



## SECURITY AGAINST NETWORK ATTACKS IN SUPERVISORY CONTROL SYSTEMS

Públio Macedo Lima

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

Orientadores: Marcos Vicente de Brito  
Moreira  
Lilian Kawakami Carvalho

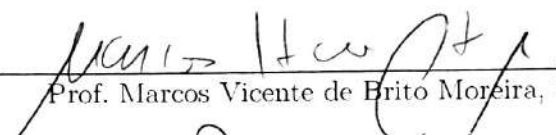
Rio de Janeiro  
Março de 2017

SECURITY AGAINST NETWORK ATTACKS IN SUPERVISORY CONTROL  
SYSTEMS

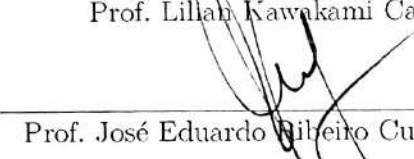
Públio Macedo Lima

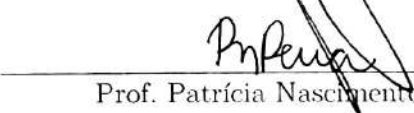
DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO  
ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE  
ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE  
JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A  
OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA  
ELÉTRICA.

Examinada por:

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

  
Prof. Lillian Kawakami Carvalho, D.Sc.

  
Prof. José Eduardo Ribeiro Cury, Docteur d'Etat

  
Prof. Patrícia Nascimento Pena, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2017

Lima, Públio Macedo

Security Against Network Attacks in Supervisory Control Systems/Públio Macedo Lima. – Rio de Janeiro: UFRJ/COPPE, 2017.

X, 54 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira

Lilian Kawakami Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 50 – 54.

1. Security. 2. Cyber-physical system. 3. Discrete event system. 4. Automata. I. Moreira, Marcos Vicente de Brito *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

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

## SEGURANÇA CONTRA ATAQUES CIBERNÉTICOS EM SISTEMAS DE CONTROLE SUPERVISÓRIO

Públio Macedo Lima

Março/2017

Orientadores: Marcos Vicente de Brito Moreira

Lilian Kawakami Carvalho

Programa: Engenharia Elétrica

Sistemas ciberfísicos (SCFs) usam as capacidades de comunicações e de computação atuais para monitorar e controlar processos físicos. Para tal, redes de comunicação são normalmente usadas para conectar os sensores, atuadores e controladores. Com o uso dessas redes de comunicação, a vulnerabilidade de SCFs a ataques cibernéticos aumentam, que podem levar o sistema para estados não seguros. Um tipo de ataque em rede é o “ataque do homem do meio” (*man-in-the-middle attack*), em que um intruso pode observar, esconder, criar e/ou trocar as informações de um canal de comunicação atacado. Neste trabalho é proposta uma estratégia de defesa para detectar e evitar danos causados por “ataques do homem do meio” nos canais de comunicação de sensores e/ou de controle para o sistema de controle supervisório. A definição de controlabilidade segura a ataques na rede (Controlabilidade Segura-AR), que é relacionada com a possibilidade de detectar um ataque a rede e prevenir que o sistema chegue a estados não seguros é proposta. Além disso, é proposto um algoritmo para verificar essa propriedade.

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

## SECURITY AGAINST NETWORK ATTACKS IN SUPERVISORY CONTROL SYSTEMS

Públio Macedo Lima

March/2017

Advisors: Marcos Vicente de Brito Moreira  
Lilian Kawakami Carvalho

Department: Electrical Engineering

Cyber-physical systems (CPSs) integrate computing and communication capabilities to monitor and control physical processes. In order to do so, communication networks are commonly used to connect sensors, actuators, and controllers to monitor and control physical systems. The use of communication networks increases the vulnerability of the CPS to cyber attacks that can drive the system to unsafe states. One of the most powerful cyber attacks is the so-called man-in-the-middle attack, where the intruder can observe, hide, create or change information in the attacked network channel. We propose in this work a defense strategy that detects intrusions and prevents damages caused by man-in-the-middle attacks in the sensor and/or control communication channels in supervisory control systems. We also introduce the definition of NA-Safe controllability, that is related with the possibility of detecting an attack in the network and preventing the reach of unsafe states, and we propose an algorithm to verify this property.

# Contents

|  |             |
|--|-------------|
| <b>List of Figures</b>   | <b>viii</b> |
| <b>Lista de Símbolos</b>   | <b>x</b>    |
| <b>1 Introduction</b>  | <b>1</b>    |
| <b>2 Background</b>  | <b>3</b>    |
| 2.1 Discrete event systems . . . . .                                       | 3           |
| 2.1.1 Languages . . . . .  | 4           |
| 2.1.2 Automata . . . . .   | 6           |
| 2.1.3 Automata language . . . . .  | 7           |
| 2.1.4 Operations with automata . . . . .                                   | 9           |
| 2.2 Supervisory control in DES . . . . .                                   | 12          |
| 2.2.1 Controllability . . . . .  | 12          |
| 2.2.2 Definitions for supervisory control . . . . .                        | 13          |
| 2.2.3 Realization of a Supervisor . . . . .                                | 14          |
| 2.2.4 Control under partial observation . . . . .                          | 15          |
| 2.3 Security in cyber-physical systems . . . . .                           | 19          |
| 2.3.1 Structure of a CPS . . . . .   | 20          |
| 2.3.2 Security in CPS . . . . .  | 20          |
| 2.3.3 Man-in-the-middle attack . . . . .                                   | 21          |
| 2.4 Final comments . . . . .   | 22          |
| <b>3 Security Against Network Attacks in Supervisory Control Systems</b>   | <b>23</b>   |
| 3.1 Model of the plant subject to sensor channel attacks . . . . .         | 25          |
| 3.2 Model of the supervisor subject to supervisory control channel attacks | 28          |
| 3.3 Model of the closed-loop system subject to network attacks . . . . .   | 30          |
| 3.4 NA-Safe Controllability . . . . .                                      | 31          |
| 3.4.1 Verification of NA-Safe controllability . . . . .                    | 32          |
| 3.5 Implementation of the Intrusion Detection<br>Module . . . . .          | 37          |
| 3.6 Example . . . . .  | 43          |

|                                     |           |
|-------------------------------------|-----------|
| <b>4 Conclusion and Future work</b> | <b>48</b> |
| <b>Bibliography</b>                 | <b>50</b> |

# List of Figures

|      |   |    |
|------|---|----|
| 2.1  | Storage unit . . . . .  | 4  |
| 2.2  | State transition diagram of automaton $G$ of Example 7. . . . .             | 6  |
| 2.3  | Example of a nondeterministic automaton $\mathcal{G}$ of Example 8. . . . . | 7  |
| 2.4  | Automaton $G$ of Example 9. . . . .   | 8  |
| 2.5  | Automaton $G$ of Example 10 . . . . .                                       | 9  |
| 2.6  | Accessible part of automaton $G$ of Example 10. . . . .                     | 10 |
| 2.7  | Automaton $G$ of Example 11. . . . .  | 10 |
| 2.8  | Coaccessible part of automaton $G$ of Example 11. . . . .                   | 11 |
| 2.9  | Automata $G_1$ and $G_2$ of Example 12. . . . .                             | 12 |
| 2.10 | Parallel composition of $G_1$ and $G_2$ of Example 12. . . . .              | 12 |
| 2.11 | Feedback control $S/G$ . . . . .  | 13 |
| 2.12 | Automaton $G$ of Example 13 . . . . .                                       | 14 |
| 2.13 | Supervisor realization $H$ of Example 13. . . . .                           | 15 |
| 2.14 | Closed-loop system $T = G \parallel H$ of Example 13. . . . .               | 15 |
| 2.15 | Automaton $G$ of Example 14. . . . .  | 16 |
| 2.16 | Automaton $G_{obs}$ of Example 14. . . . .                                  | 17 |
| 2.17 | Closed-loop system under partial observation . . . . .                      | 17 |
| 2.18 | Example of automaton $G$ and supervisor $H$ of Example 15. . . . .          | 18 |
| 2.19 | Closed-loop system under partial observation $T$ of Example 15. . . . .     | 18 |
| 2.20 | Example of structure for CPS . . . . .                                      | 20 |
|      |   |    |
| 3.1  | Closed-loop system under attack . . . . .                                   | 24 |
| 3.2  | Intrusion detection module . . . . .  | 25 |
| 3.3  | Plant model . . . . .   | 26 |
| 3.4  | Attacked plant model . . . . .  | 27 |
| 3.5  | Supervisor . . . . .  | 29 |
| 3.6  | Realization of the attacked Supervisor. . . . .                             | 30 |
| 3.7  | Controlled System . . . . .   | 31 |
| 3.8  | Attacked controlled system . . . . .  | 31 |
| 3.9  | Safe part of $T_A$ . . . . .  | 35 |
| 3.10 | Unsafe part of $T_A$ . . . . .  | 35 |



|      |  |    |
|------|--|----|
| 3.11 | Renamed unsafe part of $T_A$ . . . . .   | 36 |
| 3.12 | Verifier . . . . .   | 36 |
| 3.13 | Automaton $G$ of Example 20 . . . . .  | 38 |
| 3.14 | Supervisor automaton $H$ of Example 20. . . . .                                      | 38 |
| 3.15 | Automaton $G$ subject to sensor attack . . . . .                                     | 39 |
| 3.16 | Closed-loop system subject to attack $T_a$ . . . . .                                 | 39 |
| 3.17 | Automata $G$ and $H$ of Example 21. . . . .  | 41 |
| 3.18 | Automata $\tilde{G}$ and $\tilde{H}$ of Example 21. . . . .                          | 42 |
| 3.19 | Initial house position for cat/mouse example. . . . .                                | 44 |
| 3.20 | Automaton model for mouse/cat problem $G$ . . . . .                                  | 44 |
| 3.21 | Supervisor $H$ for cat/mouse example . . . . .                                       | 45 |
| 3.22 | Similar plant system $\tilde{G}$ from $G$ in cat/mouse example. . . . .              | 45 |
| 3.23 | Similar supervisor system $\tilde{H}$ from $H$ in cat/mouse example. . . . .         | 45 |
| 3.24 | Closed-loop system $T$ for cat/mouse example. . . . .                                | 46 |
| 3.25 | Plant model under attack $\mathcal{G}_A$ for cat/mouse example. . . . .              | 46 |
| 3.26 | Supervisor model under attack $\mathcal{H}_A$ for cat/mouse example. . . . .         | 46 |
| 3.27 | Closed-loop system model under attack $\mathcal{T}_A$ for cat/mouse example. . . . . | 46 |
| 3.28 | Closed-loop system model unsafe part $\mathcal{T}_U$ for cat/mouse example. . . . .  | 46 |
| 3.29 | Renamed model of unsafe part $\mathcal{T}_{U,R}$ for cat/mouse example. . . . .      | 47 |
| 3.30 | Verifier model $\mathcal{V}$ for cat/mouse example . . . . .                         | 47 |
| 3.31 | Observer of safe part for cat/mouse example. . . . .                                 | 47 |

# Lista de Símbolos

|                     |  |
|---------------------|--|
| $Ac(G)$             | Accessible part of automaton $G$ , p. 9                                  |
| $CoAc(G)$           | Coaccessible part of automaton $G$ , p. 10                               |
| $G$                 | Model of the plant by deterministic automaton, p. 23                     |
| $G_1 \parallel G_2$ | Parallel composition between automata $G_1$ and $G_2$ , p. 11            |
| $G_{obs}$           | Observer of automaton $G$ with unobservable events $\Sigma_{uo}$ , p. 16 |
| $H$                 | Automaton realization of the supervisor, p. 14                           |
| $L(G)$              | Generated language by automaton $G$ , p. 8                               |
| $L/s$               | Post language of $L$ after $s$ , p. 5                                    |
| $L_m(G)$            | Marked language of automaton $G$ , p. 8                                  |
| $T$                 | Automaton realization of the closed-loop system $S/G$ , p. 14            |
| $CoAc(Ac(G))$       | Coaccessible part of automaton $G$ , p. 11                               |
| $UR(x)$             | Unobservable reach of a state $x$ , p. 15                                |
| $\Sigma^*$          | Kleene-closure of $\Sigma$ , p. 5  |
| $\Sigma_s \in s$    | At least one of the events that form $s$ belongs to $\Sigma_s$ , p. 5    |
| $ s $               | The length of a sequence $s$ , p. 4                                      |
| $\bar{L}$           | Prefix-closure of $L$ , p. 5   |
| $\varepsilon R(x)$  | $\varepsilon$ - reach of state $x$ , p. 8                                |
| $\varepsilon$       | Empty sequence, p. 4   |
| $S/G$               | Supervisor $S$ controlling plant $G$ ., p. 12                            |

# Chapter 1

## Introduction

Cyber-physical systems (CPSs) integrate computing and communication capabilities to monitor and control physical processes [1–3]. In order to do so, communication networks are commonly used to connect sensors, actuators, and controllers in the feedback system.

