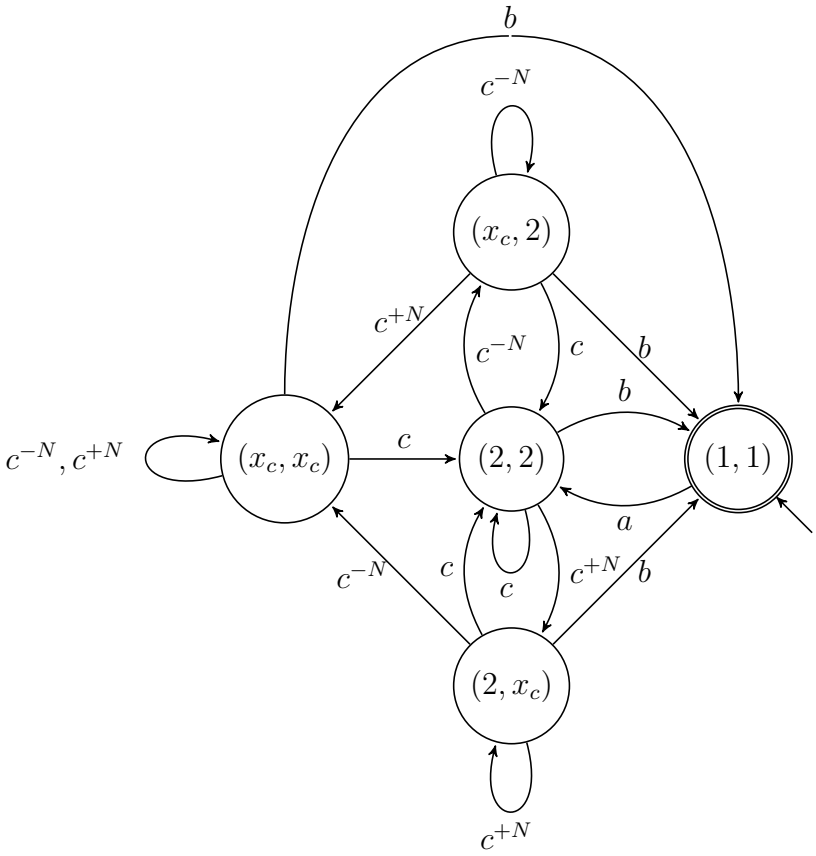


Figure 5.14: Non-exposing attack structure Ψ_{G_i} of Example 22.

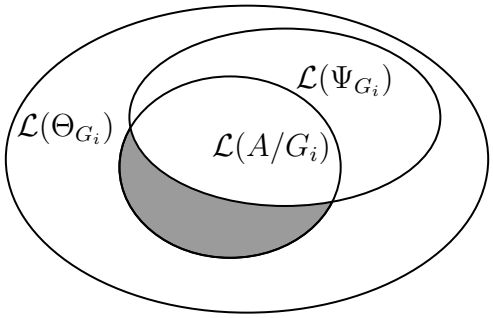


Figure 5.15: Relationship between languages.

$\delta_{GN}(x_0, wv^a) = (\_, x_E)$. From Lemma 6, it is true that

$$P^N(wv^a) \notin \mathcal{L}(G_i) \tag{5.36}$$

According to Def. 13, a string $w' = wv^a \in \mathcal{L}(A/G_i)$, with $w \in \mathcal{L}(A/G_i)$, if $v^a \in f_A(w, s)$ and $P^G(w)s \in \mathcal{L}(G_i)$.

For the passive mode of Def. 12, defined by (5.8), $v^a = s$ and one can say that $P^N(wv^a) = P^G(w)s \in \mathcal{L}(G_i)$, which contradicts (5.36).

For the delay mode, there are have three possibilities:

a) $f_A^D(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_{i,c}$. Thus, $v^a = s = \sigma$ and if $P^N(wv^a) = P^N(w)s \notin \mathcal{L}(G_i)$, which results in a contradiction, since it is assumed that a controllable event will only be triggered if it is feasible at the current state estimate;

b) $f_A^D(w,s) = \{\sigma^{-N}\}$, if $s = \sigma \in \Sigma^1_{i,uc}$. Hence $v^a = \sigma^{-N}$ and $P^N(v^a) = \varepsilon$. Then $P^N(wv^a) = P^N(w) \in \mathcal{L}(G_i)$ (Lemma 6), what contradicts (5.36);

c) $f_A^D(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Then it is possible to say that $v^a = \varepsilon$ or $v^a = \lambda^{+N}$. If $v^a = \varepsilon$, then $P^N(v^a) = \varepsilon$ and $P^N(wv^a) = P^N(w) \in \mathcal{L}(G_i)$, which contradicts (5.36). For the case where $v^a = \lambda^{+N}$, then $P^N(v^a) = \lambda$. According to (5.11), λ is an event feasible at the current state estimate, that is, $P^N(w)\lambda \in \mathcal{L}(G_i)$, which again, contradicts (5.36).

Finally, for the forward mode, there are two possibilities:

a) $f_A^F(w,s) = \{\sigma\}$, if $s = \sigma \in \Sigma^1_i$. Hence $v^a = s = \sigma$ and $P^N(v^a) = s$. Thus, $P^N(wv^a) = P^N(w)s \in \mathcal{L}(G_i)$, what contradicts (5.36).

b) $f_A^F(w,s) = \{\varepsilon, \lambda^{+N}\}$, if $s = \varepsilon$. Then it is possible to say that $v^a = \varepsilon$ or $v^a = \lambda^{+N}$. If $v^a = \varepsilon$, then $P^N(v^a) = \varepsilon$ and $P^N(wv^a) \in \mathcal{L}(G_i)$, what contradicts (5.36). For the case where $v^a = \lambda^{+N}$, then $P^N(v^a) = \lambda$. According to (5.13), λ is an event feasible at the current state estimate, that is, $P^N(w)\lambda \in \mathcal{L}(G_i)$, which again, contradicts (5.36).

Hence, if one assumes that $w' = wv^a \in \mathcal{L}(A/G_i)$ and $w' \notin \mathcal{L}(\Psi_{G_i})$, there is a contradiction, for all attack functions of Def. 12. Consequently, it is possible to say that for a string $w' \in \mathcal{L}(A/G_i)$, it is also true that $w' \in \mathcal{L}(\Psi_{G_i})$, and hence, $\mathcal{L}(A/G_i) \subseteq \mathcal{L}(\Psi_{G_i})$. □

The last result of this chapter is Theorem 6, which is the dual of Theorem 4. While Theorem 4 states that if an attacker A uses one of the attack functions of Def. 12, it will be a persistent attacker, Theorem 6 states that if an attacker uses the automaton returned by Alg. 10 as an attack function, it will also be a persistent attacker.

Theorem 6. *Let A be an attacker with an attack function represented by Ψ_{G_i}, where Ψ_{G_i} is the automaton returned by Alg. 10. Then $P^N(\mathcal{L}(\Psi_{G_i})) \subseteq \mathcal{L}(G_i)$.* ♦

Proof. One has to prove that for any string $s \in P^N(\mathcal{L}(\Psi_{G_i}))$, it is also true that $s \in \mathcal{L}(G_i)$. This will be done by contradiction.

Consider a string $w \in \mathcal{L}(\Psi_{G_i})$ such that $P^N(w) = s$, $s \notin \mathcal{L}(G_i)$. According to Lemma 6, if $P^N(w) \notin \mathcal{L}(G_i)$, it means that an exposing state is reached after the execution of w in Θ_{G_i}. Since the non-exposing attack structure, according to Alg. 10, is $\Psi_{G_i} = Trim(\Theta_{G_i})$ (line 2), then it is true that Ψ_{G_i} has no exposing states and, therefore, $w \notin \mathcal{L}(\Psi_{G_i})$, or $s \notin P^N(\mathcal{L}(\Psi_{G_i}))$,

which is a contradiction.

Thus it is possible to say that $P^N(\mathcal{L}(\Psi_{G_i})) \subseteq \mathcal{L}(G_i)$. □

Next, an example of how an attacker can use the non-exposing attack structure to guide its actions will be presented.

Example 23. *Consider the non-exposing attack structure Ψ_{G_i} obtained in Example 22, which is shown in Fig. 5.14. Upon initialization, the only option for the attacker is to wait for the occurrence of events sent by the controller. To extract from Ψ_{G_i} the possible actions of the attacker at a given moment, it suffices to identify the feasible events at the current state of Ψ_{G_i} and use the following reasoning:*

- *If an event is controllable, then the attacker can only wait for its occurrence;*
- *If an event is uncontrollable, then the attacker can let it remain untouched when it happens.*
- *If an event is in $\sigma^{-N} \in \Sigma_{i,uc}^{-N}$, then the attacker can erase event σ when it happens;*
- *If an event is in $\sigma^{+N} \in \Sigma_{i,uc}^{+N}$, then the attacker can insert it to the network at any time.*

Thus, at state $(2,2)$ in Ψ_{G_i}, for example, the attacker can choose between letting event c untouched when it happens, erasing it, or inserting it in the network at any time. Regarding event b, the only option is to let it go untouched, since it is a controllable event.

□

5.2.4 Complexity

In this subsection the time complexity of the algorithms presented in this section is discussed. In Alg. 6, the `foreach` loop of lines 3-13 is executed once for each transition in Δ_A. In order to check the `if` statement of lines 7 and 12, all transitions need to be considered, which means that the `if` statements are executed $|\Delta_A|^2$ times. In the worst case, when the transition function is complete, $|\Delta_A| = |Q_A||\Sigma_{ai}|$. Additionally, if all events in Σ_i are uncontrollable, then $|\Sigma_{ai}| = |\Sigma_i| + |\Sigma_{i,uc}^{-N}| + |\Sigma_{i,uc}^{+N}| = 3|\Sigma_i|$ and $|Q_A| = |Q| + |\Sigma_i|$. Thus $|\Delta_A| = (|Q| + |\Sigma_i|)3|\Sigma_i| = 3|Q||\Sigma_i| + 3|\Sigma_i|^2$ and $O(|\Delta_A|^2) \approx O(|\Sigma_i|^4)$, where Q and Σ_i are the set of states and events of G_i, respectively. Regarding Alg. 7, the `foreach` loop of lines 2-3 is executed once for each transition in Δ_A. Thus, the overall complexity is $O(|\Delta_A|) \approx O(|\Sigma_i|^2)$.

For Alg. 8, which builds the plant estimator, the `foreach` loop of lines 3-11 is executed once for each uncontrollable event, while the `foreach` loop of line 4-11 is executed once for every state in the estimator. Additionally, the inner `foreach` loop of lines 8-11 is executed

again for each transition in Δ_G. Thus, an upper bound for the number of executions of the `foreach` loop of lines 3-11 is $|\Delta_G|^2$. If the transition function is complete and all events are uncontrollable, then $|\Delta_G| = |X_G||\Sigma_{ai}| = (|Q| + |\Sigma_i|)(3|\Sigma_i|)$. Thus, the overall time complexity for the `foreach` loop of lines 3-11 is $O(|\Delta_G|) \approx O(|\Sigma_i|^4)$. The same time complexity can be associated with the `foreach` loop of lines 12-15. Thus, the overall time complexity of Alg. 8 is $O(|\Sigma_i|^4)$.

For the network estimator, built by Alg. 9, the analysis is similar to the one made for Alg. 8. Even though Alg. 9 has an additional `foreach` loop (lines 18-20), the overall time complexity does not change, since the `foreach` loop is executed sequentially. Thus, the overall time complexity of Alg. 9 is $O(|\Sigma_i|^4)$.

Finally, in Alg. 10, the parallel composition of line 1 has a complexity of $O(|X_G||X_N|)$, while the Trim operation over Θ_{G_i} has a linear complexity with respect to number of states of Θ_{G_i} (Cassandras and Lafortune, 2007). Since, in the worst case, $|X_G| \approx |X_N| \approx |Q| + |\Sigma_i|$, then the overall complexity of Alg. 10 is $O((|Q| + |\Sigma_i|)^2)$.

In the next section, the results of this chapter are discussed.

5.3 Discussion

First, it is important to highlight that, in the same way that a supervisor obtained by the SCT does not stipulate which event should be triggered next, but rather a set of disabled events, the non-exposing attack structure provides the possible actions that the attacker can take while remaining hidden. The decision of which action the attacker will do next needs to be made by another entity, the attacker's controller. Such an entity has knowledge about the physical process and can decide when and for how long to delay or anticipate events (or it can receive information through a communication network, indicating when to start or stop the attack). The design of the attacker's controller will be the subject of future research. Additionally, the non-exposing attack structure allows an attacker to perform different types of attacks simultaneously, i.e., the attacker can anticipate one event while it is delaying another one.

Furthermore, although the method for attack design proposed in this thesis was inspired by the method described in Zhang et al. (2021), there are thee main differences, summarized next.

- Attack location: in the referred work, the authors consider attacks in the communication channel while in this thesis the attacks are considered to happen at the devices;

- Attacker's goal: in this thesis, a new type of attacker is defined, called a persistent attacker, which is an attacker that wants to act and remain hidden, so it can act multiple times. In the aforementioned work, the attacker's goal is to lead the system to a critical state, which can be done just once, since the attacker is revealed;
- Estimators: in this thesis, the plant and network estimators are built based on the automaton of G_i, while in the referred work, one of the estimators was built based on the supervisor. Thus, depending on the size of the supervisor realization, the application of their technique can be limited by computational resources. Because the estimators proposed in this thesis are based on G_i, their size is normally smaller when compared to an estimator obtained from a supervisor.

The assumption that was made in this work that only uncontrollable events were to be attacked may seem too restrictive, but in reality, the impact of this assumption on the types of attacks proposed in this work is very small. This is because a delay or anticipation of events can be inserted in the production process by acting either on controllable or uncontrollable events. The advantage of acting only on uncontrollable events resides in the fact that the attacker is less likely to be found when compared with attacks on controllable events, as discussed in the beginning of the chapter.

Additionally, in reality the attacker has the ability to acting on all events, controllable and uncontrollable, which might seem desirable since it results in an easier problem from the point of view of the system's logical behavior, because the attacker would be free to choose when to turn on and off the machines and also would be free to choose which events to transmit to the network. However, such behavior would also draw the attention of a human operator, which is undesirable. Thus, restricting the attacker to act on only uncontrollable events is a more interesting problem.

Chapter 6

Implementation of a security testbed

In this chapter the details related to the implementation of a security testbed are discussed. The main goal of the testbed is to reduce the gap between theory and practice by providing a way to apply theoretical results in a physical system. The testbed consists of a prototype of a Networked Control System (NCS). The coordination between devices is done by the discrete-event controller, alongside continuous-time controllers. The contributions of this section were presented at the 16th IFAC Workshop on Discrete-Event System, held in Prague, between September 7-9, 2022 (Alves et al., 2022b). Additionally, an online repository was created to allow sharing all the details regarding the implementation.

The prototype was initially inspired by the work of De Oliveira et al. (2020), where the authors describe the implementation of a hybrid control system. The plant is instrumented with intelligent devices that communicate with each other in a foundation Fieldbus (FF) industrial network. A commercial PLC (programmable logic controller) is the logic solver linking the discrete elements, the HMI (human-machine interface) panel and the FF.

The process in (De Oliveira et al., 2020) consists of a pump, that supplies liquid to a tank and a pneumatic valve, that regulates the liquid level in the tank. The level sensor (LIT) in the tank and the valve positioner (LIC) are intelligent FF devices implementing a distributed PID control system. The PLC may read and write the parameters of LIT and LIC by means of an FF interface, receives the signals of a selector switch on an HMI panel, and commands the pump operation. A manual valve regulates the tank output. In such a continuous process, there are risks related to unforeseen perturbations to the PID controller, that may arise from system failures or human intervention in setpoint definition and manual valves manipulation, for example. The main risks are related to underflow and overflow of tank level. Also the pneumatic valve should not remain completely closed when the pump is turned on to avoid leakage in piping and overheating of the pump. These circumstances can occur in the steady state PID-controlled process, or in non-routine procedures, such as startup and shutdown.

As a solution, the authors design supervisors to ensure that the continuous-time controlled system can operate under safety constraints. For this purpose, they model specifications and synthesize minimally restrictive supervisors, which are reactive control agents to the events on the continuous-time process.

A similar plant exists in one of the laboratories of the Department of Electronic Engineering at UFMG. However, due to the restrictive measures adopted by the university during the COVID-19 pandemic, the lab was inaccessible. Since the main idea was to have a physical system and not a simulated one, the alternative was to build such a prototype at home. To acquire industrial FF devices was not a viable option either, due to their high cost, which would have impacted one of the prototype's goal, namely, to propose an implementation scheme that can be easily replicated. The final solution employed low-cost off-the-shelf micro-controlled devices combined with communication modules, allowing the different devices to exchange information with each other.

In order to allow other people to adapt or reproduce the prototype, a free online repository[1] was created, aiming to share information about the models, electronic circuits and code.

In the next section the adopted architecture is presented.

6.1 System architecture

In an NCS, sensor, actuator and controller devices can be considered as computational systems with communication interfaces. There are three basic configurations, which are shown in Figure 6.1. Additionally, these topologies can be combined with each other to make more complex ones. For example, suppose a given system has three variables that have to be controlled. Each control loop can have one of the topologies shown in Fig. 6.1 while sharing the same control network.

The proposed architecture is based on the topology of Figure 6.1(c) to implement the DES control while the continuous-time control loops are based on the topologies of Figures 6.1(a) and 6.1(b). Furthermore, considering the topology of Figure 6.1(c), there are three new configurations regarding the position of the controller with respect to the other nodes. In Fig. 6.2(a), the controller node is physically connected to the same control network as the actuator and sensor nodes. Additionally, a SCADA (supervisory control and data acquisition) can be employed so that a human operator can monitor and/or act over the process. The SCADA does not need to be connected directly to the control network and a gateway can make the

[1] Available at: https://lacsed.github.io/security-testbed/

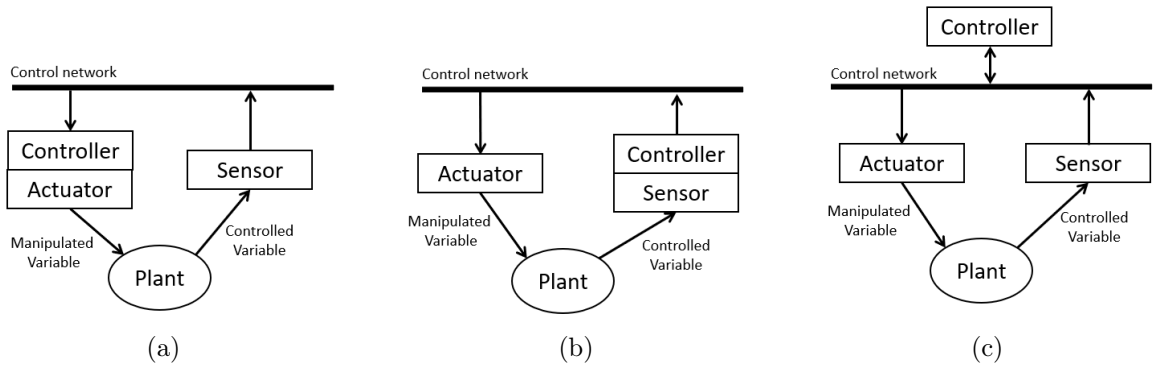


Figure 6.1: NCS topologies

bridge between them. Furthermore, in some cases, the controller also plays the role of a gateway between the control network and SCADA.

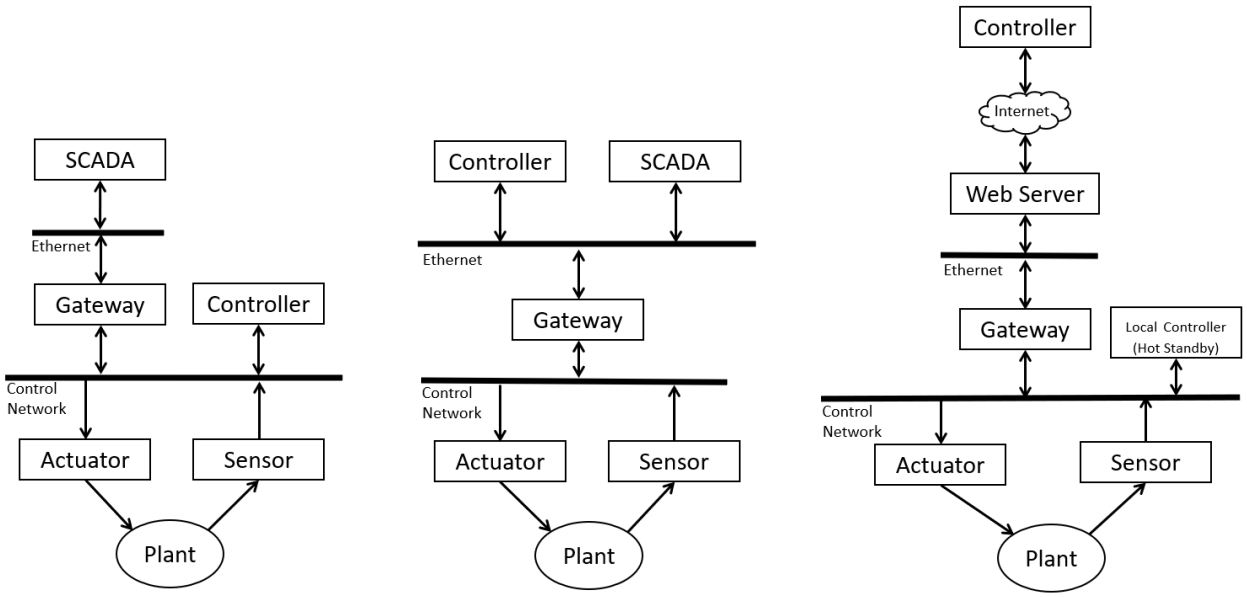
Another configuration is shown in Fig. 6.2(b), where the controller is not connected directly to the control network. In this case, the controller is said to be remote. The gateway is responsible for making the bridge between the control network and the network which the controller and SCADA are connected to. Finally, Fig. 6.2(c) shows a configuration in which the controller is connected to a web server through the internet, which in turn, is connected to the gateway. Note that in this case, it is important to have a local controller for the case the internet connection is lost.

Regarding the control network, multiple types of physical mediums and protocols can be employed. Once a specific protocol is chosen, all nodes must have a network interface capable of handling the reception and transmission of information.

The architecture in which the testbed was implemented is based on the one with the local controller, shown in Fig. 6.2(a). The local controller is responsible for implementing the DES control, which includes the supervisory control. The next subsection gives some details about the logical structure of the local controller.

6.1.1 DES control architecture

The DES control is responsible for coordinating the operation of all subsystems that comprise a given system by implementing supervisors obtained through the supervisory control theory. As described in Subsection 3.4, previous works present different approaches for such implementations, and all of them agree on the point where controllable events are associated with *commands* and uncontrollable events are associated with *responses*. The start point for the proposed DES control architecture is the setup shown in Fig. 3.4, repeated in Fig. 6.3.



(a) Local controller. (b) Remote controller. (c) Controller over internet.

Figure 6.2: NCS topologies.

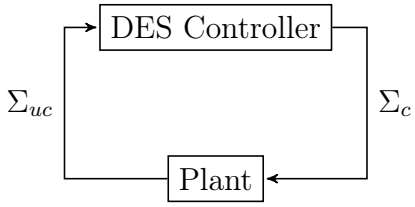


Figure 6.3: Feedback loop of DES control.

The supervisor obtained by the supervisory control theory is not enough to implement the controller of Fig. 6.3, since the supervisor is only a function whose output is a set of enabled events, while the controller needs to send to the plant only a controllable event chosen among the set of enabled events. To implement the DES control, a new element has to be added, resulting in the scheme represented by Fig. 6.4.

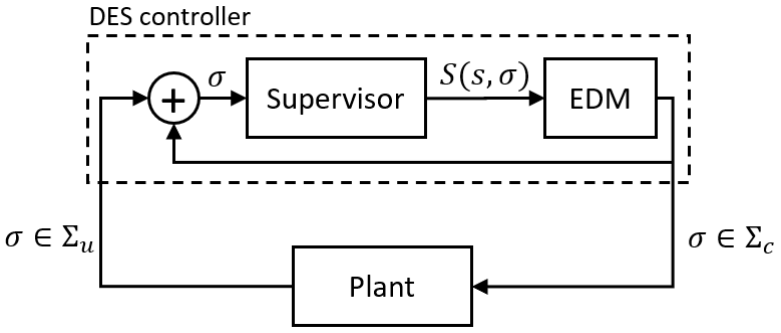


Figure 6.4: Implemented control loop of SCT.

In Fig.6.4, EDM stands for *Event Decision Maker*, which is a component that has as

input the control action of the supervisor, i.e., the set of enabled events, and has as output a single controllable event that will be sent to the plant. In this scheme, supervisor and EDM are components of the *DES controller*, shown in a dashed line in Fig. 6.4. Now, the EDM receives the set $S(s, \sigma)$, which corresponds to the control action of the supervisor S after receiving event σ and considering a string s of previous events that were executed by the system under control. The EDM will choose an event $\sigma' \in S(s, \sigma) \cap \Sigma_c$ to be sent to the plant. This will happen until the set $S(s, \sigma) \cap \Sigma_c$ becomes empty. In such a case, the DES controller can only wait for the occurrence of an uncontrollable event in the plant. Note that for every event σ' chosen by the EDM, this event is also sent back to the supervisor, allowing it to update its current state. How the choice is made by the EDM will depend on the implemented rule. A few possibilities are:

- Event priority: the EDM has information about which event has priority over the others;
- Predefined sequence: the EDM has an ordered list of events and the decisions are made according to this list. This makes sense when an optimization is in place and a best sequence is known (Pena et al., 2022; Alves et al., 2021);
- Random: the EDM chooses an event randomly.

Now that a DES controller is defined, the next step towards the implementation of the DES control is to define where the conversion from events to electrical signals and from electrical signals to events happen. The authors of (De Queiroz and Cury, 2002) propose a three-level structure for the implementation of supervisory control, already shown in Fig. 3.5. The three-level control system architecture was designed to be implemented in a single device, as a PLC. However, the proposed testbed is a prototype of an NCS. Hence a combination of the NCS topology shown in Fig. 6.2(a) and the three-level architecture was needed. The obtained result is illustrated by Fig. 6.5.