The increase in the use of communication networks for monitoring and controlling of physical systems also increases the vulnerability of CPSs to attacks in the network. Several works in the literature propose strategies to detect and prevent the effects of cyber attacks considering different approaches [4–12]. The majority of the works proposed in the literature model the system as a continuous variable dynamic system, and address the problem of stealthy deception attacks that interfere with the system state estimation.

Intrusion detection and prevention of damages caused by attacks are considered in THORSLEY and TENEKETZIS [13] in the context of supervisory control of discrete event systems. The main objective of the work proposed in [13] is to design a supervisor that achieves the specification in normal operation and after an attack. The attacks considered in [13] can be interpreted as an interference in the communication channel between the supervisor and the system that could be caused by an intruder attempting to damage the system. In [13], the attacker is able to completely change the set of enabled events commanded by the supervisor.

Recently, in CARVALHO *et al.* [14], the problem of intrusion detection and prevention in supervisory control systems, where the attacker has the ability to enable vulnerable actuator events that are disabled by the supervisor, is addressed. A mathematical model for the system under such actuator enablement attacks is obtained, and a defense strategy that detects attacks online and disables all controllable events after an attack is detected is proposed.

The attacks considered in [13] and [14] are examples of possible network attacks in feedback systems. Indeed, there are several types of network attacks, as shown in [15]. A well-known type of attack in communication networks is the so-called

man-in-the-middle attack [16]. In this type of attack, the intruder can observe, hide, create or even change information that transits from one device to another in a communication channel. Thus, once the intruder has attacked a sensor and/or a control communication channel in a supervisory control system, it can lead the plant to execute traces with the objective to reach unsafe states that can damage the system. Although this type of attack is one of the most difficult to be executed by the intruder, its potential to damage the system is very high since the attacker has freedom to change completely the information that transits in the attacked network channel.

In this work, we propose a defense strategy that detects intrusions and prevents damages caused by man-in-the-middle attacks in the sensor and/or control communication channels in supervisory control systems, which includes the results in LIMA *et al.* [17]. In order to do so, automaton models of the plant under sensor channel attacks and of the supervisor under control channel attacks are obtained. Then, the definition of safe controllability under network attacks, called NA-Safe controllability, that is related with the possibility of detecting an attack in the network and preventing the reach of unsafe states, is presented. We also propose an algorithm to verify this property. Finally, the computation of a device that detects the attacks that lead to unsafe states, called in this work as Intrusion Detection Module, is proposed. It is important to remark that, differently from [14] and [13], that only consider attacks in the supervisory control communication channel, we also consider attacks in the vulnerable sensor communication channels.

A problem in discrete event systems with wide repercussion in the literature is the problem of failure diagnosis [18–30]. The detection of an attack presented in this work may be related to this problem. However, the detection of an attack is slightly different from the detection of a failure, since the objective in this work is to allow a system to operate even after an attack is detected as long as this operation does not cause the system any harm, *i.e.*, the objective of this work is to prevent the system from reaching an unsafe state, instead of just indicating when an unobservable event has occurred.

The remainder of this work is structured as follows. In Chapter 2 we present some fundamentals on DES and supervisory control, and then also present a basic background over security in network communication systems. In Chapter 3 we propose a model for attacked CPSs and propose the Implementation of the Intrusion Detection Module, most of the results presented in this chapter has been summarized in a paper [17] accepted for presentation on an international conference. In Chapter 4 we summarize our contributions in this work and present some possibles future works.

# Chapter 2

## Background

In this chapter we present the background needed to understand this work. In order to do so, we divided this chapter into three sections: in Section 2.1 we present the theoretical background of discrete event systems (DES), including the automaton model of DES; and in Section 2.2, we introduce the basic ideas of the supervisory control theory; in Section 2.3 we introduce the structure and security of CPSs, and present one of the main attacks in network systems.

### 2.1 Discrete event systems

DESs are dynamic systems in which the set of states is a discrete set, and the state transitions are only observed at discrete points in time associated with the occurrence of “events” [31]. We call an event as an instantaneous occurrence that causes the system to transit from a state to another. The event can be associated with a specific action, such as a button that is pressed or a sensor that transits from low to a high value, or any other action or condition that makes the system change its state.

In opposition to continuous-state systems, the evolution in DESs are not necessarily associated with time. As a consequence, we cannot represent these systems by differential equations, such as in the case of continuous time system, or difference equations, in the case of discrete time system. Thus, we need a different formalism to model and work with this type of system.

**Example 1** *Let us consider a simple storage unit, represented in Figure 2.1, that can receive only one truck at a time, and let us consider that the truck can either remove or drop a box in the unit. Assume also that the storage unit has an infinity capacity, i.e., there is no maximum number of boxes that can be in the storage unit at the same time. Notice that representing this system by a time-driven system is a difficult task since there does not exist rules on the arrival of trucks and, therefore,*

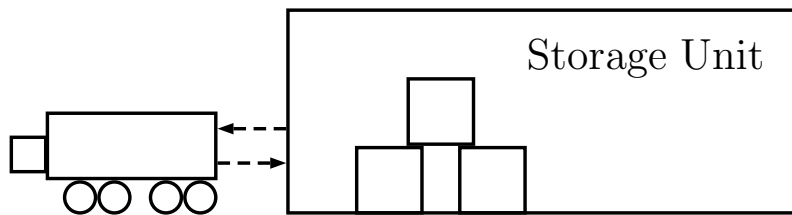


Figure 2.1: Storage unit

there is no way of determining when a truck will arrive. As a DES on the other hand, we can assume that the state is the number of boxes in the unit, and consider two events, a truck drops a box on the unit, which increases the value of the state by 1, or a truck removes a box from the unit, which decreases the value of the state by 1.

Discrete event systems can be modeled by using different formalisms, such as Petri nets and automata. In this work, systems are modeled by automata. In this regard, this section is organized as follows. In Subsection 2.1.1 we introduce the notion of languages of a system and some operations with languages, and in Subsection 2.1.2 we present the model of a deterministic and of a nondeterministic automaton. We also present the language of these automata, and some basic operations using them.

### 2.1.1 Languages

A formal way to study DESs is using the theory of languages. DESs are associated with an event set  $\Sigma$ , that can be seen as an “alphabet”. The concatenation of events form sequences that can be seen as “words” of a language (notice that in the literature a sequence is also called “string” or “trace”), and a language of a system is a set of sequences that the system can execute. The length of a sequence is the number of events that form it, counting multiple occurrences of the same event, and it is denoted by  $|s|$ . The sequence with zero length is called empty sequence and is denoted by  $\varepsilon$ .

**Definition 1 (Language)** [31]

A language defined over an event set  $\Sigma$  is a set of finite length sequences formed with events in  $\Sigma$ . □

**Example 2** Let  $\Sigma = \{a, b\}$ . Then,  $L = \{\varepsilon, a, ab, aba, bbaa\}$  is a language defined over  $\Sigma$ , where the length of sequence “bbaa” is  $|bbaa| = 4$ .

Let us denote by  $\Sigma^*$  the Kleene-closure of the set of events  $\Sigma$ , which consists of all finite length sequences that can be formed using elements of  $\Sigma$  including the empty sequence  $\varepsilon$ .

**Example 3** Let  $\Sigma = \{a, b\}$ . The Kleene-closure of  $\Sigma$  is given by  $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ . It is important to remark that since  $L$  is defined over  $\Sigma$ , then  $L \subseteq \Sigma^*$ .

A sequence  $s$  can be partitioned as  $s = tuv$ , where  $t$  is the prefix of  $s$ , and  $v$  is its suffix. Notice that  $\varepsilon$  is always a prefix and a suffix of any sequence.

**Example 4** Let sequence  $s = a$ . Then it can be partitioned as  $a = \varepsilon a \varepsilon$  where  $\varepsilon$  is a prefix and a suffix of  $s$ .

The prefix closure of a language  $L$ , denoted by  $\bar{L}$ , is the set formed with all prefixes of all sequences of  $L$ . Formally  $\bar{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}$ .

**Example 5** Let  $L = \{abc, bb\}$ . The prefix-closure of  $L$  is given by  $\bar{L} = \{\varepsilon, a, ab, abc, b, bb\}$ .

We can also define the post language after a trace  $s$  in a language  $L$ ,  $L/s$ , as the set formed with the continuation of all sequences of  $L$  after  $s$ , i.e.,  $L/s = \{t \in \Sigma^* : st \in L\}$ . For example, let  $L = \{ab, abb, acb, bac\}$ , then the post language after sequence  $a$  in the language is  $L/a = \{b, bb, cb\}$ .

Let  $\Sigma_s \subset \Sigma$ , and let  $s \in \Sigma^*$ . Then, with a slight abuse of notation we denote by  $\Sigma_s \in s$  if at least one of the events that form  $s$  belongs to  $\Sigma_s$ .

Another important operation on sequences and languages is the natural projection operation. Let  $\Sigma_o \subset \Sigma$  be a set of events. Then, the projection  $P : \Sigma^* \rightarrow \Sigma_o^*$ , is defined as  $P(\varepsilon) = \varepsilon$ ,  $P(\sigma) = \sigma$ , if  $\sigma \in \Sigma_o$  or  $P(\sigma) = \varepsilon$ , if  $\sigma \in \Sigma \setminus \Sigma_o$ , and  $P(s\sigma) = P(s)P(\sigma)$ , for all  $s \in \Sigma^*$  and  $\sigma \in \Sigma$ . Notice that the projection  $P(s)$  erases events that do not belong to  $\Sigma_o$  in  $s \in \Sigma^*$ . The projection operation can be extended to languages by applying them to all sequences in the language [31],  $P(L) = \{t \in \Sigma_o^* : (\exists s \in L)[P(s) = t]\}$ .

**Example 6** If we consider a language  $L = \{a, abc, bbaa, cca, cba\}$  that belongs to  $\Sigma^*$ , where  $\Sigma = \{a, b, c\}$ , and consider a set  $\Sigma_o = \{b, c\} \subset \Sigma$ , the projection  $P : \Sigma^* \rightarrow \Sigma_o^*$  applied to language  $L$  is the projection of each sequence in  $L$ , i.e.,  $P(L) = \{\varepsilon, bc, bb, cc, cb\}$ .

The inverse projection denoted by  $P^{-1} : \Sigma_o^* \rightarrow 2^{\Sigma^*}$  is defined as  $P^{-1}(s) = \{t \in \Sigma^* : [P(t) = s]\}$ . The inverse projection can also be applied to a language  $L \subseteq \Sigma^*$  by applying it to all sequences in  $L$  [31],  $P^{-1}(L) = \{s \in \Sigma^* : (\exists t \in L)[P(s) = t]\}$ .

A language represents a DES and specifies all sequences that can be executed in this system, however languages are not always easy to specify or work with and, therefore, we need another formalism to represent DESs. Another way to model a DES is by using automata, whose mathematical and graphical representation will be detailed in Subsection 2.1.2.

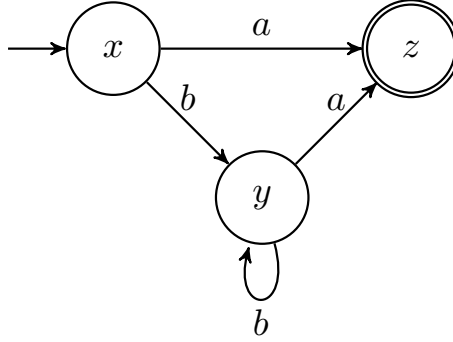


Figure 2.2: State transition diagram of automaton  $G$  of Example 7.