In the DES control architecture of Fig. 6.5, each device connected to the network is called a *node*. A node can be the *global controller*, identified by the number (1) in the top right corner of the block global controller in Fig. 6.5, or a *local controller* (blocks (5)). The global controller includes the modular supervisors (block (2)) and the product system (block (3)). The supervisors obtained are local modular supervisors (De Queiroz and Cury, 2000) and, in order to reduce even more the size of the automata that have to be coded, a supervisor reduction technique is applied (Su and Wonham, 2004). Then, the information about which event is disabled or enabled in each of the states of the reduced supervisor is also obtained.

The modular supervisor (block (2)) is informed of the occurrence of all events in the system. Upon the reception of an event σ, each supervisor that has a transition labeled with σ will make the transition, updating their current state. Each state of the modular supervisors

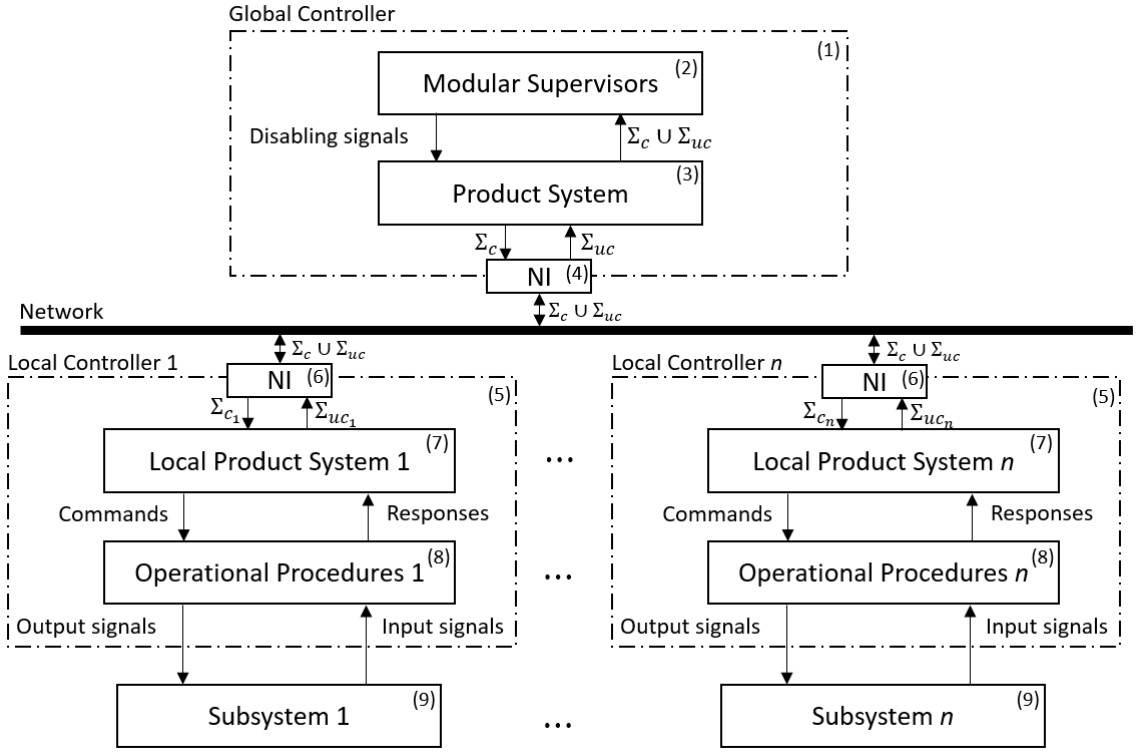


Figure 6.5: Proposed DES control architecture.

has associated a set of controllable events that are disabled. If a particular event is disabled by at least one of the supervisors, then the event is assumed to be disabled and this information is sent to the product system level.

The product system (block (3)) controls the evolution of the system. The internal structure of this block includes all the models of the subsystems and the EDM, which receives from each subsystem model i the set $\Gamma_{ci}(q) = \Gamma_i(q) \cap \Sigma_{ci}$, corresponding to the set of feasible controllable events at state q of subsystem i. The set $\Gamma_c = \bigcup_{\forall i} \Gamma_{ci}(q)$ is the set of feasible controllable events of the overall system at a given moment. Note that this set may change whenever one of the subsystems changes its corresponding state. The EDM also receives the set of disabled events $\mathcal{D}$, information that is provided by the block modular supervisors. Whenever the set $\Gamma_c \setminus \mathcal{D}$ is not empty, the EDM will choose an event to be triggered. If the set $\Gamma_c \setminus \mathcal{D}$ is empty, the EDM waits for the reception of an uncontrollable event, which comes from the subsystems through the network. Upon reception of an uncontrollable event, all subsystem models and supervisors are updated. Note that this may change the sets $\mathcal{D}$ and Γ_c.

Once a controllable event is chosen by the EDM, it will be sent into the network through the network interface (NI), represented by block (4) in Fig. 6.5, and sent back to the subsystem and supervisor models, allowing them to update their current state. All nodes connected to the network (blocks (5)) will receive the event, but if a node does not have

that event defined in its alphabet, the event will be simply ignored. This selection of events is done by the network interface (blocks (6)). If the event is accepted into a node, it will be treated by the local product system (blocks (7)). The local product system is responsible for translating controllable events into commands and responses into uncontrollable events.

Finally, the operational procedures level (blocks (8)) is responsible for mapping commands to the physical signals and the signals back to responses. A human operator can also interact with the system if we model the human-machine interface as a node that is also connected to the network. Events generated by such a node are uncontrollable from the global controller's point of view, since they cannot be prevented from happening.

The proposed implementation scheme is applicable to a wide range of industrial processes that have a centralized controller. Furthermore, the technology employed to implement it can vary from PLCs to micro-controlled based devices. Also, the communication protocol does not necessarily need to be the same for all devices. The condition is that the controller has to be able to communicate with all devices.

In the next section the physical process for which the testbed was designed is described.

6.2 Physical process

The physical process is represented by the P&I (process and instrumentation) diagram of Fig. 6.6 and its goal is to allow the production of batches of a liquid that is the product of chemical reactions that happen at different temperatures inside a reactor.

Upon initialization by a human operator, the process starts with the admission of three different liquids. It is assumed that the flow of each liquid is defined by a previous process. There are three on-off input valves, each one for a different liquid, that receive the same control action, that is, they open and close together. Since the valves remain open for the same amount of time, the proportion of each liquid in the tank is determined by their flow.

The total amount of liquid in the tank, is a controlled variable and it is measured by a level sensor LT. Once the level in the tank reaches the setpoint, the input valves are closed and the mixer is turned on. The mixer makes the mix more homogeneous and therefore diminishes the total time of the batch production.

A heating control system is also in place and after the mixer has started, a continuous-time PI controller TC controls the heater in order to make the temperature, measured by a

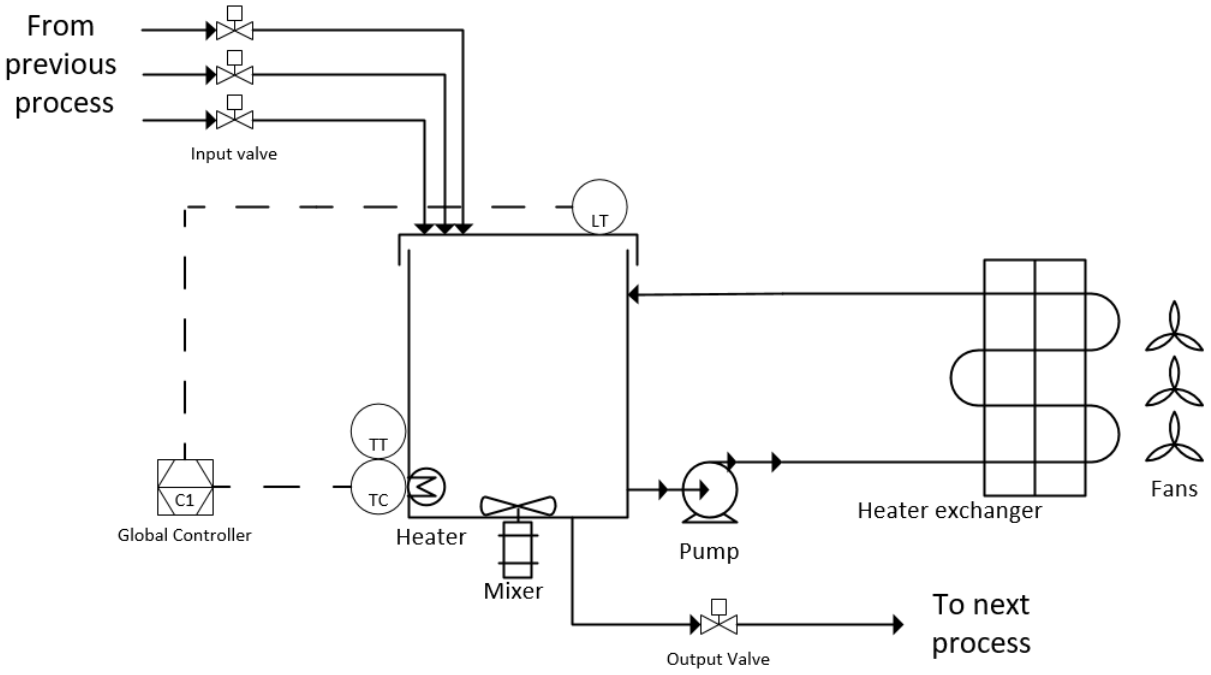


Figure 6.6: P&I diagram of the physical process.

sensor TT, reach a desired value and remain there for a predefined period of time. After that, the controller makes the temperature reach a second predefined value, which is lower than the previous one, for another amount of time. A heater installed in the tank makes it possible to increase the temperature. To allow the liquid to cool faster, there is a heat exchanger coupled to the tank. To make the liquid go through it, a pump is turned on. Once this part of the process is finished, the mixer and the pump stop and an output valve is opened. The valve remains open until the level sensor indicates that the tank has reached the low level. When this happens, the output valve is closed and the batch is finished. The system is now ready to produce another batch.

The models of plants and supervisors are presented in Appendix A. In the next section, the details regarding the prototype's hardware are discussed.

6.3 Hardware

This section describes the hardware employed to implement the DES control architecture. The topology of the testbed is shown in Fig. 6.7. The DES control runs in the lower two levels of the topology, between the global controller and the local controllers LC_1 to LC_3. In the top level a SCADA allows an operator to command the start of a batch production and to monitor the process. The hardware corresponding to the physical components of the system such as the tank, input and output valves, mixer, pump and temperature sensor were simplified and represented

as LEDs for the valves and motors. An RC (resistor-capacitor) circuit emulates the liquid temperature in the tank while a trimpot allows the local controller to make the correspondence between an analog signal and events related to the level. The physical implementation can be seen in Fig. 6.8.

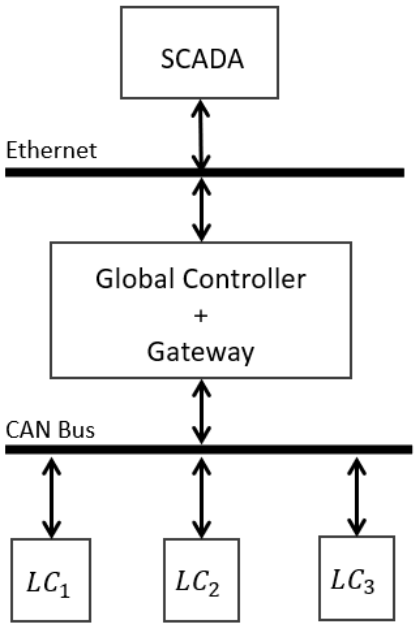


Figure 6.7: Testbed architecture.

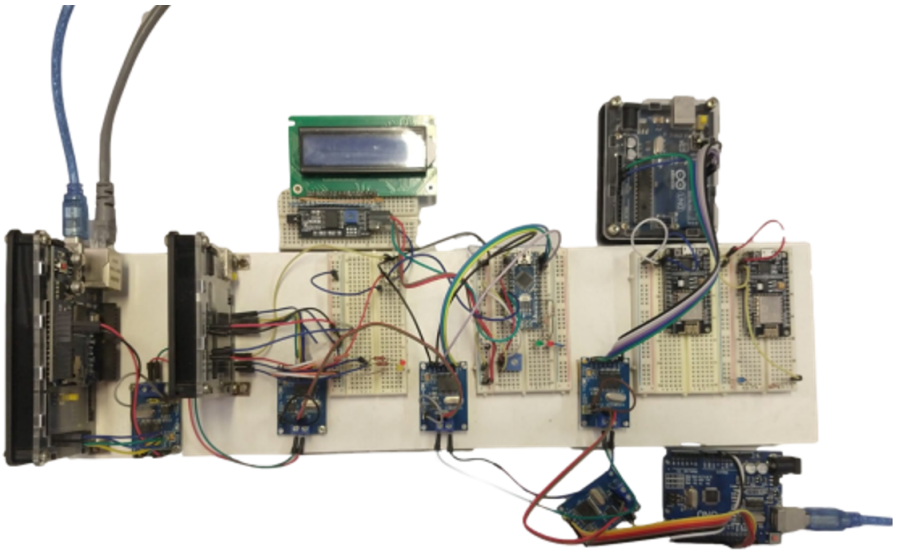


Figure 6.8: Physical implementation.

The hardware uses Arduino, which is an open-source electronics platform based on easy-to-use hardware and software. Arduino also simplifies the process of working with microcontrollers by wrapping up details related to their setup. Furthermore, according to its developers, the Arduino platform has the following characteristics:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than BRL 100,00.

- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well.

- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it is based.

- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it.

The presented characteristics are in consonance with the testbed's goals and that is the reason why Arduino was chosen. The Arduino hardware can be classified in two main groups: boards and shields. An Arduino board has a microcontroller that can be programmed by the user and a set of input and output pins that allows it to exchange information with the external world while a shield or module is a circuit board that is connected to an Arduino board to add extra functionalities to it. In the next two subsections details about the boards and shields employed in the implementation are given.

6.3.1 Arduino boards

The prototype was implemented using several types of Arduino boards. A brief summary of their specifications is given next. For more details, the reader is referred to the Arduino website[2].

- Arduino Mega 2560 - It is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins and 16 analog inputs. It has 8KB of SRAM memory, 256KB of FLASH memory and 4KB of EEPROM memory.

- Arduino UNO - It is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins and 6 analog inputs. It has 2KB of SRAM memory, 32KB of FLASH memory and 1KB of EEPROM memory.

[2]Available at https://docs.arduino.cc/

- Arduino Nano - It is a microcontroller board based on the ATMega328. It has 14 digital input/output pins and 8 analog inputs. It has 2KB of SRAM memory, 32KB of FLASH memory and 1KB of EEPROM memory.

6.3.2 Arduino Modules

Although the Arduino boards have some built-in communication capabilities, they have limitations that do not allow the implementation of the system architecture described in Section 6.1. Thus, two communication-related modules were added, in addition to a third one that is an LCD display.

In this implementation, each Arduino board in addition to its modules and other attached components is called a *node*. The communication between nodes is done by means of the CAN (controller area network) protocol. The CAN protocol provides a bus network to which all nodes are connected. Because of its characteristics, any message sent in the network by a given node is seen by all other nodes in the network. Each node decides if the message is addressed to itself. If it is not, the message is simply ignored.

The module MCP2515, shown in Fig. 6.9, provides an interface between the nodes and the network.

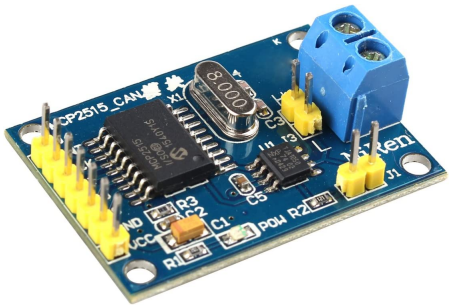


Figure 6.9: CAN module MCP2515[3].

For the communication between the global controller and the SCADA, an Ethernet network was employed. For this reason, an Ethernet shield, as the one shown in Fig. 6.10 was connected to the global controller.

The schematics of each node are presented in Appendix B. In next subsection the details about the software developed to run the DES control system are presented.

[3]Image available at https://circuitdigest.com/microcontroller-projects/arduino-can-tutorial-interfacing-mcp2515-can-bus-module-with-arduino. Accessed on 27/08/2022.

[4]Image available at: https://www.aranacorp.com/en/connect-arduino-to-the-web-using-ethernet-shield-w5100/. Accessed on 27/08/2022.

Figure 6.10: Ethernet shield[4].

6.4 Software

The software that was developed can be divided into two parts: SCT library and SCADA.

6.4.1 SCT library

The discrete-event models and the supervisory control is implemented by using the SCT Library. It has two modes of operation, depending on which type of node, plant or controller, it is applied to. For the controller, the first step to use it is to model each subsystem as an automaton as well as the specifications. The subsystem models should have disjoint alphabets. Once the models are obtained, the automata implementing the reduced supervisors can be obtained, by any computational tool. In this work, Ultrades (Alves et al., 2017) was used. The use of a reduced supervisor is for the purpose of having smaller automata, when compared with automata generated by the classical supervisor synthesis. Another important piece of information needed is the list of disablements, that is, the information about which events are disabled or enabled at each one of the states of the supervisors.

Once the models are available, the next step is to map each event to an integer. Controllable events are mapped to odd numbers while uncontrollable events are mapped to even ones. One of the basic data types in the SCT Library is the State, which is a `struct` with 3 attributes, as shown in Fig. 6.11. During the instantiation of a State, the parameters `on_enter` and `on_exit` are pointers to action functions that are executed when the state is entered and when the state is exited, respectively. A state is entered in the moment it becomes the current state of the automaton and is exited in the moment it is no longer the current state.

Before building the automata, all of their states have to be declared first, as well as the

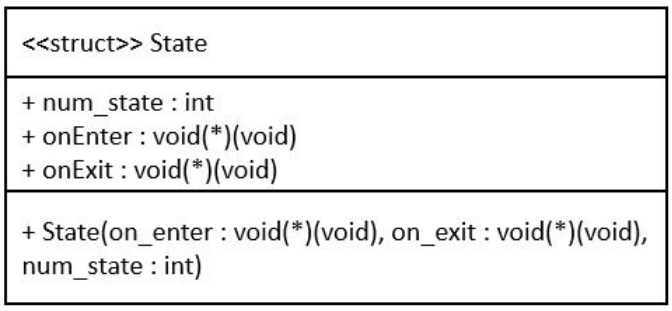


Figure 6.11: State data type representation.

action functions, if they are needed. More details on what the action functions should do will be given later.

Now the automata can be instantiated. The representation for the Automaton class is shown in Fig. 6.12. The Supervisor class is a child of the Automaton class, as shown in Fig. 6.13, since it is also an automaton with additional specific methods to enable or disable events. As can be seen, the initial state of the Automaton and Supervisor objects are defined during their instantiation.

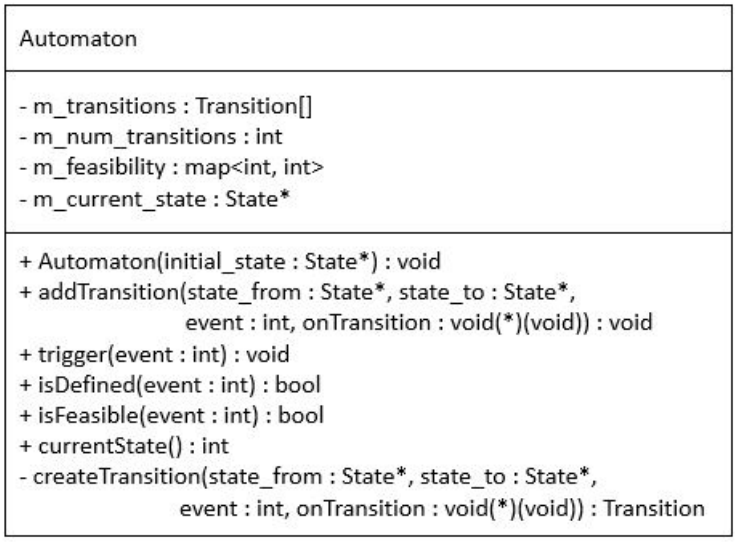


Figure 6.12: Automaton class.

Next, the DES object needs to be instantiated. The representation for the DES class is given in Fig. 6.14. During its instantiation, vectors of integers that are associated with the controllable and uncontrollable events, as well as their length should be provided. A DES object will handle the dynamic of the system under control. The methods `addPlant` and `addSupervisor` have to be used in order to add the automata models to the system. When the method `updateDES` is called, it verifies if a controllable event can be triggered and triggers it. A controllable event cannot be triggered if it is disabled by at least one of the supervisors.

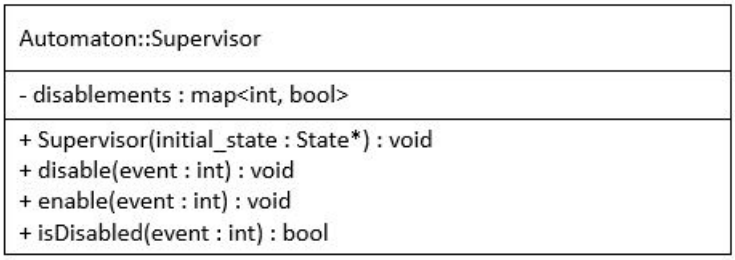


Figure 6.13: Supervisor class.

Once controllable events are enabled, they will be triggered one by one until there are none. The rule that determines the order in which the controllable events will be triggered can be set with method `setMode`, which receives an integer corresponding to one of the following three rules: 1) random; 2) pre-defined order and; 3) priority list. To trigger uncontrollable events, the method `triggerIfPossible` should be used. The method will update all automata in which a transition labeled with that event is feasible.

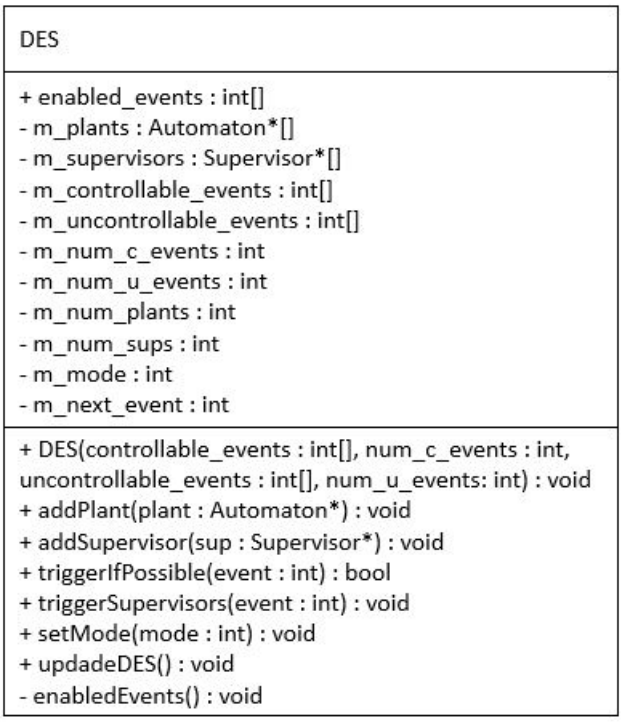


Figure 6.14: DES class.

Before the system can be initialized, each automaton needs to be fully defined, by adding its transitions. This is done by the `addTransition` method. Finally, the system can be initialized by calling the `updateDES` method. It will trigger controllable events if they are enabled or the system will wait for an uncontrollable event to happen.

The action functions will have different roles depending on the type of automaton they are defined for. For plant automata, the action function is responsible for sending the controllable events through the communication channel. For the supervisors, the action function will define which events are enabled or disabled at that state. This is done by calling the methods `disable` and `enable`.

For the plant nodes, a DES object has to be created and to which only the plant automata need to be added, in the same way as is done for the controller node. The state action function in the plant nodes is where the user will code the translation of controllable events to electrical signals, by setting or resetting a single output pin or by initiating a sequence of operations. Additionally, the user has to code the translation from electrical signals to uncontrollable events.

6.4.2 Communication

Regarding the communication, the specifics of the CAN protocol are handled by the CAN library. In order to send a packet, three functions have to be called in the following order:

1. `beginPacket()`: receives the packet id and fills out the headers of the CAN packet;
2. `write()`: receives a byte of data to be transmitted. If the data has more than one byte, this function has to be called multiple times;
3. `endPacket()`: encloses the packet and sends it through the CAN module.

The id of a packet can be used to identify the device that has send it or to identify the type of message. In this implementation, the id is used to distinguish discrete events from the continuous-time variables. To receive a packet, the following functions have to be used.

1. `parsePacket()`: verifies if there is an incoming packet. It returns the size of a packet, if it exists, or zero otherwise;
2. `packetId()`: retrieves the packet id;
3. `read()`: reads the data from the packet. Each call to this function returns a byte of data. If the packet load has more than one byte, than this function has to be called multiple times.

In order to concentrate the communication handling in a specific part of the code, which will allow the user to easily simulate the action of the attacker, two queues were employed. One queue handles the incoming packets and the other one handles the outgoing ones. Thus, whenever a state action function needs to send a controllable event to the plant, the event is first added to the queue of outgoing packets. The program checks periodically if this queue is

not empty. When there is a new event in the queue, a packet is sent and the event is removed from it. For the incoming events, the program checks periodically if there is a new packet. Once a new packet arrives, the event is obtained from it and it is added to the queue of incoming events. Once this queue is not empty, the event is removed from it and it is triggered by the `triggerIfPossible` function.

Another important part of the communication is related to the handling of data exchange between the controller and the SCADA. The protocol used is the Modbus IP. Modbus is an industrial protocol standard that was developed in the late 1970's for communication among PLCs and is still used for connecting industrial devices. The Modbus protocol specification is openly published and use of the protocol is royalty-free. Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as a slave. This means a slave device cannot volunteer information; it must wait to be asked for it. The master will write data to a slave device's registers, and read data from a slave device's registers. A register address or register reference is always in the context of the slave's registers.

It is important to highlight that the only events that are transmitted over the communication between the SCADA and controller are uncontrollable events related to the action of the user, as starting or resetting the system.