## 2.1.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules [31]. In this subsection, we first formally introduce the definition of deterministic and nondeterministic automata, and then relate them to languages in order to represent DESs. Finally, we present some important operations with automata.

### Deterministic automata

Formally, a deterministic automaton is represented by a five-tuple,  $G = (X, \Sigma, f, x_0, X_m)$ , where  $X$  is the state-space,  $\Sigma$  is the finite set of events,  $f : X \times \Sigma \rightarrow X$  is the transition function,  $x_0 \in X$  is the initial state of the system, and  $X_m \subseteq X$  is the set of marked states, *i.e.*, states that, for some reason, are important in the system. We can also define  $\Gamma_G : X \rightarrow 2^\Sigma$  as the active event function in a state of  $G$ . Notice that the set of marked states may be omitted, *i.e.*, the automaton may be represented by  $G = (X, \Sigma, f, x_0)$  which implies that the set of marked states is  $X_m = \emptyset$ .

**Example 7** *Let us consider automaton  $G$  where  $X = \{x, y, z\}$ ,  $\Sigma = \{a, b\}$ , the transition function  $f$  is given by  $f(x, a) = y$ ,  $f(x, b) = z$ ,  $f(y, a) = z$  and  $f(y, b) = y$ ,  $z$  is a marked state, *i.e.*,  $X_m = \{z\}$ , and the initial state is  $x_0 = x$ . The graphical representation of this automaton is shown in Figure 2.2, where every state is represented by a circle with its name on it, all transitions are represented by an arrow that goes from the origin state  $x$  to state  $f(x, \sigma)$  labeled with event  $\sigma$ , the marked state is represented by a double circle, and the initial state have an arrow on it.*

In a deterministic automaton the transition function is unique for each state and event, *i.e.*, there do not exist two transitions from the same state reaching different states labelled with the same event. Another characteristic of deterministic automata is that all transitions have an event label, which means that the system cannot evolve with  $\varepsilon$ .



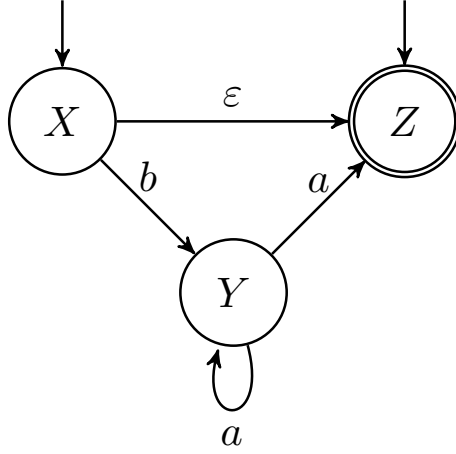


Figure 2.3: Example of a nondeterministic automaton  $\mathcal{G}$  of Example 8.

The transition function  $f$  can be extended recursively to domain  $X \times \Sigma^*$  in order to consider a sequence  $t\sigma$  by making  $f(x, \varepsilon) = x$ , and  $f(x, t\sigma) = f(f(x, t), \sigma)$ .

### Nondeterministic automata

A nondeterministic automaton is also a five-tuple  $\mathcal{G} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, x_0, X_m)$ , and it has the same interpretation as a deterministic automaton, except for  $f_{nd}$  that is now a nondeterministic transition function  $f_{nd} : X \times \{\varepsilon\} \rightarrow 2^X$ , i.e., the transition function can now evolve to a subset of  $X$ , and the initial state  $x_0$  may now be a subset of  $X$ . Moreover, a transition may be labeled with  $\varepsilon$  in a nondeterministic automaton. We can also define  $\Gamma_{\mathcal{G}}$  as the set of active events in a state of  $\mathcal{G}$ , and in the case of nondeterministic it may be extended to consider a set of states  $\Gamma(B) = \bigcup_{x \in B} \Gamma(x)$ .  $X_m$  may also be omitted as in deterministic automata.

**Example 8** *In order to illustrate a nondeterministic automaton let us consider the system represented in Figure 2.3, where the initial set of states is given by  $x_0 = \{X, Z\}$ , and the transition function is given by  $f_{nd}(X, b) = \{Y\}$ ,  $f_{nd}(X, \varepsilon) = \{Z\}$  and  $f_{nd}(Y, a) = \{Y, Z\}$ .*

### 2.1.3 Automata language

A language is a formal way to describe a DES. As an automaton also represent a DES it can be related with languages, more specifically each automaton is related with two languages, the generated language and the marked language. This connection is made by observing the state transition diagram of an automaton considering its extended transition function  $f : X \times \Sigma^* \rightarrow X$ .

**Definition 2** *The language generated by a deterministic automaton  $G = (X, \Sigma, f, x_0, X_m)$*

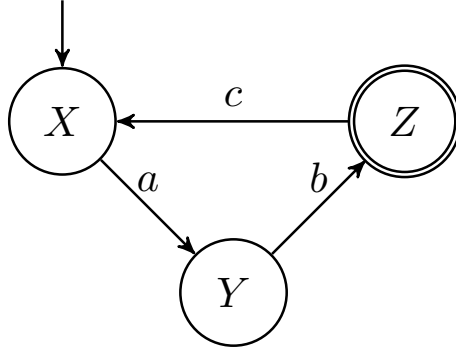


Figure 2.4: Automaton  $G$  of Example 9.

is:

$$L(G) := \{s \in \Sigma^* : f(x_0, s) \text{ is defined} \},$$

and the marked language of automaton  $G$  is given by:

$$L_m(G) := \{s \in \Sigma^* : f(x_0, s) \in X_m\}.$$

**Example 9** The generated and marked languages of automaton  $G$ , shown in Figure 2.4, are, respectively,  $L(G) = \{\varepsilon, a, ab, abc, abca, abcab, \dots\}$  and  $L_m(G) = \{ab, abcab, abcabcab, \dots\}$ .

Similarly, for nondeterministic automata the generated and marked languages are obtained from the extended transition function  $f_{nd}$ . In order to characterize these languages it is first needed to extend the transition function to domain  $X \times \Sigma^*$ . In opposition to deterministic automaton, where  $f(x, \varepsilon) = x$ , in a nondeterministic automaton we start defining the  $\varepsilon$ -reach of a state  $x$ , denoted by  $\varepsilon R(x)$ , which is the set of all states, including  $x$ , that are reached from state  $x$  by following transitions labeled with  $\varepsilon$ . This function can be extended to consider a set of states  $B \subseteq X$ , and it is defined by  $\varepsilon R(B) := \cup_{x \in B} \varepsilon R(x)$

Then, we can extend the transition function for nondeterministic automata recursively as follows:

$$f_{nd}^{ext}(x, \varepsilon) := \varepsilon R(x),$$

and for  $u \in \Sigma^*$ , and  $e \in \Sigma$ :

$$f_{nd}^{ext}(x, ue) := \varepsilon R(z : z \in f_{nd}(y, e) \text{ for some state } y \in f_{nd}^{ext}(x, u)).$$

**Definition 3** The language generated by a nondeterministic automaton  $\mathcal{G} = (X, \Sigma \cup$

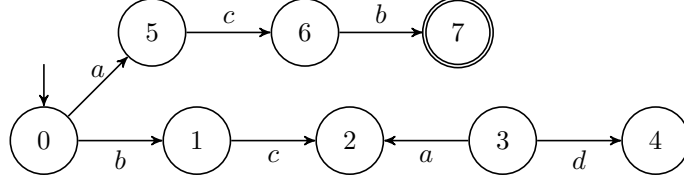


Figure 2.5: Automaton  $G$  of Example 10

$\{\varepsilon\}, f_{nd}, x_0, X_m)$  is:

$$L(\mathcal{G}) := \{s \in \Sigma^* : f_{nd}^{ext}(x_0, s) \text{ is defined } \},$$

and the marked language of automaton  $G$  is given by:

$$L_m(\mathcal{G}) := \{s \in \Sigma^* : f_{nd}^{ext}(x_0, s) \cap X_m \neq \emptyset\}.$$

## 2.1.4 Operations with automata

We need a set of operations in order to properly work with DESs modeled by automata. In this regard, we define first, in this subsection, unary operations, and then compositions operations in order to modify the state transition diagrams of these automata.

### Accessible part

The operation of taking the accessible part of an automaton  $G$  is denoted as  $Ac(G)$ , and its state set is formed by all states that can be reached after the occurrence of a sequence  $s \in L(G)$  from the initial state  $x_0$ , *i.e.*:

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

where

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

$$X_{m,ac} = X_m \cap X_{ac},$$

$$f_{ac} = f |_{X_{ac} \times \Sigma \rightarrow X_{ac}},$$

where  $f |_{X_{ac} \times \Sigma \rightarrow X_{ac}}$  denotes the transition function  $f$  restricted to domain  $X_{ac} \times \Sigma$ .

**Example 10** *In order to illustrate the operation of taking the accessible part of an automaton, consider automaton  $G_3$ , represented in Figure 2.5.*

*Notice that there is no sequence from the initial state that leads to states 3 and 4. Thus, the state transition diagram of the accessible part of automaton  $G$  is given*

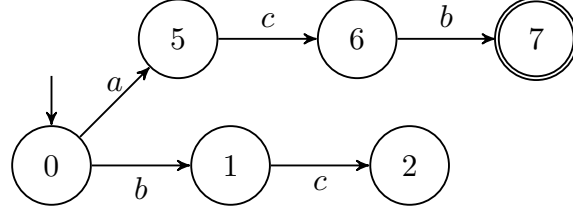


Figure 2.6: Accessible part of automaton  $G$  of Example 10.

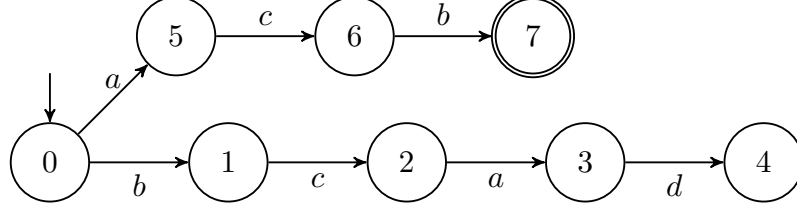


Figure 2.7: Automaton  $G$  of Example 11.

in Figure 2.6.

### Coaccessible part

The coaccessible part of an automaton  $G$  is denoted as  $CoAc(G)$ , and its state set is formed by all states from which there is a sequence that leads the system to a marked state. Formally,

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

where,

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

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

$$f_{coac} = f \upharpoonright_{X_{coac} \times \Sigma^* \rightarrow X_{coac}},$$

where  $f \upharpoonright_{X_{coac} \times \Sigma^* \rightarrow X_{coac}}$  denotes the transition function  $f$  restricted to domain  $X_{coac} \times \Sigma^*$

**Example 11** In order to illustrate the operation of taking the coaccessible part let us consider automaton  $G$  shown in Figure 2.7.

From automaton  $G$  we can find the states from which it is possible to reach a marked state. Notice that from states  $\{1, 2, 3, 4\}$  it is not possible to reach the marked state 7. Therefore,  $CoAc(G)$  is represented by the state transition diagram shown in Figure 2.8.

**Remark 1** It is important to remark that in the operations of taking the accessible and the coaccessible part of an automaton  $G = (X, \Sigma, f, x_0, X_m)$  the resulting

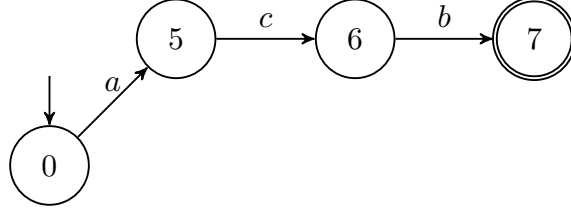


Figure 2.8: Coaccessible part of automaton  $G$  of Example 11.

automaton has  $\Sigma$  as its event set.

**Remark 2** The operation of taking the accessible part and the coaccessible part of an automaton  $G$ , is denoted by  $Trim(G)$ , i.e.,  $Trim(G) = Ac(CoAc(G)) = CoAc(Ac(G))$ . An automaton  $G$  is called a trim automaton if  $G = Trim(G)$ .

### Parallel composition

In the literature two compositions of automata are defined, the parallel composition or synchronous composition, and the product composition or completely synchronous composition. As we only use the parallel composition in this work there is no need to present the product operation.

The parallel composition is represented by  $\parallel$ , and thus  $G_1 \parallel G_2$  denotes the parallel composition of automata  $G_1$  and  $G_2$ . In the parallel composition, a common event between two automata can only occur if it is active in both automata simultaneously. The parallel composition of automata  $G_1 = (X_1, \Sigma_1, f_1, x_{01}, X_{m1})$  and  $G_2 = (X_2, \Sigma_2, f_2, x_{02}, X_{m2})$  is given by:

$$G_1 \parallel G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\parallel 2}, (x_0^1, x_0^2), X_m^1 \times X_m^2)$$

where

$$f_{1\parallel 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_{G_1}(x_1) \cap \Gamma_{G_2}(x_2) \\ (f_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Gamma_{G_1}(x_1) \setminus \Sigma_2 \\ (x_1, f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_{G_2}(x_2) \setminus \Sigma_1 \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

**Example 12** Let us consider two automata  $G_1$  and  $G_2$  represented in Figure 2.9 (a) and (b), respectively, where  $\Sigma_1 = \{a\}$  and  $\Sigma_2 = \{a, b\}$ .

The parallel composition of  $G_1$  and  $G_2$  is shown in Figure 2.10. Notice that the event set of  $G_1 \parallel G_2$  is given by  $\Sigma_1 \cup \Sigma_2 = \{a, b\}$ .

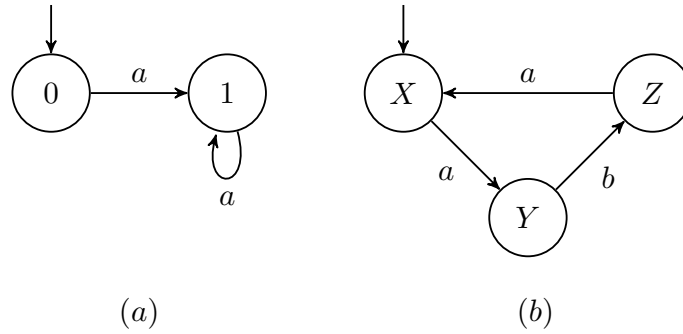


Figure 2.9: Automata  $G_1$  and  $G_2$  of Example 12.

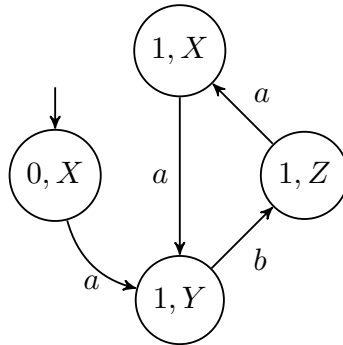


Figure 2.10: Parallel composition of  $G_1$  and  $G_2$  of Example 12.

## 2.2 Supervisory control in DES

The automaton model of a DES presented in Section 2.1 may represent the behavior of an uncontrolled plant. However, if a specification is given, the plant must be controlled by using a feedback control in order to achieve this specification. In order to do so a supervisor can be implemented for the system [31, 32].

A supervisory control system limits the behavior of the plant, *i.e.*, the language generated by the closed-loop system is a subset of  $L(G)$ , in order to reach a certain specification. The language  $L(G)$  may contain sequences that violate some conditions, then a specification is defined in order to avoid such sequences.

In this section, we introduce the supervisor, denoted by  $S$ . The system  $G$  controlled by  $S$  is represented in Figure 2.11. The feedback control system is denoted by  $S/G$ , which is read as  $S$  controlling  $G$ .

### 2.2.1 Controllability

#### Controllable events

Let us assume that the set of events  $\Sigma$  is partitioned as  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ , where  $\Sigma_c$  and  $\Sigma_{uc}$  denote, respectively, the sets of controllable and uncontrollable events of the system. The controllable events of the system are the events that can be disabled

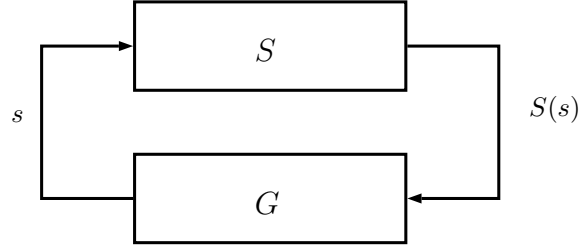


Figure 2.11: Supervisor  $S$  controlling the plant  $G$ .

or prevented from happening by the supervisor, while the uncontrollable events are events that cannot be prevented from happening.

The controllability condition is intuitive and reflects an important notion in supervisory control [31].

**Definition 4 (Controllability)** *Let  $K$  and  $M = \overline{M}$  be languages defined over event set  $\Sigma$ . Let  $\Sigma_{uc}$  be a subset of  $\Sigma$ . Then,  $K$  is said to be controllable with respect to  $M$  and  $\Sigma_{uc}$  if*

$$K\Sigma_{uc} \cap M \subseteq K.$$

The concept of controllability can be understood as, any sequence that cannot be prevented from happening must be legal. In other words, the language expression for the controllability condition can be rewritten as:

$$\forall s \in \overline{K}, \forall e \in \Sigma_{uc}, se \in M \Rightarrow se \in \overline{K}.$$

### Controllability Theorem

The key result for the existence of a supervisor in the presence of uncontrollable events is called controllability theorem, and is presented in the sequel [31].

**Theorem 1 (Controllability Theorem)** *Consider  $G = (X, \Sigma, f, x_0, X_m)$  and the set of uncontrollable events  $\Sigma_{uc} \subseteq \Sigma$ . Let  $K \subseteq L(G)$ , where  $K \neq \emptyset$ . Then, there exists a supervisor  $S$  such that  $L(S/G) = \overline{K}$  if, and only if,*

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

## 2.2.2 Definitions for supervisory control

### The supervisory function

Mathematically, the supervisor is a function  $S : L(G) \rightarrow 2^\Sigma$ , in which controllable events of  $G$  can be disabled or enabled by  $S$ . So, for each  $s \in L(G)$  the set of enabled events is given by  $S(s) \cap \Gamma_G(s)$ . Notice that a supervisor  $S$  cannot disable

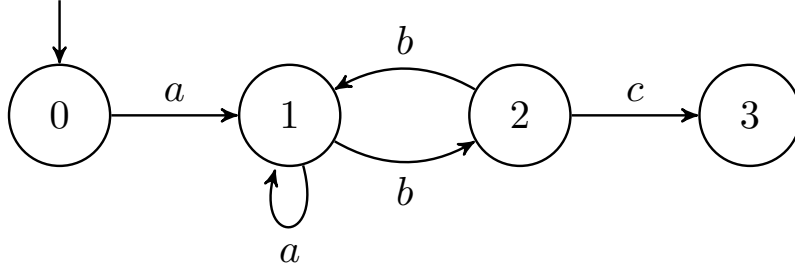


Figure 2.12: Automaton  $G$  of Example 13

uncontrollable events. The supervisor is said to be admissible if for all  $s \in L(G)$ ,  $\Sigma_{uc} \cap \Gamma_G(f(x_0, s)) \subseteq S(s)$ . In this work, we only consider admissible supervisors.

### Languages of a supervisor

After defining the supervisor function we can also define the languages that  $G$  generates and marks being controlled by  $S$  that is  $L(S/G)$  and  $L_m(S/G)$ .

The generated language by  $S/G$  is defined recursively as:

$$1: \varepsilon \in L(S/G)$$

$$2: [(s \in L(S/G)) \text{ and } (s\sigma \in L(G)) \text{ and } (\sigma \in S(s))] \Leftrightarrow [s\sigma \in L(S/G)]$$

The marked language is defined as:

$$L_m(S/G) := L(S/G) \cap L_m(G).$$

### 2.2.3 Realization of a Supervisor

After defining the languages of  $S/G$  we can define an automaton in order to represent the whole closed-loop system. In this work the closed-loop system is called  $T$ . Therefore, it is easier if we define a realization  $H$  of the function  $S$  that when composed with  $G$  generates automaton  $T$ , *i.e.*,  $T = G \parallel H$ . The supervisor will then disable controllable events of  $G$  generating the automaton of the closed-loop system.

**Example 13** Consider automaton  $G$  where state transition diagram shown in Figure 2.12, and suppose that the specification is to avoid the occurrence of event  $c$ . In order to do so, the supervisor function  $S$  may allow the occurrence of all events except event  $c$ . Function  $S$  will then be  $S(s) = \{a, b\}$ , where  $s$  is any sequence in  $\{a, b\}^*$ .

A realization of this supervisor denoted here by  $H$  is presented in Figure 2.13.

Notice that the parallel composition between  $G$  and  $H$  results in the automaton  $T$  that generates the specified language represented in Figure 2.14.



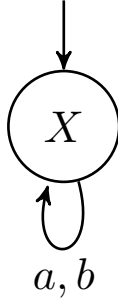


Figure 2.13: Supervisor realization  $H$  of Example 13.

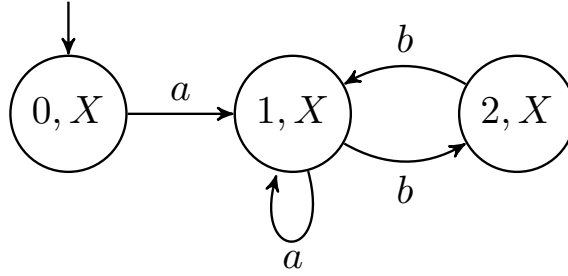


Figure 2.14: Closed-loop system  $T = G \parallel H$  of Example 13.

## 2.2.4 Control under partial observation

### Observable events

Notice that, in real systems, there may exist events whose occurrence are not communicated to the supervisor of the plant. In these cases, the set of events can be partitioned as  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ , where  $\Sigma_o$  and  $\Sigma_{uo}$  are the sets of observable and unobservable events of the system. The observable events represent those events that the supervisor can observe, and the unobservable events represent the events of the plant that do not have sensors to identify their occurrence or are not communicated to the supervisor.

### Observer automaton

For the system operation under partial observation, it is important to construct an automaton  $G_{obs}$  that generates the natural projection of the language  $L$  generated by automaton  $G$ ,  $P_o(L)$ . In order to compute  $G_{obs}$ , we first need to define the operation of obtaining the unobservable reach of a state  $x \in X$ , which is a generalization of the notion of  $\varepsilon$ -reach [31].

**Definition 5 (Unobservable reach)** *The unobservable reach of a state  $x \in X$ , denoted by  $UR(x)$ , is defined as:*

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^*)(f(x, t) = y)\}. \quad (2.1)$$

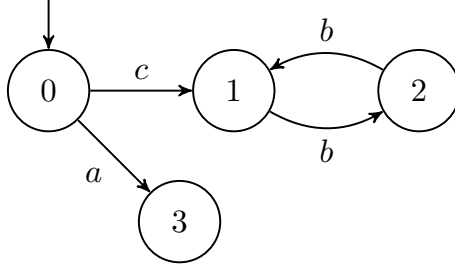


Figure 2.15: Automaton  $G$  of Example 14.

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

$$UR(B) = \bigcup_{x \in B} UR(x). \quad (2.2)$$

Using definition of unobservable reach we can present an algorithm for the computation of  $G_{obs} = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs})$  [31, 33].

---

**Algorithm 1** *Observer automaton*

---

**Input:**  $G = (X, \Sigma, f, x_0)$ .

**Output:** Observer automaton  $G_{obs} = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs})$ .

1: Define  $x_{0,obs} = UR(x_0)$ . Do  $X_{obs} = \{x_{0,obs}\}$  and  $\tilde{X}_{obs} = X_{obs}$ .

2:  $\bar{X}_{obs} = \tilde{X}_{obs}$ ,  $\tilde{X}_{obs} = \emptyset$ .