6.5 Cyber-Attacks

In the proposed implementation, it is assumed that the attacker has already infiltrated the system. How this is done by the attacker is out of scope of this work. The proposed testbed can easily be used for implementing two types of attacks. The first one is when the attacker is an independent node in the network. In this case, the attacker can insert false occurrences of controllable and uncontrollable events in the network.

The second type of attack that can be simulated is when the attacker infects one of the legitimate nodes in the network. In such a case, the user can alter the function that handles the node's communication and implement the attacker's controller. If the node is a plant, then the user can use the technique presented in Section 5.2 in order to obtain the possible actions at a given moment, if the attacker is a persistent attacker.

The communication is handled by two FIFO queues, one for incoming events and the other for the outgoing events. When a relevant event for a given node arrives, it is immediately added to the associated queue. Then, it is retrieved from the queue and the automata are updated. However, an attack can erase the event by retrieving it from the queue before the models are updated. The attacker can also insert events by adding them into the queue of in-

coming events. This will update the automata although the event was not received by the node.

For the outgoing communication, whenever an event is generated by the node, it is added to queue of outgoing events. This queue is monitored periodically and when an event is detected in it, the event is sent in the network. An attacker can add or remove events from this queue as desired, representing the action of inserting or erasing events, respectively.

In order to detect attacks, a special node can be implemented, called Intrusion Detection System (IDS). The IDS has the models of all subsystems and specifications. It observes all the events exchanged by the nodes and updates its models accordingly. If it detects a behavior that is not legal, then it triggers an alarm.

The next section presents some discussion about the results obtained in this chapter.

6.6 Discussion

The proposed implementation deals with all the problems enumerated in Section 3.4. The problem of causality is solved by assigning the generation of controllable and uncontrollable events to the global controller and local controllers, respectively. The translation of signals to events and events to signals is handled in the local controllers. The issue of having multiple events happening simultaneously is solved by the network's constraint of transmitting only one event at a time. The avalanche effect is not an issue in this implementation since the models are updated only once for each event.

The problem of choice is dealt within the product system level in the global controller. The EDM is responsible for choosing an event among a set of enabled events. The fact that a system under control may be blocking even when the supervisors are nonblocking is a modeling issue. Basically, to avoid the problem one has to guarantee that if a marked state is reachable by a sequence of uncontrollable events, then the system has to reach a state after the execution of controllable events where it can only wait for the occurrence of the uncontrollable ones. Otherwise, a marked state may never be reached if a sequence of uncontrollable events that lead the system to the marked state can be preempted at some point by the occurrence of a controllable event.

The proposed solution does not suffer from the problem of inexact synchronization between plant and supervisor, which can occur when the communication delay is considerable when compared with the time constants of the system. This is not the case for the proposed implementation since events are treated as soon as they are received by the nodes and sent within a time window of a few milliseconds, which is adjustable by the user. However, if delays

are the subject of study, they can be forced to occur by changing the function that handles the reception and transmission of events.

By having access to the incoming and outgoing events of the nodes, different types of attacks can be implemented. The testbed can also be adapted to implement more complex defense techniques than an IDS.

Chapter 7

Conclusion

As the use of CPSs is increasing, carrying with them a voluminous use of communication networks, the surface area exposed to malicious agents is growing as well. This justifies the endeavor of studying different aspects of security in control systems.

In the context of DES control, one of the approaches to ensure the correct system behavior is to design supervisors that are robust to attacks. For attacks in the output symbols, one of the conditions that allows robust supervisors to be designed is the P-observability for an attack set. Although a test for this property already existed, it was based on a series of tests of the classical observability property. The contribution of this thesis is to introduce a new test, that checks for P-observability for an attack set itself and considers the effect of all attackers in a single run, along with the visual explanation and some definitions that explain how an attacker acts, providing insight to understand the attacks under consideration. In addition, an algorithm for checking P-observability for an attack set was presented, that directly applied the definitions proposed and whose overall complexity is $O(|\mathcal{A}||Q||\delta| + |\mathcal{A}||Q|^2 + |Q|^3)$.

Another approach for improving the control system's security, which can seem counter-intuitive, is the study of design techniques for attackers. Nonetheless, having a good understanding about how an attack works facilitate the development of better defense techniques. In this context, the first contribution that this thesis brings is to propose an attack model that is closer to real-world applications and considers that an attack happens at the interface network of the devices that integrate the control system. Considering that the attack has infiltrated one of the subsystems, a new type of attack was introduced, called persistent attack. Differently from other types of attacks described in the literature, which normally have the goal to lead the system to a critical or unsafe state, a persistent attacker wants to anticipate events or to insert small delays in the production process, in a way that it remains stealthy and can act multiple times. Finally, a design technique for such attacks was proposed. Moreover, another contribution of this thesis is the establishment of tools and common ground

that can be adapted to solve more complex problems, such as attacks on controllers, which will be left for future research.

Lastly, this thesis proposes a testbed for testing security-related techniques, which can be implemented by employing low-cost devices. The fact that problems related to the DES control had to be dealt with during the development of the testbed, highlights its importance. Such issues normally do not appear when a simulation, instead of a physical implementation, is employed. Ultimately, the testbed contributes to reducing the gap between theory and practice.

The next subsection presents some directions for future work.

7.1 Future works

Regarding the work on P-observability for an attack set, it is still to be investigated how to extend the property and its test to consider multiple attackers acting at once and on controllable events as well. For the case that a given desired language is not P-observable for an attack set, it would be interesting to have some guidelines on how to improve the security, e.g., by turning one or more vulnerable events into non-vulnerable ones. This is also a possible path for new research.

Concerning the content of Chapter 5, that presents the results on persistent attacks, a natural continuity for the work is to propose a design technique for different attack locations, such as an attack in the controller. These design techniques can also be developed for more complex scenarios, such as in a decentralized DES control system. Furthermore, it is also natural after having explored an attack design technique, to propose a defense technique against such attacks.

Finally, with respect to the testbed implementation, some fronts of future work can be enumerated. The first avenue for facilitating the testbed use could entail implementing a computational tool for automatic generation of code. Secondly, to test possible defense techniques, it would be beneficial to implement various types of attacks. Additionally, it will be useful to have also an architecture of a DES control system with decentralized controllers.

Bibliography

Akinyemi, B. O., Jekoyemi, O. V., Aladesanmi, T. A., Aderounmu, G. A., and Kamagaté, B. H. (2018). A Scalable Attack Graph Generation for Network Security Management. *Journal of Marketing Management*, 6(2):34–43.

Alves, L. V. R., Martins, L. R. R., and Pena, P. N. (2017). Ultrades - a library for modeling, analysis and control of discrete event systems. *Proceedings of the 20th World Congress of the International Federation of Automatic Control*, pages 5831–5836.

Alves, L. V. R., Pena, P. N., and Takahashi, R. H. (2021). Planning on Discrete Event Systems Using Parallelism Maximization. *Control Engineering Practice*, 112(August 2020):104813.

Alves, M. R., Pena, P. N., and Rudie, K. (2022a). Discrete-event systems subject to unknown sensor attacks. *Discrete Event Dynamic Systems: Theory and Applications*, 32(1):143–158.

Alves, M. R. C., Rudie, K., and Pena, P. N. (2022b). A Security Testbed for Networked Control Systems. In *Proceedings of the 16th IFAC Workshop on Discrete Event Systems*, Prague.

Alves, M. V., Barcelos, R. J., Carvalho, L. K., and Basilio, J. C. (2022c). Robust Decentralized Diagnosability of Networked Discrete Event Systems Against DoS and Deception Attacks. *Nonlinear Analysis: Hybrid Systems*, 44(309652):101162.

Alves, M. V., da Cunha, A. E., Carvalho, L. K., Moreira, M. V., and Basilio, J. C. (2019). Robust Supervisory Control of Discrete Event Systems Against Intermittent Loss of Observations. *International Journal of Control*, 7179:1–13.

Alves, M. V. S., Carvalho, L. K., and Basilio, J. C. (2020). Supervisory Control of Networked Discrete Event Systems With Timing Structure. *IEEE Transactions on Automatic Control*, pages 1–13.

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H., and Franklin, G. F. (1993). Supervisory Control of a Rapid Thermal Multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059.

Bhamare, D., Zolanvari, M., Erbad, A., Jain, R., Khan, K., and Meskin, N. (2020). Cybersecurity for Industrial Control Systems: A Survey. *Computers and Security*, 89:1–12.

Cao, L., Jiang, X., Zhao, Y., Wang, S., You, D., and Xu, X. (2020). A Survey of Network Attacks on Cyber-Physical Systems. *IEEE Access*, 8:44219–44227. ISSN 21693536.

Cárdenas, A. A., Amin, S., Lin, Z.-S., Huang, Y.-L., Huang, C.-Y., and Sastry, S. (2011). Attacks against process control systems. page 355.

Carvalho, L. K., Wu, Y. C., Kwong, R., and Lafortune, S. (2016). Detection and Prevention of Actuator Enablement Attacks in Supervisory Control Systems. *13th International Workshop on Discrete Event Systems, WODES 2016*, pages 298–305.

Carvalho, L. K., Wu, Y. C., Kwong, R., and Lafortune, S. (2018). Detection and Mitigation of Classes of Attacks in Supervisory Control Systems. *Automatica*, 97:121–133.

Cassandras, C. and Lafortune, S. (2007). *Introduction to Discrete Event Systems*, volume 11. Springer, New York, 2nd edition.

Chen, Q., Su, R., and Li, Z. (2022). Attackable Detectability of Partially-Observed Discrete-Event Systems under Sensor Attack. In *Proceedings of the 16th IFAC Workshop on Discrete Event Systems*, Prague.

De Oliveira, R. G., de Queiroz, M. H., and Cury, J. E. R. (2020). Synthesis of Supervisors for a PID-Controlled Industrial Process and Implementation on Foundation Fieldbus. In *15th IFAC Workshop on Discrete Event Systems*, pages 83–88.

De Queiroz, M. H. and Cury, J. E. R. (2000). Modular Supervisory Control of Large Scale Discrete Event Systems. In *Boel R., Stremersch G. (eds) Discrete Event Systems*, pages 103–110. Springer, Boston, MA.

De Queiroz, M. H. and Cury, J. E. R. (2002). Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. *Proceedings of the 6th International Workshop on Discrete Event Systems, WODES 2002*, pages 377–382.

Dibaji, S. M., Pirani, M., Flamholz, D. B., Annaswamy, A. M., Johansson, K. H., and Chakrabortty, A. (2019). A Systems and Control Perspective of CPS Security. *Annual Reviews in Control*, 47:394–411.

Dietrich, P., Malik, R., Wonham, W. M., and Brandin, B. A. (2002). Implementation Considerations in Supervisory Control. In Caillaud, B., Darondeau, P., Lavagno, L., and Xie, X., editors, *Synthesis and Control of Discrete Event Systems*, pages 185–201. Kluwer Academic Publishers.

Ding, D., Han, Q. L., Ge, X., and Wang, J. (2021). Secure State Estimation and Control of Cyber-Physical Systems: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):176–190.

Ding, D., Han, Q. L., Xiang, Y., Ge, X., and Zhang, X. M. (2018). A survey on security control and attack detection for industrial cyber-physical systems. *Neurocomputing*, 275:1674–1683.

Duo, W., Zhou, M. C., and Abusorrah, A. (2022). A Survey of Cyber Attacks on Cyber Physical Systems: Recent Advances and Challenges. *IEEE/CAA Journal of Automatica Sinica*, 9(5):784–800.

Fabian, M. and Hellgren, A. (1998). PLC-Based Implementation of Supervisory Control for Discrete Event Systems. *Proceedings of the IEEE Conference on Decision and Control*, 3(December):3305–3310.

Fadlallah, A., Sbeity, H., Malli, M., and Lteif, P. (2016). Application of Attack Graphs in Intrusion Detection Systems: An Implementation. *International Journal of Computer Networks*, (81):2016–1.

Fritz, R., Schwarz, P., and Zhang, P. (2019). Modeling of Cyber Attacks and a Time Guard Detection for ICS Based on Discrete Event Systems. *Proceeding of the 18th European Control Conference*, pages 4368–4373.

Fritz, R. and Zhang, P. (2018). Modeling and detection of cyber attacks on discrete event systems. *IFAC-PapersOnLine*, 51(7):285–290.

Gao, C., Seatzu, C., Li, Z., and Giua, A. (2019). Multiple Attacks Detection on Discrete Event Systems. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pages 2352–2357.

Ginter, Andrew (2018). *The Top 20 Cyberattacks on Industrial Control Systems*. Waterfall Security Solutions.

Giraldo, J., Urbina, D., Cardenas, A., Valente, J., Faisal, M., Ruths, J., Tippenhauer, N. O., Sandberg, H., and Candell, R. (2018). A Survey of Physics-Based Attack Detection in Cyber-Physical Systems. *ACM Computing Surveys*, 51(4).

Hemsley, K. E. and Fisher, R. E. (2018). History of Industrial Control System Cyber Incidents. *INL/CON-18-44411-Revision-2*, (December):1–37.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages and Computation*. Prentice Hall, 3rd edition.

Jang-Jaccard, J. and Nepal, S. (2014). A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973–993.

Khoumsi, A. (2019). Sensor and Actuator Attacks of Cyber-Physical Systems: A Study Based on Supervisory Control of Discrete Event Systems. *Proceedings of the 8th International Conference on Systems and Control*, pages 176–182.

Leitão, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., and Colombo, A. W. (2016). Smart Agents in Industrial Cyber-Physical Systems. *Proceedings of the IEEE*, 104(5):1086–1101.

Li, Y., Tong, Y., and Giua, A. (2020). Detection and Prevention of Cyber Attacks in Networked Control Systems. In *Proceedings of the 15th IFAC Workshop on Discrete Event Systems*, pages 7–13.

Lima, P. M., Alves, M. V., Carvalho, L. K., and Moreira, M. V. (2017). Security Against Network Attacks in Supervisory Control Systems. *IFAC-PapersOnLine*, 50(1):12333–12338.

Lima, P. M., Alves, M. V., Carvalho, L. K., and Moreira, M. V. (2022). Security of Cyber-Physical Systems: Design of a Security Supervisor to Thwart Attacks. *IEEE Transactions on Automation Science and Engineering*, 19(3):2030–2041.

Lima, P. M., Alves, M. V. S., Carvalho, L. K., and Moreira, M. V. (2019). Security Against Communication Network Attacks of Cyber-Physical Systems. *Journal of Control, Automation and Electrical Systems*, 30(1):125–135.

Lima, P. M., Carvalho, L. K., and Moreira, M. V. (2018). Detectable and Undetectable Network Attack Security of Cyber-physical Systems. *IFAC-PapersOnLine*, 51(7):179–185.

Lin, F. and Wonham, W. M. (1988). On Observability of Discrete-Event Systems. *Information Sciences*, 44(3):173–198. ISSN 00200255.

Lin, L. and Su, R. (2020). Synthesis of Covert Actuator Attackers for Free. In *Proceedings of the 15th IFAC Workshop on Discrete Event Systems*, number 1-6, pages 561–577.

Lin, L., Thuijsman, S., Zhu, Y., Ware, S., Su, R., and Reniers, M. (2019a). Synthesis of Supremal Successful Normal Actuator Attackers on Normal Supervisors. In *Proceedings of the 2019 American Control Conference*, pages 5614–5619. American Automatic Control Council.

Lin, L., Zhu, Y., and Su, R. (2019b). Towards Bounded Synthesis of Resilient Supervisors against Actuator Attacks. In *Proceedings of the 58th IEEE Conference on Decision and Control*, pages 7659–7664.

Lin, L., Zhu, Y., and Su, R. (2020). Synthesis of Covert Actuator Attackers for Free. *Discrete Event Dynamic Systems*, 30:561–577.

Lu, Y. (2017). Cyber Physical System (CPS)-Based Industry 4.0: A Survey. *Journal of Industrial Integration and Management*, 02(03):17500141–175001457.

Mahmoud, M. S., Hamdan, M. M., and Baroudi, U. A. (2019). Modeling and Control of Cyber-Physical Systems Subject to Cyber Attacks: A Survey of Recent Advances and Challenges. *Neurocomputing*, 338:101–115.

Matthews, I., Mace, J., Soudjani, S., and van Moorsel, A. (2020). Cyclic Bayesian Attack Graphs: A Systematic Computational Approach.

Meira-Goes, R., Kang, E., Kwong, R., and Lafortune, S. (2017). Stealthy Deception Attacks for Cyber-Physical Systems. In *Preceedings of the 56th IEEE Annual Conference on Decision and Control*, pages 4224–4230.

Meira-Góes, R., Kang, E., Kwong, R. H., and Lafortune, S. (2020). Synthesis of Sensor Deception Attacks at the Supervisory Layer of Cyber–Physical Systems. *Automatica*, 121:109172.

Meira-Goes, R., Lafortune, S., and Marchand, H. (2021). Synthesis of Supervisors Robust Against Sensor Deception Attacks. *IEEE Transactions on Automatic Control*, pages 1–8.

Meira-Goes, R., Marchand, H., and Lafortune, S. (2019). Towards resilient supervisors against sensor deception attacks. *Proceedings of the IEEE Conference on Decision and Control*, 2019-December(Cdc):5144–5149.

Mohajerani, S., Góes, R. M., and Lafortune, S. (2020). Efficient Synthesis of Sensor Deception Attacks Using Observation Equivalence-Based Abstraction. In *Proceedings of the 15th IFAC Workshop on Discrete Event Systems*, pages 28–34.

Orojloo, H. and Azgomi, M. A. (2019). Modelling and Evaluation of the Security of Cyber-Physical Systems Using Stochastic Petri Nets. *IET Cyber-Physical Systems: Theory & Applications Research*, 4(1):50–57.

Pan, Y., Wu, Y., and Lam, H. K. (2022). Security-Based Fuzzy Control for Nonlinear Networked Control Systems with DoS Attacks via a Resilient Event-Triggered Scheme. *IEEE Transactions on Fuzzy Systems*, 6706(c):1–10.

Pang, Z. H., Fan, L. Z., Sun, J., Liu, K., and Liu, G. P. (2021). Detection of Stealthy False Data Injection Attacks Against Networked Control Systems via Active Data Modification. *Information Sciences*, 546:192–205.

Pena, P. N., Vilela, J. N., Alves, M. R. C., and Rafael, G. C. (2022). Abstraction of the Supervisory Control Solution to Deal with Planning Problems in Manufacturing Systems. *IEEE Transactions on Automatic Control*, 67(1):344–350.

Prinsloo, J., Sinha, S., and von Solms, B. (2019). A Review of Industry 4.0 Manufacturing Process Security Risks. *Applied Sciences (Switzerland)*, 9(23).

Rashidinejad, A., Lin, L., Wetzels, B., Zhu, Y., Reniers, M., and Su, R. (2019). Supervisory Control of Discrete-Event Systems Under Attacks: An Overview and Outlook. In *Proceedings of the 18th European Control Conference*, pages 1732–1739. EUCA.

Rasmussen, T. B., Yang, G., Nielsen, A. H., and Dong, Z. (2017). A Review of Cyber-Physical Energy System Security Assessment. *2017 IEEE Manchester PowerTech*.

Seatzu, C. (2018). Partially observed discrete-event systems: from state estimation to intrusion detection. *IFAC-PapersOnLine*, 51(7):508–511.

Shareef, T. (2022). 9 Times Hackers Targeted Cyberattacks on Industrial Facilities. Make use of. Available at https://www.makeuseof.com/cyberattacks-on-industry-hackers. Accessed on 23/09/2022.

Singh, S., Yadav, N., and Chuarasia, P. K. (2020). A Review on Cyber Physical System Attacks: Issues and Challenges. *Proceedings of the 2020 IEEE International Conference on Communication and Signal Processing*, 2:1133–1138.

Slowik, J. (2020). Evolution of ICS Attacks and the Prospects for Future Disruptive Events.

Su, R. (2018). Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. *Automatica*, 94:35–44.

Su, R. and Wonham, W. M. (2004). Supervisor Reduction for Discrete-Event Systems. *Discrete Event Dynamic Systems*, 14:31–53.

Tahoun, A. H. and Arafa, M. (2021). Cooperative Control for Cyber–Physical Multi-Agent Networked Control Systems with Unknown False Data-Injection and Replay Cyber-Attacks. *ISA Transactions*, 110:1–14.

Tan, S., Guerrero, J. M., Xie, P., Han, R., and Vasquez, J. C. (2020). Brief Survey on Attack Detection Methods for Cyber-Physical Systems. *IEEE Systems Journal*, 14(4):5329–5339.

Vaz, A. F. and Wonham, W. M. (1986). On supervisor reduction in discrete-event systems. *International Journal of Control*, 44(2):475–491.

Vieira, A. D., Cury, J. E. R., and De Queiroz, M. H. (2006). A model for PLC implementation of supervisory control of discrete event systems. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 225–232.

Vieira, A. D., Santos, E. A. P., De Queiroz, M. H., Leal, A. B., De Paula Neto, A. D., and Cury, J. E. (2017). A Method for PLC Implementation of Supervisory Control of Discrete Event Systems. *IEEE Transactions on Control Systems Technology*, 25(1):175–191.

Wakaiki, M., Tabuada, P., and Hespanha, J. P. (2019). Supervisory Control of Discrete-Event Systems Under Attacks. *Dynamic Games and Applications*, 9(4):965–983.

Wang, Q. and Yang, H. (2019a). A survey on the recent development of securing the networked control systems. *Systems Science and Control Engineering*, 7(1):54–64.

Wang, Q. and Yang, H. (2019b). A survey on the Recent Development of Securing the Networked Control Systems. *Systems Science and Control Engineering*, 7(1):54–64.

Wang, W., Lafortune, S., and Lin, F. (2007). An Algorithm for Calculating Indistinguishable States and Clusters in Finite-State Automata with Partially Observable Transitions. *Systems and Control Letters*, 56(9-10):656–661.

Wang, Y., Li, Y., Yu, Z., Wu, N., and Li, Z. (2021). Supervisory Control of Discrete-Event Systems under External Attacks. *Information Sciences*, 562:398–413.

Wang, Y. and Pajic, M. (2019a). Attack-Resilient Supervisory Control with Intermittently Secure Communication. *Proceedings of the IEEE Conference on Decision and Control*, 2019-December(Cdc):2015–2020.

Wang, Y. and Pajic, M. (2019b). Supervisory Control of Discrete Event Systems in the Presence of Sensor and Actuator Attacks. In *Proceedings of the 58th IEEE Conference on Decision and Control*, pages 5350–5355.

Wang, Z. Y., Góes, R. M., Lafortune, S., and Kwong, R. H. (2020). Mitigation of Classes of Attacks using a Probabilistic Discrete Event System Framework. In *Proceedings of the 15th IFAC Workshop on Discrete Event Systems*, pages 35–41.

Weerakkody, S., Liu, X., Son, S. H., and Sinopoli, B. (2017). A Graph-Theoretic Characterization of Perfect Attackability for Secure Design of Distributed Control Systems. *IEEE Transactions on Control of Network Systems*, 4(1):60–70.

Wonham, W. M. and Cai, K. (2019). *Supervisory Control of Discrete-Event Systems*. Springer, Toronto.

You, D., Wang, S., and Seatzu, C. (2022). A Liveness-Enforcing Supervisor Tolerant to Sensor-Reading Modification Attacks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(4):2398–2411.

Zhang, K. and Feng, L. (2020). Revisiting Strong Detectability of Networked Discret-Event Systems. In *Proceedings of the 15th IFAC Workshop on Discrete Event Systems*, pages 21–27.

Zhang, Q., Li, Z., Seatzu, C., and Giua, A. (2018). Stealthy Attacks for Partially-Observed Discrete Event Systems. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2018-Septe:1161–1164.

Zhang, Q., Seatzu, C., Li, Z., and Giua, A. (2020). Cyber Attacks with Bounded Sensor Reading Edits for Partially-Observed Discrete Event Systems. (122):1–20.

Zhang, Q., Seatzu, C., Li, Z., and Giua, A. (2021). Joint State Estimation under Attack of Discrete Event Systems. *IEEE Access*, 9:168068–168079.

Zhou, S., Yu, Z., Nasr, E. S. A., Mahmoud, H. A., Awwad, E. M., and Wu, N. (2020). Homomorphic Encryption of Supervisory Control Systems Using Automata. *IEEE Access*, 8:147185–147198.

Zhu, Y., Lin, L., and Su, R. (2019a). Supervisor Obfuscation against Actuator Enablement Attack. In *Proceedings of the 18th European Control Conference*, pages 1760–1765.

Zhu, Y., Lin, L., Ware, S., and Su, R. (2019b). Supervisor Synthesis for Networked Discrete Event Systems with Communication Delays and Lossy Channels. In *Proceedings of the 58th IEEE Conference on Decision and Control*, pages 6730–6735.

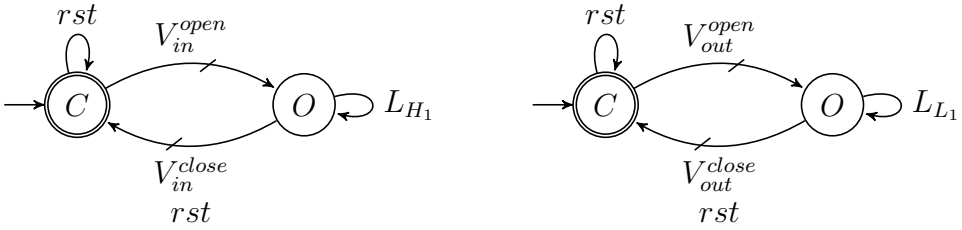
Appendix A

Models

The next two subsections present the models for each one of the subsystems and the models for the specifications, as well as the obtained supervisors.

A.1 Subsystems

The first two automata are the models of the input valve and output valve, shown in Figures A.1(a) and A.1(b), respectively. Notice that, although the physical process has three input valves, it is assumed that they operate synchronized, which allows the use of a single model representing their behavior.



(a) VIN - Input valve.

(b) VOUT - Output valve.

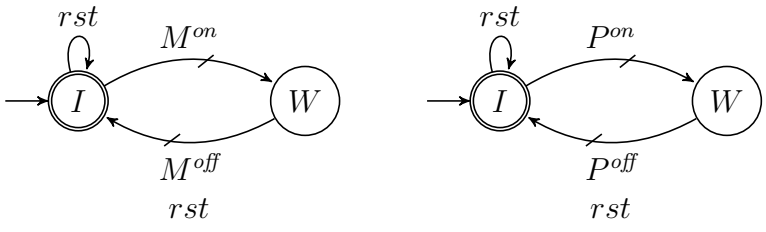
Figure A.1: Models of input and output valves.