3: For each  $B \in \bar{X}_{obs}$  do

3.1:  $\Gamma_{G_{obs}}(B) = (\bigcup_{x \in B} \Gamma_G(x)) \cap \Sigma_o$ .

3.2: For each  $\sigma \in \Gamma_{G_{obs}}(B)$ ,

$$f_{obs}(B, \sigma) = UR(\{x \in X : (\exists y \in B)[x = f(y, \sigma)]\}).$$

3.3:  $\tilde{X}_{obs} \leftarrow \tilde{X}_{obs} \cup f_{obs}(B, \sigma)$ .

4:  $X_{obs} \leftarrow X_{obs} \cup \tilde{X}_{obs}$ .

5: Repeat steps 2 to 4 until all accessible part of  $G_{obs}$  is constructed.

---

**Example 14** Let us consider an automaton  $G$ , represented in Figure 2.15, with set of observable events  $\Sigma_o = \{a, b\}$ .

Then, the observer of  $G$ ,  $G_{obs}$  is represented in Figure 2.16. Notice that, since event  $c$  is unobservable, the system cannot differentiate if it is on state 1 or state 0 before an occurrence of event  $b$ .

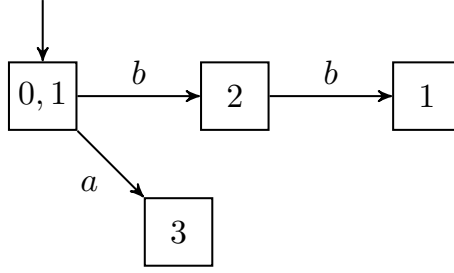


Figure 2.16: Automaton  $G_{obs}$  of Example 14.

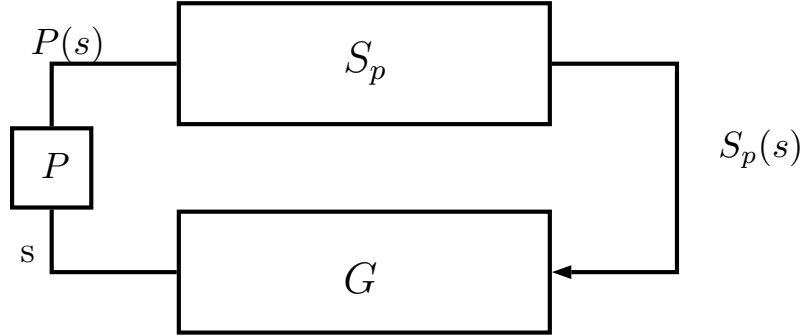


Figure 2.17: Closed-loop system under partial observation

### Controllability and observability

The controllability theorem 1 is not sufficient for the construction of a supervisor under partial observation. As stated in [31], the observability is defined by:

**Definition 6 (observability)** *Let  $K$  and  $M = \overline{M}$  be languages defined over event set  $\Sigma$ . Let  $\Sigma_c$  be a subset of  $\Sigma$ , let  $P : \Sigma^* \rightarrow \Sigma_o^*$  be a projection.  $K$  is said to be observable with respect to  $M$ ,  $\Sigma_o$ , and  $\Sigma_c$  if for all  $s \in \overline{K}$  and for all  $\sigma \in \Sigma_c$ ,*

$$(s\sigma \notin \overline{K}) \text{ and } (s\sigma \in M) \Rightarrow P^{-1}[P(s)]\sigma \cap \overline{K} = \emptyset.$$

The concept of observability can be understood as: if you cannot differentiate between two strings then this two strings should enable the same controllable events.

The supervisor under partial observation observes just the projection of the sequence that was executed by the system. The closed-loop system under partial observation is depicted in Figure 2.17.

The closed-loop system considered in this case is then  $S_P/G$  and we can define the language of this closed-loop system similarly to the case with full observation by:

- 1:  $\varepsilon \in L(S_P/G)$
- 2:  $[(s \in L(S_P/G)) \text{ and } (s\sigma \in L(G)) \text{ and } (\sigma \in S_P(P(s)))] \Leftrightarrow [s\sigma \in L(S_P/G)]$

**Example 15** *Consider the realization of a system represented by  $G$ , depicted in Figure 2.18 (a), with observable event set  $\Sigma_o = \{a, b\}$  and controllable event set*

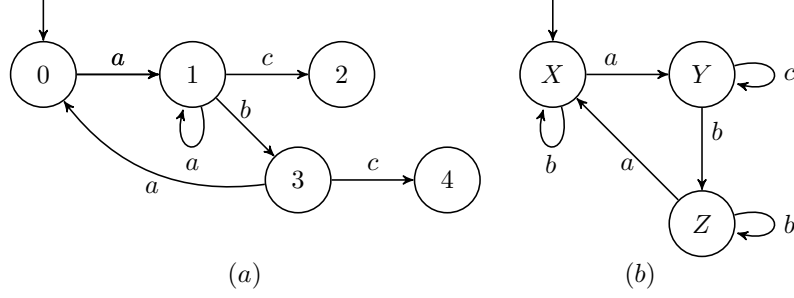


Figure 2.18: Example of automaton  $G$  and supervisor  $H$  of Example 15.

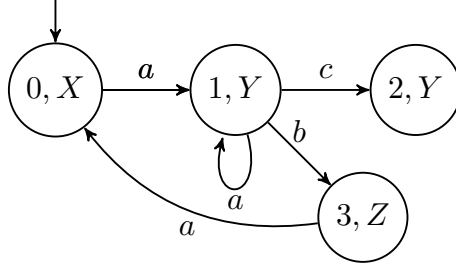


Figure 2.19: Closed-loop system under partial observation  $T$  of Example 15.

$\Sigma_c = \{a, c\}$ . Let  $H$  be the realization of the supervisor that controls this system, shown in Figure 2.18 (b). Notice that all uncontrollable events must be active in all states of  $H$  in order to be admissible, and also that as the supervisor only “sees” observable events, then a transition labeled by an unobservable event cannot change the state of  $H$ , i.e., all transitions with unobservable events must be a self-loop. It is important to remark that as  $c$  is an unobservable event, the projection of a sequence  $s$  and the projection of  $sc$  is the same. So the state in  $H$  cannot change with an occurrence of  $c$ . In addition the event  $b$  is an uncontrollable event, and then, for the supervisor to be admissible,  $b$  must be always enabled in the supervisor when it is active in the plant. Therefore,  $b$  is active in all states of the supervisor. The closed-loop system under partial observation  $T$  is represented in Figure 2.19.

### Construction of a P-supervisor

In Subsection 2.2.3 it is presented how to obtain a realization of a supervisor  $S$  for a system with full observation, however for systems under partial observation the construction of a P-supervisor  $S_P$  is not straightforward. With the specification language  $L(S_P/G) = \overline{K}$ , we first build a trim automaton  $R$  that generates and marks  $\overline{K}$ . Notice that the event set of  $R$  is  $\Sigma$  and  $\Sigma_o$  is the subset of observable events. Then, we build the observer of  $R$ ,  $R_{obs}$ , corresponding to the set of observable events  $\Sigma_o$ . Since all states of  $R$  are marked then all states of  $R_{obs}$  are also marked.

Here, differently from the supervisor  $S$  with full observation, automaton  $R_{obs}$  does not encode the set of enabled events by function  $S_p$ , because the set of events

in  $R_{obs}$  is  $\Sigma_o$ . However, since each state of  $R_{obs}$  is a set of events of  $R$  we can recover the control actions by examining the states of  $R$ .

Let  $s$  be the string of observable events executed by the plant, and  $x_{obs,current}$  be the state in  $R_{obs}$  after the execution of  $s$ . Then:

$$S_P^{real}(t) = \bigcup_{x \in x_{obs,current}} [\Gamma_R(x)].$$

where  $\Gamma_R$  is the active event function of  $R$ . Then, in order to create the control function, we need to include the controllable events that are not observable. Then for each state  $x_{obs} \in X_{obs}$  we add self loops for all unobservable events that appear in

$$\bigcup_{x \in x_{obs}} [\Gamma_R(x)]$$

The supervisory control theory found in the literature is wider than presented in this section, *e.g.*, there are modular approaches [34, 35], the use of abstractions for supervisory control systems [36, 37], hierarchical control [38, 39], etc. However, in this work, we consider an existing supervisor, and therefore, there is no need to cover all these topics.

In order to position this work with respect to other works in the literature, and formulate the problem addressed in this work, a background on cyber-physical systems and security is needed. In this regard, in Section 2.3 we present some background of CPSs and security.

## 2.3 Security in cyber-physical systems

Cyber-physical systems are systems that integrate computing and communication capabilities to monitor and control physical processes. An example of this type of system is a smart grid, and there are several works in the literature that address the problem of security in CPS [1–3, 10, 40–42]. In RAWAT and BAJRACHARYA [42], for instance, the authors make a review in smart grids security and show the difference between a smart grid and the traditional Internet, and then present the vulnerabilities in smart grids. The authors in [42] also show that the majority of the cyber attacks in smart grids are due to malicious threats in the communication network.

This section is divided as follows: in Section 2.3.1 we present the structure of a network, commonly used in CPS. In Section 2.3.2 we describe the structure and challenges in security of a CPS. Finally, in Section 2.3.3, we present the type of attack considered in this work called man-in-the-middle attack.

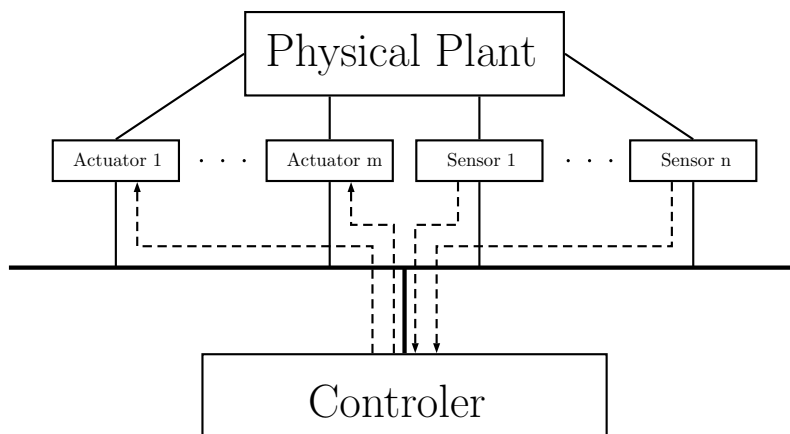


Figure 2.20: Example of structure for CPS

### 2.3.1 Structure of a CPS

As a CPS is a wide concept then the structure of a CPS can be a simple connection between two physical systems or a complex network, like Internet connection with all its layers and protocols in order to control a physical system, for example, a home automation system.

As, in this work, we do not explore all the layers and protocols used in complex networks, there is no need to describe this type of network here. Therefore, we present a simple structure of a CPS as described in [43]. The structure considered in this work is shown in Figure 2.20, in which the physical channel of communication is represented by the solid lines, and the communication flow is represented by dashed lines.

The communications between devices in a CPS normally use specific protocols for the area for which they are designed. For example, in power grids, it is commonly used the Distributed Networking Protocol 3.0 (DNP3), which is standard in North America for power grids, or the International Electrotechnical Commission (IEC) 61850, created to substitute DNP3 in smart grids. More information on these protocols can be found in [42].

On the other hand, the authors of RAWAT and BAJRACHARYA [42] also describe the security challenges in smart grids, which have some factors that must be taken into account in every security system for CPS. This structure and characteristics are best explained in the following section.

### 2.3.2 Security in CPS

There are important characteristics that every networked system must have in order to work properly. These characteristics are known as the CIA triad [44], which are, confidentiality, integrity and availability (CIA triad). Every security protocol should

not compromise any of these factors.

Confidentiality refers to giving access only to authorized people, *i.e.*, any request made by unauthorized parties cannot be executed. Protecting the integrity of information indicates protecting this data from being adulterated by an unauthorized source. Availability means that all information must be available to authorized parties.

There are two main groups of attacks, they can be classified as “active” when it attempts to alter system resources or affects their operation, or classified as “passive” when attempts to learn or make use of information from the system but does not affect system resources. Active attacks, by definition, compromise integrity or availability. On the other hand, “passive” attacks compromises confidentiality.