The model of the valves has two states, closed (C) and opened (O). Controllable events V_{in}^{open} (V_{out}^{open}) and V_{in}^{close} (V_{out}^{close}) correspond to the commands to open and to close the input valve, VIN (output valve, $VOUT$). Once the input valve is opened, the level sensor can trigger the uncontrollable event L_{H_1}, which signalizes that the tank is full. Note that the action of closing the input valve once the tank is full is not commanded directly by the level sensor; the V_{in}^{close} event has to be sent by the controller.

Regarding the output valve, when it is in the opened state, the level in the tank starts to drop. When the level reaches the lower setpoint, the level sensor triggers the

uncontrollable event L_{L_1}. Additionally, in both models and in the ones that are going to be presented next, a transition labeled with an uncontrollable event rst is added to all states of the automata. Once this event is executed, the automata go back to their initial state, *resetting* the system. This is useful to synchronize the automata in case they go out of sync, which can happen due to an attack or to a malfunction of one of the subsystems.

The next two models are the ones representing the behavior of the mixer and the pump, shown in Figures A.2(a) and A.2(b), respectively. Their behavior is modeled by a two-state automaton, representing the idle (I) and working (W) states. The transitions from idle to working states and from the working state back to idle happen with the execution of the controllable events M^{on} and M^{off} for the mixer and P^{on} and P^{off} for the pump.



(a) Mixer. (b) Pump.

Figure A.2: Models of mixer and pump.

The temperature control of the liquid in the tank is modeled by the automaton of Figure A.3. The states represent that the continuous-time temperature controller is either in idle (I) or in the working (W) state. The controllable events T^{on} and T^{off} are responsible for turning the temperature control on and off, respectively. When the temperature control is working, a temperature sensor can trigger the uncontrollable events *heated* and *cooled*, each one associated to a previously adjusted setpoint.

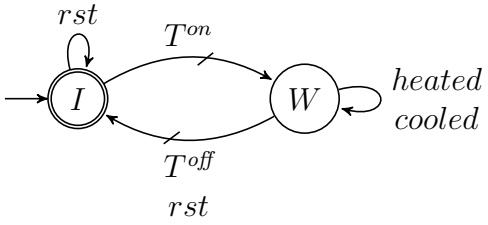


Figure A.3: Temperature control plant

The last plant model does not model a specific subsystem, but rather the whole process, which can be in idle (I) or working (W) states. The automaton is shown in Figure A.4. This model is useful when modeling the specifications that constrain the functioning of a subsystem

to to beginning or to end of a production cycle, as will be seen in the next section. The transition from state idle to working is made by the uncontrollable event *start*, which is triggered when a human operator presses a button to start the production of a batch. Once the batch is finished, the event $finish$ is triggered in the controller.

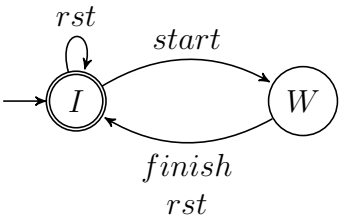


Figure A.4: Process automaton

In the next subsection the models of specifications and supervisors are presented.

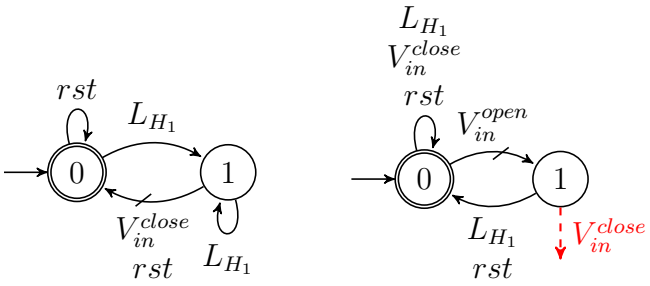
A.2 Specifications and supervisors

With the goal to specify the correct order in which each subsystem in the physical process has to operate to produce a batch of the final product, a set of specifications were designed. The use of specifications with a small number of states, one for each of the necessary conditions for batch production was preferred over more complex automata, where a single one represents a set of conditions. In other words, it was preferred to obtain a large number of small automata instead a small number of large automata, since small automata are easier to code.

For each one of the specifications that will be presented next, a reduced supervisor was also obtained. The local modular synthesis (De Queiroz and Cury, 2002) combined with the supervisor reduction technique (Su and Wonham, 2004) was employed in order to obtain even smaller automata, which are easier to code in the controller. Red dashed arrows represent that a specific event is disabled by the supervisor at a given state. If an event is not disabled, then the event is enabled. This information about event disablement/enablement is the control action of the supervisor at each of its states.

The specification E_1, shown in Fig. A.5(a), makes the occurrence of event V_{in}^{close} conditional upon the occurrence of event L_{H_1}. This means that the input valve can only close after the tank is full. The corresponding supervisor, S_1, is shown in Fig. A.5(b). Notice that event V_{in}^{close} is disabled after the occurrence of V_{in}^{open} until event L_{H_1} occurs.

The specification E_2 of Fig. A.6(a) makes the occurrence of event V_{out}^{close} conditional upon the occurrence of event L_{L_1}. In other words, once the output valve is opened, it has to

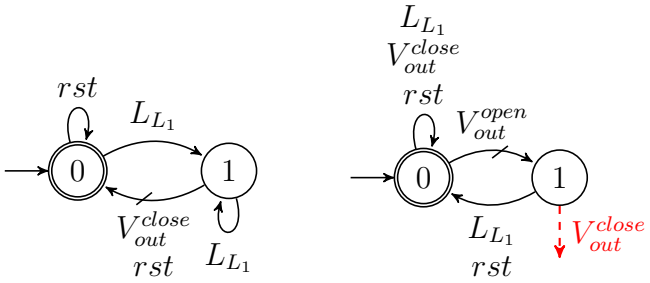


(a) Specification E_1 - Input valve closing

(b) Supervisor S_1

Figure A.5: Specification E_1 and Supervisor S_1

remain that way until event the tank is empty. The corresponding supervisor S_2 is shown in Fig. A.6(b), which disables event V_{out}^{close} before the occurrence of event L_{L_1}.



(a) Specification E_2 - Output valve closing and tank

(b) Supervisor S_2

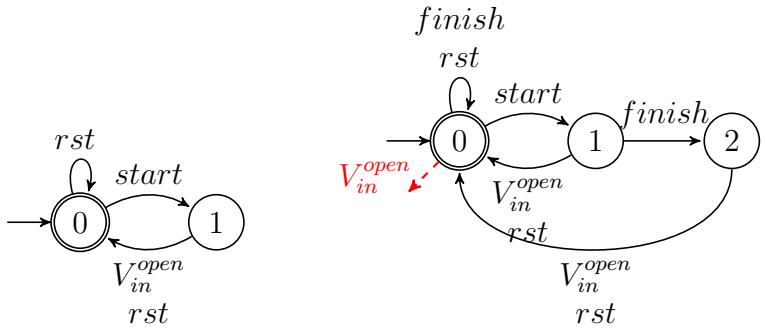
Figure A.6: Specification E_2 and Supervisor S_2

The requirement for the input valve is that event *start* must occur before the input valve is opened, which is modeled by the specification E_3 of Fig. A.7(a). The corresponding supervisor S_3 is shown in Fig. A.7(b); which disables event V_{in}^{open} before the occurrence of event *start*.

Regarding the output valve, it can only open after the liquid has been cooled, which means that the final product is ready, condition modeled by specification E_4, shown in Fig. A.8(a). The corresponding supervisor S_4, shown in Fig. A.8(b), disables event V_{out}^{open} before the occurrence of event *cooled*.

The specification E_5 of Fig. A.9(a) makes the starting of the mixer conditional upon the filling of the tank. For this, the supervisor S_5, shown in Fig. A.9(b), disables event M^{on} before the occurrence of event L_{H_1}.

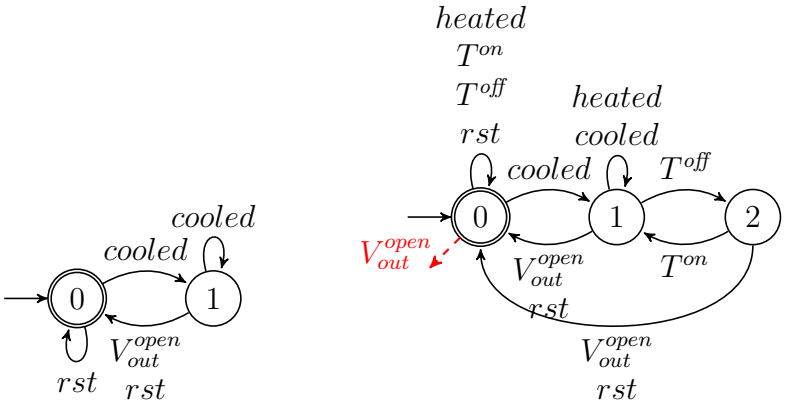
Once the mixer is working, it can only stop when the liquid cools down. Thus, specifi-



(a) Specification E_3 - Input valve opening and process

(b) Supervisor S_3

Figure A.7: Specification E_3 and Supervisor S_3



(a) Specification E_4 - Output valve opening and process

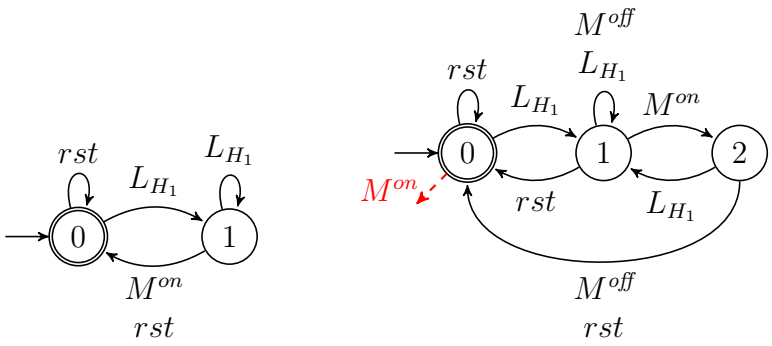
(b) Supervisor S_4

Figure A.8: Specification E_4 and Supervisor S_4

cation $A.10(a)$ makes the execution of event M^{off} conditional upon the occurrence of event *cooled*. This is accomplished by the supervisor S_6 of Fig. A.10(b), which disables event M^{off} while event *cooled* has not yet occurred.

The pump can only be turned on after the liquid has been heated, when the cooling phase starts. This requirement is modeled by the specification E_7, shown in Fig. A.11(a). The corresponding supervisor S_7, shown in Fig. A.11(b), disables event P^{on} while event *heated* has not occurred.

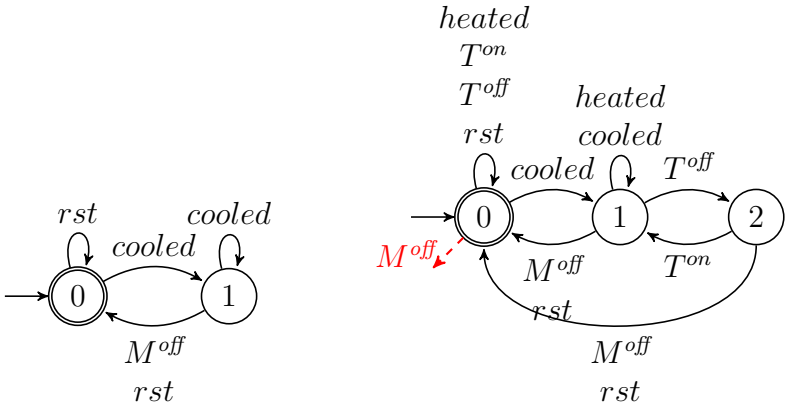
In order to be possible for the pump to stop working, the liquid has to be completely cooled down. This condition is modeled by specification E_8 of Fig. A.12(a). The supervisor S_8, shown in Fig. A.12(b), disables event P^{off} until event *cooled* occurs.



(a) Specification E_5 - Starting of mixer and input valve

(b) Supervisor S_5

Figure A.9: Specification E_5 and Supervisor S_5



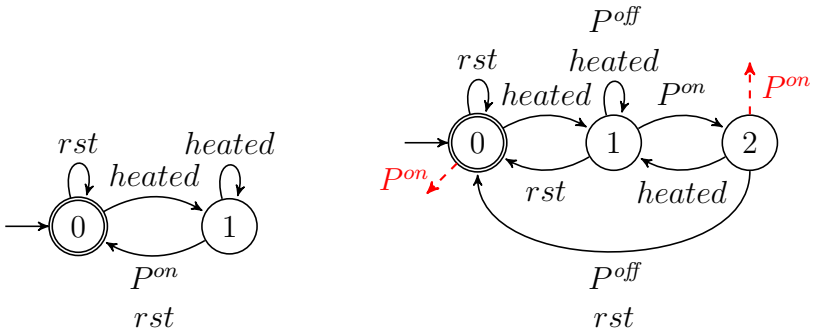
(a) Specification E_6 - Stopping of mixer and temperature control

(b) Supervisor S_6

Figure A.10: Specification E_6 and Supervisor S_6

The continuous-time temperature control can only start to operate once the tank is filled, a condition captured by specification E_9, illustrated in Fig. A.13(a). The corresponding supervisor S_9, shown in Fig. A.13(b), disables event T^{on} while waiting for the occurrence of event L_{H_1}.