The most common attack in Internet are: *(i)* the Denial-of-service attack (DoS) or Distributed Denial-of-service attack (DDoS), targeting availability, which consists of requesting a service from different sources at the same time, and with this overloading, the system becomes unavailable; *(ii)* Port Scanning attack, which consists of sending requests to different ports, and then construe this information in order to facilitate future attacks; *(iii)* Trojan Horse, which consists of sending a virus or worm disguised as something else that an user may open, and then creating loopholes in the system in order to leak information or creating security breaches such that an attack can be executed; *(iv)* Brute force attack, which targets confidentiality, and is a type of password cracking, it consists of trying every combination of password until cracking it.

Another attack, that has increased its use in the last years is called man-in-the-middle attack (MITM). The MITM attack is classified as “active” as it can change data, making it a more dangerous attack than the others previously presented.

### **2.3.3 Man-in-the-middle attack**

A dangerous attack in computer security is the MITM attack [45]. In 2016, a wide survey on this type of attack was presented in CONTI *et al.* [46] where the authors show that the MITM attack has been considered as one of the most dangerous attacks against network security in the literature.

The MITM is an attack where the attacker compromises a communication channel between two devices. Once compromised the attacker can see, send or change data from this channel, *i.e.*, the attacker can intercept, modify, change, or replace target communication traffic data. Notice that the MITM attack is different from a simple eavesdropper, since in the latter type of attack the malicious agent can only retrieve information.

The major concern for security in networked connected systems is to maintain

the CIA triad. The MITM is an attack that compromises all factors of the CIA triad, since it compromises confidentiality, by eavesdropping on the communication; it also attacks integrity since it can intercept the communication and modify messages; and lastly it endangers availability, by intercepting and destroying messages. In this regard, a MITM attack is a serious threat to CPSs.

## 2.4 Final comments

In this chapter, in Section 2.1, we define DES, and present the main difference between DES and time-driven systems. It is also shown how to model DESs using automata, and we define formally the languages generated and marked by an automaton. In Section 2.1 we show two types of automata, namely deterministic and nondeterministic automata, and show how to relate each of them with their corresponding languages.

In the following, we present automaton operations. First, it is shown how to take the accessible part and the coaccessible part of an automaton, and then we also present how to operate with two automata in order to compose them with the parallel composition, generating a new automaton.

In Section 2.2, we present some basic notions of the supervisory control theory. We associate the supervisor with a function  $S$  and then show how we can make an automaton realization  $H$  of this supervisor in order to compose it with the plant. In Section 2.2 we also define controllability for DES. In Subsection 2.2.4 we show how the systems behave under partial observation, and how the supervisor must operate under this situation is shown.

In Section 2.3 the concept of CPS is introduced and we present a basic knowledge about security in networked communication system. It is presented some common attacks that compromise a network based system, and then we introduce the man-in-the-middle attack.

In this work, we assume that the attacker has used a MITM attack to compromise a networked based communication between plant and supervisor. The malicious agent in this scenario is able to attack two different types of communication channel: a sensor channel from the supervisor to the plant; or a supervisory control channel from the supervisor to the plant. In the MITM attack the malicious agent can hide, create or change events transmitted in the attacked communication channel. In the next chapter we model this scenario, and then develop a way to identify and verify if the MITM attack would be critical to the system.



## Chapter 3

# Security Against Network Attacks in Supervisory Control Systems

The increase in the use of networks in the feedback control system also increases its vulnerability to malicious attacks. Depending on the type of attack executed by the intruder, the information in the network can be corrupted or completely changed, driving the feedback system to undesirable and unsafe states. One of the most powerful attacks that can be executed is the so-called man-in-the-middle attack. In this type of attack the intruder can observe, hide, create or change the information that transits in the attacked communication channel.

In this work we consider a networked supervisory control system as shown in Figure 3.1. We assume that the communication between supervisor and plant, and conversely, is carried out by using several different channels. The channels that are used to send information, gathered by sensors, from the plant to the supervisor are denoted as sensor channels, and the channels that transmit the control actions, enabling actuators, from supervisor to plant are called supervisory control channels. In Figure 3.1, we can see the communication channels, where the physical bus and its connections with the devices in the system are represented by solid lines, and the flow of information by dashed lines. The plant in the networked supervisory control system is modeled by a deterministic automaton  $G = (X, \Sigma, f, x_0)$ , the realization of the supervisor  $S$  is modeled by a deterministic automaton  $H = (X_H, \Sigma, f_H, x_{0H})$ , and the closed-loop system model  $T = (X_T, \Sigma, f_T, x_{0T})$  is obtained by the parallel composition  $T = G \parallel H$ . We consider that the plant has a set of unsafe states denoted by  $X_{US} \subset X$ , and we assume that the supervisor is designed to avoid the plant from reaching any unsafe state  $x \in X_{US}$ , *i.e.*, no unsafe states are reachable in  $T$ .

Let us consider that there are vulnerable communication channels in the networked supervisory control system of Figure 3.1, and assume that the intruder can execute MITM attacks, *i.e.*, the information in the attacked channel can be completely changed by the intruder. Let us also assume that both sensor communication

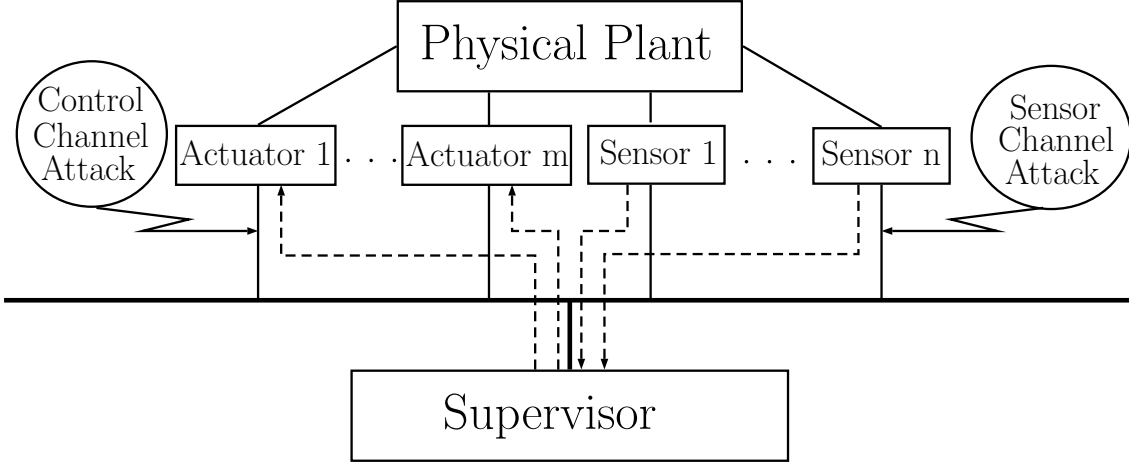


Figure 3.1: Closed-loop system under attack

channels and supervisory control communication channels, can be attacked. Thus, the intruder can hide, insert or change events whose occurrence is detected by sensors in the plant, and can modify the enabling action commanded by the supervisor to the actuators of the plant connected through attacked channels, with the objective to drive the system to reach unsafe states. Let  $ch_{s_i}$  and  $ch_{a_j}$ , for  $i = 1, \dots, n_s$  and  $j = 1, \dots, n_a$ , denote the attacked sensor channels and the attacked supervisory control channels, respectively, where  $n_s$  is the number of attacked sensor channels and  $n_a$  is the number of attacked supervisory control channels. Let  $\Sigma_{s_i} \subset \Sigma_o$  and  $\Sigma_{a_j} \subset \Sigma_c$ , denote, respectively, the set of observable events transmitted through channel  $ch_{s_i}$  and the set of controllable events enabled through channel  $ch_{a_j}$ . Then, the set of events associated with the vulnerable sensor channels is defined as  $\Sigma_{vs} = \sum_{i=1}^{n_s} \Sigma_{s_i}$ , and the set of events associated with the vulnerable supervisory control channels is defined as  $\Sigma_{va} = \bigcup_{j=1}^{n_a} \Sigma_{a_j}$ .

The following assumption over the model of the system is made in LIMA *et al.* [17], and is also considered initially in this work.

**A1.** *the sets of controllable and observable events are disjoint, i.e.,  $\Sigma_o \cap \Sigma_c = \emptyset$ .* Assumption A1 will be relaxed later in this chapter.

In this work, we propose the design of an Intrusion Detection Module that, as shown in Figure 3.2, observes the traces observed by the supervisor and, after detecting an intrusion that certainly leads to unsafe states, prevents the system from reaching these states by forcing the supervisor to disable all controllable events of the system. It is important to remark that the Intrusion Detection Module allows that traces generated after network attacks, that do not belong to the language generated by the closed-loop system  $T$ , be executed if these traces lead to safe states.

The existence of the Intrusion Detection Module depends on the language of the system, and on the vulnerable event sets  $\Sigma_{vs}$  and  $\Sigma_{va}$ . Thus, we present in

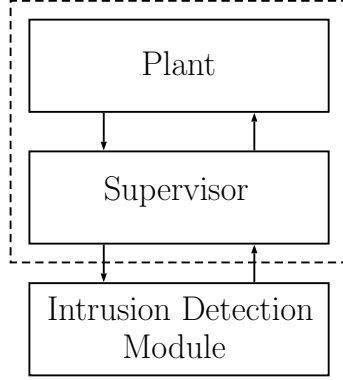


Figure 3.2: Intrusion detection module.

the sequel the modelling of the system under sensor channel attacks and control channel attacks, and then we present the definition of NA-Safe controllability that is associated with the capability of detecting and preventing damages caused by network attacks.

### 3.1 Model of the plant subject to sensor channel attacks

Since the Intrusion Detection Module observes the same traces that the supervisor observes, and assuming that the attack has been executed in sensor communication channels, then, in order to model this type of attack, we propose an algorithm for the construction of a nondeterministic automaton  $\mathcal{G}_A$ , from automaton  $G$ , that generates all possible traces modified by the attacks in the sensor channels. In order to do so, we define the sensor attack function  $A_S : \Sigma^* \rightarrow 2^{\Sigma^*}$  as follows.

$$\begin{aligned}
 A_S(\sigma) &= \begin{cases} \Sigma_{vs}^*, & \text{if } \sigma \in \Sigma_{vs} \cup \{\varepsilon\} \\ \Sigma_{vs}^* \{\sigma\} \Sigma_{vs}^*, & \text{if } \sigma \in \Sigma \setminus \Sigma_{vs} \end{cases} \\
 A_S(s\sigma) &= A_S(s)A_S(\sigma), \text{ for all } s \in \Sigma^*, \text{ and } \sigma \in \Sigma.
 \end{aligned}$$

Function  $A_S$  can be extended to a language  $K \subseteq \Sigma^*$  by applying  $A_S(s)$  to all traces  $s \in K$ . It can be seen that when a trace  $s \in L$  is executed by the plant  $G$ , the intruder can modify  $s$  to obtain any trace in  $A_S(s)$  by removing, replacing and/or adding events belonging to  $\Sigma_{vs}$ . Thus, the supervisor may observe any trace in  $P_o[A_S(s)]$ , where  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  is a projection. In order to illustrate this fact, consider  $\Sigma = \Sigma_o = \{a, b, c\}$ ,  $\Sigma_{vs} = \{a, c\}$  and trace  $s = abba$ . Since events  $a$  and  $c$  can be removed, replaced and/or added, then, when trace  $s$  is executed by the plant, the intruder is able to generate any trace in  $A_S(s) = \{a, c\}^* \{b\} \{a, c\}^* \{b\} \{a, c\}^*$ . Therefore, when the feedback system is subject to sensor communication attacks,

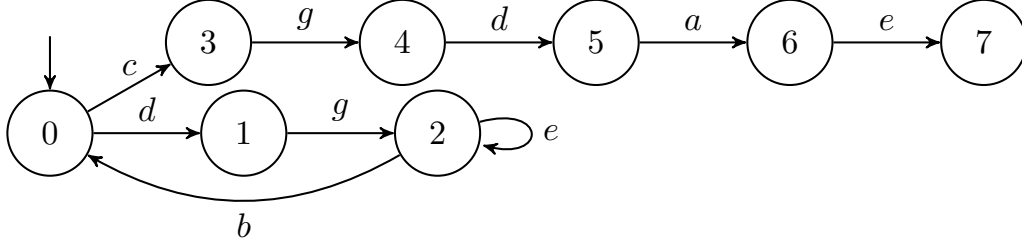


Figure 3.3: Model of the plant  $G$ .

the decisions of the supervisor can drive the plant to an unsafe state, since it is based on  $P_o[A_s(L(G))]$ , we denote the set of unsafe states of the plant by  $X_{US}$ . In the following algorithm, we construct an automaton that generates language  $A_S(L(G))$ .

---

**Algorithm 2** *Computation of automaton  $\mathcal{G}_A$  that models the plant subject to sensor channel attacks*

---

**Input:**

- $G = (X, \Sigma, f, x_0)$  [Plant model]
- $\Sigma_{vs} \subseteq \Sigma_o$  [Set of vulnerable observable events]

**Output:**

- $\mathcal{G}_A = (X, \Sigma_A, f_A, x_0)$  [Model of the plant subject to sensor channel attacks]

1: Define  $\Sigma_A = \Sigma \cup \{\varepsilon\}$ .

2: Define  $f_A(x, \sigma)$ ,  $\forall x \in X$ , and  $\forall \sigma \in \Sigma_A$ , as follows:

$$f_A(x, \sigma) = \begin{cases} \{f(x, \sigma)\}, & \text{if } \sigma \in \Gamma_G(x) \wedge \sigma \in \Sigma \setminus \Sigma_{vs}, \\ \{x\} \cup \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}, & \\ & \text{if } \sigma \in \Sigma_{vs}, \\ \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}, & \text{if } \sigma = \varepsilon, \\ \text{undefined, otherwise.} & \end{cases}$$

---

**Example 16** *Let  $G$  be the automaton model of the plant shown in Figure 3.3, where  $\Sigma = \Sigma_o = \{a, b, c, d, e\}$ , and consider that only one sensor channel, that transmits the observation of event  $c$  to the supervisor, is attacked. In this case,  $\Sigma_{vs} = \{c\}$ . Following the steps of Algorithm 2, we obtain the model of the plant subject to sensor channel attacks  $\mathcal{G}_A$ , shown in Figure 3.4.*

*It is important to notice that  $\mathcal{G}_A$  is a nondeterministic automaton, since it models the false observation of events when the plant does not change its state (modeled by*

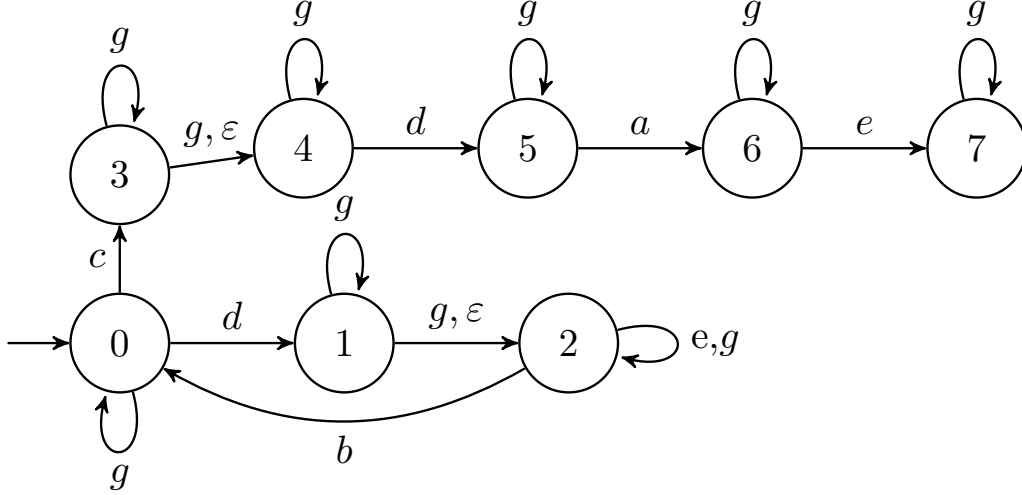


Figure 3.4: Model of the plant subject to sensor channel attacks  $\mathcal{G}_A$ .

the self-loops labeled with events in  $\Sigma_{vs}$ ), and also the creation of transitions labeled with  $\varepsilon$ , and events in  $\Sigma_{vs}$ , in parallel with all transitions labeled with any event in  $\Sigma_{vs}$ , to represent the actions of the intruder of deleting and changing the observation of events, respectively.  $\square$

The following propositions show that the language generated by  $\mathcal{G}_A$  represents all possible modifications in the traces of  $L(G)$  after sensor channel attacks.

**Proposition 1** *Let  $X_G(t) \subseteq X$  denote the set of states that can be reached in  $G$  after the occurrence of a trace  $t \in A_S(L(G))$ , i.e.,  $X_G(t) = \{x \in X : (\exists s \in L(G))[t \in A_S(s) \wedge f(x_0, s) = x]\}$ . Then,  $X_G(t) = f_A^e(x_0, t)$ , where  $f_A^e$  is obtained from  $f_A$  by extending its domain to  $X \times \Sigma^*$ .*

*Proof:* The proof is by induction in the length of trace  $t$ .

(i) *Basis step.* Since the malicious agent can only remove events belonging to  $\Sigma_{vs}$ , only traces  $s \in \Sigma_{vs}^*$  can be modified to  $t = \varepsilon$  and, consequently,  $X_G(\varepsilon)$  is equal to the set of states in  $X$  that are reached in  $G$  from  $x_0$  by following a trace in  $\Sigma_{vs}^*$ . According to the definition of  $f_A$ , for all  $x \in X$ ,  $f_A(x, \varepsilon) = \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}$ . Therefore,  $X_G(\varepsilon) = f_A^e(x_0, \varepsilon)$ .

(ii) *Induction hypothesis.* For all  $t \in \Sigma^*$  such that  $|t| \leq n$ ,  $X_G(t) = f_A^e(x_0, t)$ .

(iii) *Inductive step.* Let us consider a trace  $t_{n+1} = t\sigma$  such that  $|t| = n$ . Then,  $|t_{n+1}| = n + 1$ . According to the induction hypothesis,  $X_G(t) = f_A^e(x_0, t)$ , which implies that  $X_G(t_{n+1})$  is the set of states of  $G$  that can be reached from a state in  $f_A^e(x_0, t)$  by following a trace  $s_\sigma$  such that  $\sigma \in A_S(s_\sigma)$ . The following cases can occur:

(a) If  $\sigma \in \Sigma_{vs}$ , then a trace in  $\Sigma_{vs}^*$  may have occurred in the plant, and its observation removed by the malicious agent that allows the observation of only one event  $\sigma$ .

Then,  $X_G(t_{n+1}) = X_G(t) \cup \{x \in X : (\exists x' \in X_G(t) \wedge \exists s_v \in \Sigma_{vs}^*)[f(x', s_v) = x]\}$ . According to the definition of  $f_A$ , for all  $x \in X$ ,  $f_A(x, \sigma) = \{x\} \cup \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}$  if  $\sigma \in \Sigma_{vs}$ , and  $f_A(x, \varepsilon) = \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}$ . Then, we conclude that  $X_G(t_{n+1}) = f_A^e(x_0, t_{n+1})$ .

(b) If  $\sigma \in \Sigma \setminus \Sigma_{vs}$ , then event  $\sigma$  occurred since the malicious agent cannot alter the observation of events in  $\Sigma \setminus \Sigma_{vs}$ . Moreover, a trace in  $\Sigma_{vs}^*$  may have occurred in the plant and its observation removed by the malicious agent. Then,  $X_G(t_{n+1}) = \{x \in X : (\exists x' \in X_G(t) \wedge \exists s_v, s'_v \in \Sigma_{vs}^*)[f(x', s_v \sigma s'_v) = x]\}$ . According to the definition of  $f_A$ , if  $\sigma \in \Gamma_G(x) \cap (\Sigma \setminus \Sigma_{vs})$ , then, for all  $x \in X$ ,  $f_A(x, \sigma) = f(x, \sigma)$ . In addition,  $f_A(x, \varepsilon) = \{f(x, \sigma_v) : \sigma_v \in \Gamma_G(x) \cap \Sigma_{vs}\}$ . Therefore, we conclude that  $X_G(t_{n+1}) = f_A^e(x_0, t_{n+1})$ . ■

**Proposition 2**  $L(\mathcal{G}_A) = A_S(L(G))$ .

*Proof:* The proof is by induction in the length of the traces of  $\Sigma^*$ .

(i) *Basis step.* By definition,  $\varepsilon \in L(\mathcal{G}_A)$ , and  $\varepsilon \in A_S(L(G))$  since  $\varepsilon \in A_S(\varepsilon) \subseteq A_S(L(G))$ .

(ii) *Induction hypothesis.* For all  $s \in \Sigma^*$  such that  $|s| \leq n$ ,  $s \in L(\mathcal{G}_A)$  if, and only if,  $s \in A_S(L(G))$ .

(iii) *Inductive step.* Consider a trace  $s\sigma \in \Sigma^*$ , where  $|s| = n$ . Notice that languages  $L(\mathcal{G}_A)$  and  $A_S(L(G))$  are prefix-closed. Then, by using the induction hypothesis, we conclude that it is sufficient to consider  $s \in L(\mathcal{G}_A) \cap A_S(L(G))$ . There are two possible cases: (a) when  $\sigma \in \Sigma_{vs}$ , according to the definition of  $f_A$ , for all  $x \in f_A^e(x_0, s)$ ,  $f_A(x, \sigma)$  is always defined, which implies that  $s\sigma \in L(\mathcal{G}_A)$ . In addition, since  $s\sigma \in \{s\}A_S(\varepsilon) \subseteq A_S(L(G))$ , then  $s\sigma \in A_S(L(G))$ . Thus,  $s\sigma \in L(\mathcal{G}_A) \cap A_S(L(G))$ ; (b) when  $\sigma \in (\Sigma \setminus \Sigma_{vs})$ ,  $s\sigma \in L(\mathcal{G}_A)$  if and only if  $(\exists x \in f_A^e(x_0, s))[\sigma \in \Gamma_G(x)] \Leftrightarrow (\exists s' \in L(G))[s \in A_S(s') \wedge s'\sigma \in L(G)] \Leftrightarrow s\sigma \in A_S(L(G))$ , where the first equivalence relation is a consequence of Proposition 1, and the last relation is a consequence of the definition of  $A_S$ . ■

## 3.2 Model of the supervisor subject to supervisory control channel attacks

If the attack occurs in the supervisory control communication channel, the intruder has the ability to enable or disable controllable events associated with the actuators of the plant. In this case, the intruder drives the plant to unsafe states by changing the enablement of the vulnerable events in  $\Sigma_{va}$ . Thus, in order to model the attacks in the supervisory control channels, we modify the model of the supervisor by allowing the modifications that the intruder can execute in the vulnerable controllable

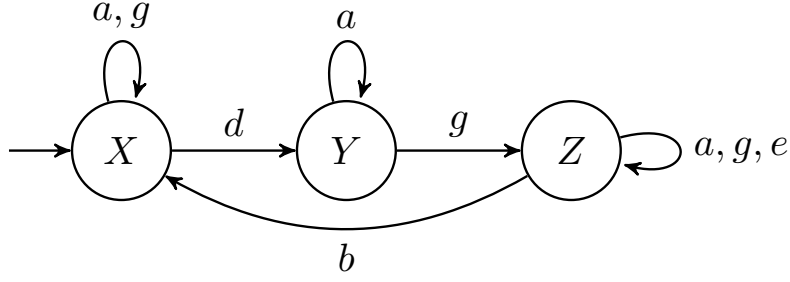


Figure 3.5: Supervisor realization  $H$ .

events. It is important to remark that only the attacks that lead to unsafe states are considered in this work. Thus, it is not important to model the disabling actions of the attack, since, when this happens, the supervisory control system does not allow the plant to reach an unsafe state.