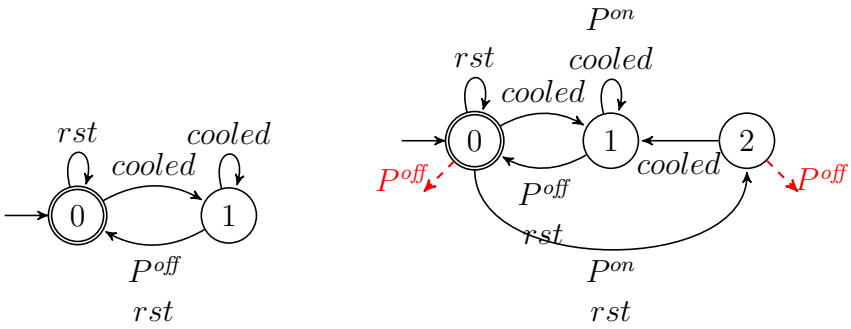
Finally, the last specification, E_{10}, constrains the stopping of the continuous-time temperature controller to cooling of the liquid. The specification is shown in Fig. A.14(a). Supervisor S_{10} disables event T^{off} while event *cooled* has not occurred.



(a) Specification E_7 - Starting of pump and temperature control

(b) Supervisor S_7

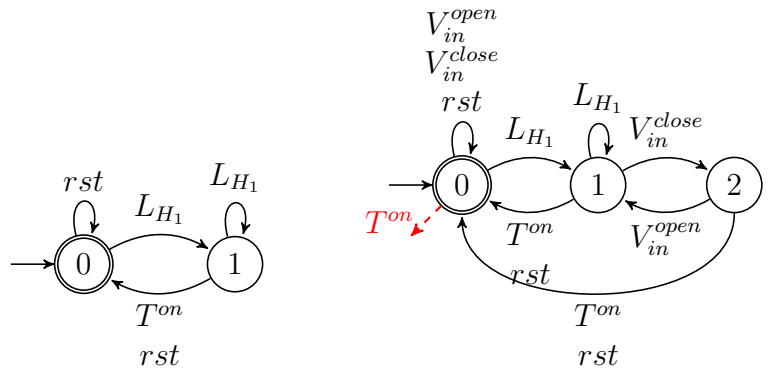
Figure A.11: Specification E_7 and Supervisor S_7



(a) Specification E_8 - Stopping of pump and temperature control

(b) Supervisor S_8

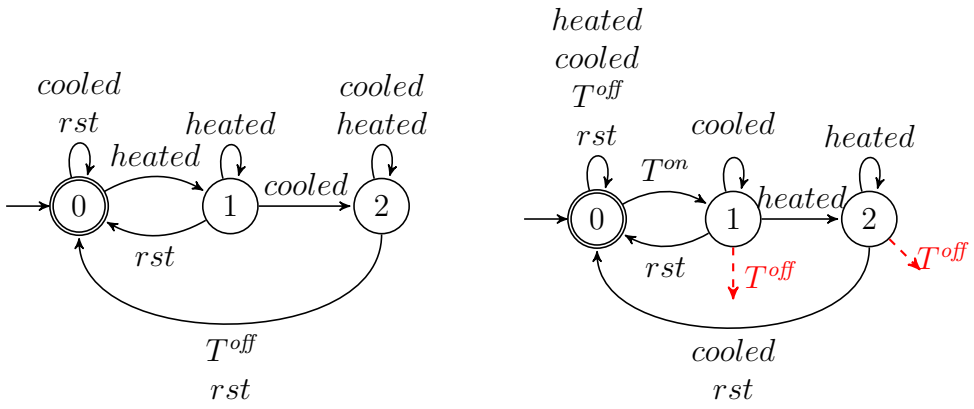
Figure A.12: Specification E_8 and Supervisor S_8



(a) Specification E_9 - Starting of temperature control and input valve

(b) Supervisor S_9

Figure A.13: Specification E_9 and Supervisor S_9



(a) Specification E_{10} - Stopping of temperature control

(b) Supervisor S_{10}

Figure A.14: Specification E_{10} and Supervisor S_{10}

Appendix B

Schematics

The first schematic is the one of the global controller and it is shown in Fig. B.1. For this node, which is based on the Arduino Mega board, only the communication modules were attached to it, with the connections described in the schematic. The Ethernet shield is placed over the Arduino board and requires no additional electrical connections. In the other hand, the CAN module needs to be connected according to Fig. B.1. Notice that the jumper in the CAN module that enables the terminating resistor of 120Ω is connected.

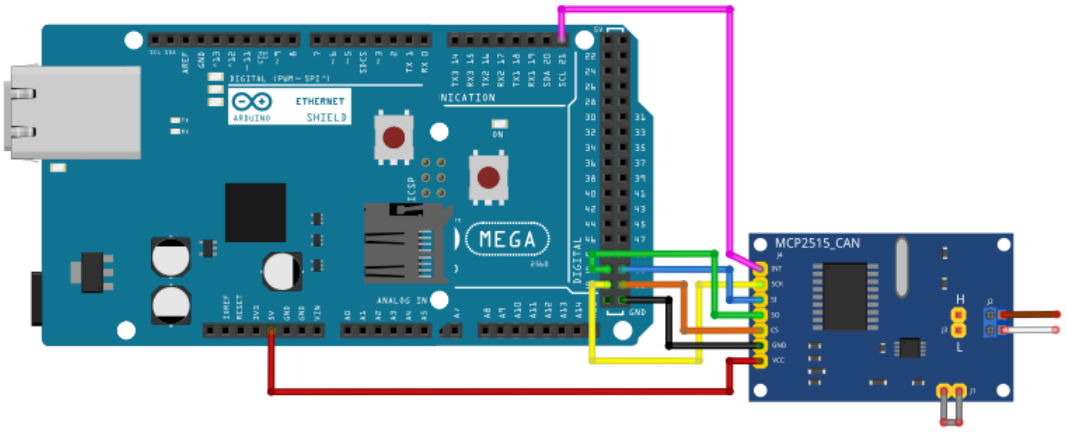


Figure B.1: Schematic of the global controller node.

The next node is shown in Fig. B.2. It is based on an Arduino UNO board and has a CAN module attached to it, in addition to a RC circuit and two LEDs. The RC circuit emulates the dynamic of the liquid temperature inside the tank. The output of the RC circuit is connected to an analog input of the Arduino, while its input is connected to an Arduino output with PWM (pulse width modulation) capability. The values for the capacitor and resistor are $47\mu F$ and $680K\Omega$, respectively, which gives a time constant of $\tau = 31,96s$. Because the time constant is much higher than the period of the PWM signal, which is $2ms$, then the signal applied to the RC circuit can be considered as analog. The two LEDs represent the

input and output valves and each one is connected through a 220Ω resistor.

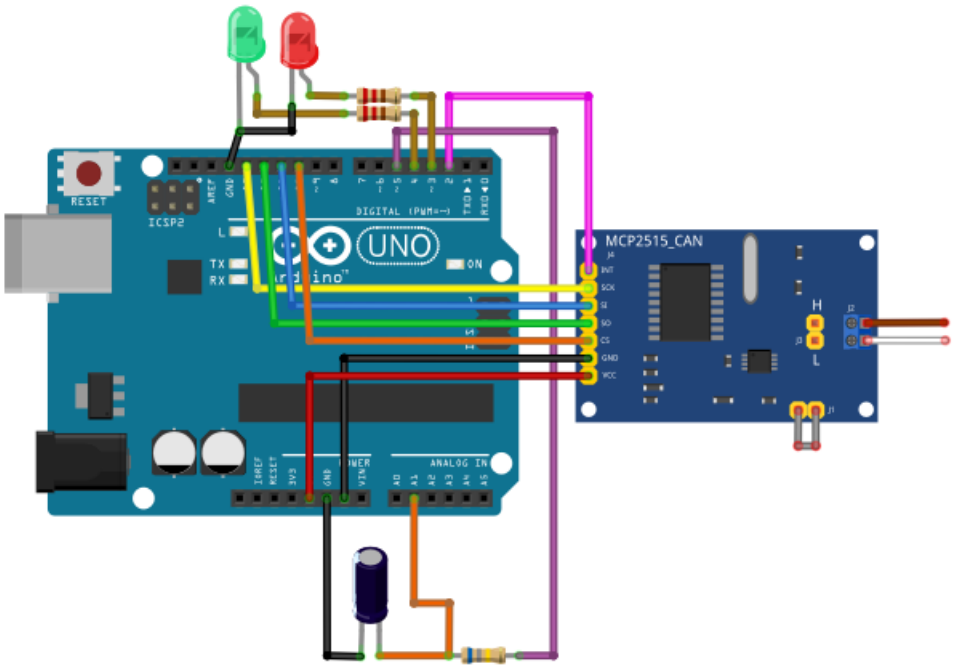


Figure B.2: Schematic of the node related to the temperature control and valves.

Figure B.3 shows the schematic of the node in which the LEDs representing the pump and the mixer are connected to. It is based on an Arduino Nano board. A trimpot allows the user to vary the analog signal sent to one of the analog inputs of the Arduino, representing the level of liquid in the tank. Additionally, an LCD display is connected to the Arduino by using an I2C to LCD converter. The display allows the user to see the current values of continuous-time process variables, such as temperature and level.

Finally, in Fig. B.4, the schematic of a generic node is shown. Is consists only in the Arduino UNO board and the CAN module. This node can be used to run the Intrusion Detection System (IDS) or an attacker, for example.

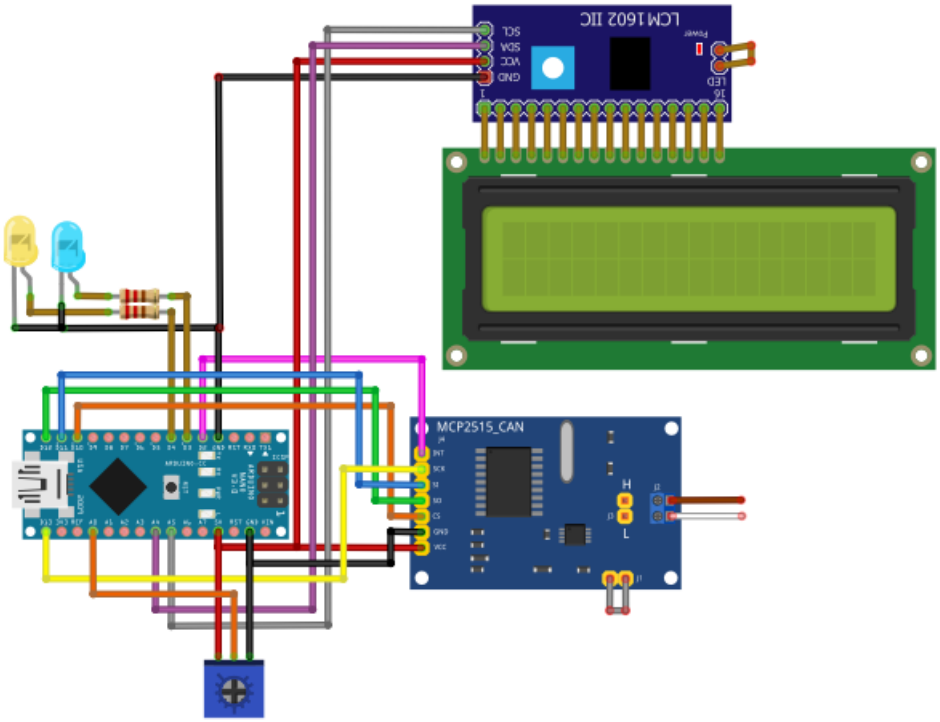


Figure B.3: Schematic of the node related to the level sensor, mixer and pump.

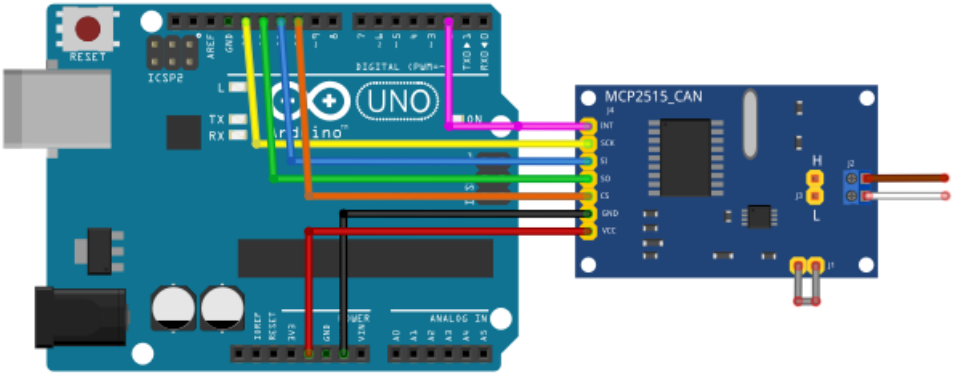


Figure B.4: Schematic of the generic node.