In the following algorithm we present a method to generate a deterministic automaton  $H_A$  that models the supervisor after attacks in the supervisory control communication channels.

---

**Algorithm 3** *Computation of automaton  $H_A$  that models the supervisor subject to control channel attacks*

---

**Input:**

- $H = (X_H, \Sigma, f_H, x_{0H})$  [Supervisor model]
- $\Sigma_{va} \subseteq \Sigma_c$  [Set of vulnerable controllable events]

**Output:**

- $H_A = (X_H, \Sigma, f_{HA}, x_{0H})$ . [Model of the supervisor subject to control channel attacks]

1: Define  $f_{HA}(x, \sigma), \forall x \in X_H, \forall \sigma \in \Sigma$  as follows:

$$f_{HA}(x, \sigma) = \begin{cases} f_H(x, \sigma), & \text{if } \sigma \in \Gamma_H(x) \\ x, & \text{if } \sigma \in \Sigma_{va} \setminus \Gamma_H(x) \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

---

**Example 17** Let  $G$  be the plant shown in Figure 3.3, where  $\Sigma_c = \{b, c, d, e\}$ , and let  $H = (X_H, \Sigma, f_H, x_{0H})$ , depicted in Figure 3.5, be a realization of the supervisor  $S$  of the system.

Assume that the set of vulnerable controllable events is  $\Sigma_{va} = \{e\}$ . Then, using Algorithm 3, we obtain the realization of the supervisor after control attacks  $H_A$

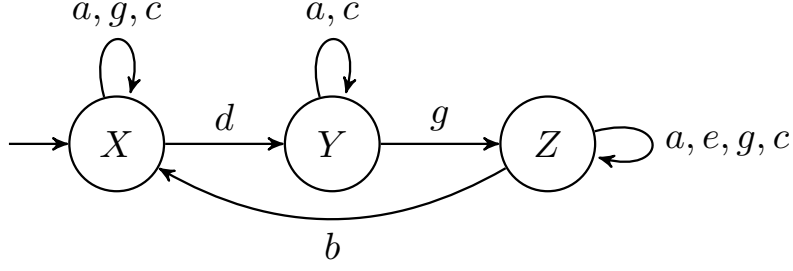


Figure 3.6: Realization of the supervisor after control channel attacks  $H_A$ .

shown in Figure 3.6. Notice that  $H_A$  is obtained from  $H$  by adding self-loops in the states  $x$  of  $H$ , labeled with the events in  $\Sigma_{va}$  that are not feasible in  $x$ .  $\square$

It is important to remark that since the intruder has the capability of completely changing the enablement of the events in  $\Sigma_{va}$ , then these events become uncontrollable. This fact is modeled by the self-loops added in supervisor  $H_A$ .

**Proposition 3** For all traces  $s_o \in P_o[L(G)]$ ,

$$\Gamma_{H_A}(f_{H_A}(x_{0H}, s_o)) = \Gamma_H(f_H(x_{0H}, s_o)) \cup \Sigma_{va}.$$

*Proof:* The proof is straightforward from the construction of  $H_A$ .  $\blacksquare$

### 3.3 Model of the closed-loop system subject to network attacks

In accordance with Section 3.1, the behavior of  $G$  in the presence of sensor channel attacks is modeled by the nondeterministic automaton  $\mathcal{G}_A$  and, in accordance with Section 3.2, the supervisor realization subject to control channel attacks is modeled by the deterministic automaton  $H_A$ . Then, the closed-loop system subject to network attacks can be modeled by  $\mathcal{T}_A = (X_{TA}, \Sigma_{TA}, f_{TA}, x_{0TA}) = \mathcal{G}_A \parallel H_A$ . Notice that the vulnerable events of  $\Sigma_{va}$  are uncontrollable after the attack. Thus, we need to define the new set of controllable events of the attacked system as  $\Sigma_{ca} = \Sigma_c \setminus \Sigma_{va}$ . In addition, we define the set of unsafe states of  $\mathcal{T}_A$  as  $X_{US}^{TA} = \{(x, y) \in X_{TA} : x \in X_{US}\}$ .

**Remark 3** If there is no sensor channel attack, i.e.,  $\Sigma_{vs} = \emptyset$ , (resp. control channel attack, i.e.,  $\Sigma_{va} = \emptyset$ ), then automaton  $\mathcal{G}_A$  (resp.  $H_A$ ) will be equal to  $G$  (resp.  $H$ ). Therefore, the method presented here can be straightforwardly used when the system is subject only to control channel attacks (resp. sensor channel attacks).

**Example 18** Let us consider the system modeled by automaton  $G$ , depicted in Figure 3.3, where  $\Sigma_o = \{a, b, c, d, e\}$  and  $X_{US} = \{6\}$ , and the supervisor realization



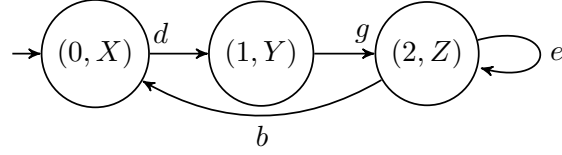


Figure 3.7: Automaton for the closed-loop system (T).

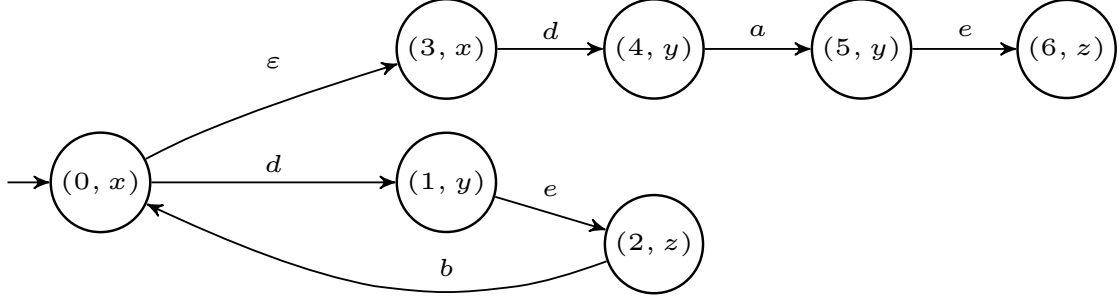


Figure 3.8: Automaton of the closed-loop system subject to network attacks  $\mathcal{T}_A$ .

$H$  presented in Figure 3.5. Notice that there is no unsafe state in the closed-loop system  $T = G||H$  shown in Figure 3.7, which shows that the supervisor avoids the reach of unsafe states.

Consider now a sensor channel attack such that  $\Sigma_{vs} = \{c\}$ , and a control channel attack such that  $\Sigma_{va} = \{e\}$ . Then, the attacked closed-loop system  $\mathcal{T}_A = \mathcal{G}_A||H_A$ , depicted in Figure 3.8, has the set of unsafe states  $X_{US}^{TA} = \{(6, z)\}$ , which means that the intruder is capable of driving the system to reach an unsafe state after the attacks.  $\square$

### 3.4 NA-Safe Controllability

Depending on the attack executed by the intruder, the closed-loop system can reach unsafe states in  $X_{US}^{TA}$ . Thus, in order to prevent damages to the plant, we propose in this work the implementation of a module that is capable of identifying traces such that their continuations certainly lead to states in  $X_{US}^{TA}$ , and then block the system before an unsafe state is reached.

Let  $L_{TA}$  denote the language generated by  $T_A$ . Then,  $L_{TA}$  can be partitioned as  $L_{TA} = L_s(\mathcal{T}_A) \dot{\cup} L_{us}(\mathcal{T}_A)$ , where  $L_s(\mathcal{T}_A)$  denotes the safe language, composed of traces  $s \in L_{TA}$  such that we cannot reach an unsafe state in  $X_{US}^{TA}$  after the occurrence of  $s$ , i.e.,  $L_s(\mathcal{T}_A) = \{s \in L_{TA} : (\forall t \in L_{TA}/s)[f_{TA}^e(x_{0TA}, st) \cap X_{US}^{TA} = \emptyset]\}$ , and  $L_{us}(\mathcal{T}_A)$  denotes the unsafe language, composed of traces  $s \in L_{TA}$  such that we can reach an unsafe state in  $X_{US}^{TA}$  after the occurrence of  $s$ , i.e.,  $L_{us}(\mathcal{T}_A) = \{s \in L_{TA} : (\exists t \in L_{TA}/s)[f_{TA}^e(x_{0TA}, st) \cap X_{US}^{TA} \neq \emptyset]\}$ . Notice that a trace in the unsafe language may be part of the normal behavior of the system, i.e., may belong to the language

generated by  $T$ ,  $L_T$ , or even be part of  $\bar{L}_s(\mathcal{T}_A)$ . Therefore, the module may act only after distinguishing traces of  $L_{TA}$  that will certainly reach an unsafe state, and do not belong to the language generated by the system before an attack  $L_T$ .

We present in the sequel a property of the language of the closed-loop system subject to network attacks,  $L_{TA}$ , that is associated with the capability of designing the Intrusion Detection Module.

**Definition 7 (NA-Safe Controllability)** *Let  $\mathcal{T}_A$  be the automaton model of the closed-loop system subject to network attacks, and let  $T$  denote the automaton of the closed-loop system without network attacks. Let  $L_T$  and  $L_{TA}$  be the generated languages of  $T$  and  $\mathcal{T}_A$ , respectively. Then,  $L_{TA}$  is said to be NA-Safe Controllable with respect to  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  and a set of unsafe states  $X_{US}^{TA}$  if*

$$(\forall s \in L_{TA})[f_{TA}^e(x_{0TA}, s) \cap X_{US}^{TA} \neq \emptyset] \Rightarrow (s = s_1s_2)[(\forall \omega \in L_T \cup \bar{L}_s(\mathcal{T}_A))[P_o(s_1) \neq P_o(\omega)] \wedge (\Sigma_{ca} \in s_2)]. \quad \square$$

According to Definition 7,  $L_{TA}$  is NA-safe controllable if all traces  $s \in L_{TA}$ , that lead the system to an unsafe state in  $X_{US}^{TA}$ , have an unsafe prefix  $s_1 \in L_{us}(\mathcal{T}_A)$  that can be distinguished from any safe and normal traces of the system, and a continuation  $s_2$  that has an event from  $\Sigma_{ca}$ , which allow that unsafe traces that certainly lead the system to an unsafe state be detected, and the reach of unsafe states prevented by disabling the controllable events of  $\Sigma_{ca}$ .

Notice that, the supervisor could be designed to disable all controllable events of the plant after detecting the observation of traces that do not belong to  $P_o(L_T)$ . However, this approach would be more restrictive than the approach proposed in this work, since the system can execute safe traces in  $\bar{L}_s(\mathcal{T}_A)$  after an attack that do not represent danger to the system.

### 3.4.1 Verification of NA-Safe controllability

In order to present an algorithm for the verification of NA-Safe controllability, we need to define two functions, namely, the rename function and the uncontrollable reach function.

The rename function  $R$  [47] is defined, for all  $\sigma \in \Sigma \cup \{\varepsilon\}$ , as follows.

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

Notice that, the function  $R$  can be extended to a set of events  $\Sigma$  by:

$$R(\Sigma) = \bigcup_{\sigma \in \Sigma} R(\sigma).$$

Let  $\mathcal{G} = (X, \Sigma, f_{nd}, x_0, X_m)$  be a nondeterministic automaton. The uncontrollable reach  $UR$  is defined, for all state  $x \in X$  and a set of uncontrollable events  $\Sigma_{uc}$ , as  $UR(x, \Sigma_{uc}) = \{y \in X : (\exists t \in \Sigma_{uc}^*) [f_{nd}^e(x, t) = y]\}$ . This function is extended to a set of states  $B \subseteq X$  as  $UR(B, \Sigma_{uc}) = \bigcup_{x \in B} UR(x, \Sigma_{uc})$ .

---

**Algorithm 4** *NA-Safe Controllability Verifier*

---

**Inputs:**

- $T = (X_T, \Sigma_T, f_T, x_{0T})$  [Automaton model of the closed-loop system]
- $\mathcal{T}_A = (X_{TA}, \Sigma_{TA}, f_{TA}, x_{0TA})$  [Automaton model of the closed-loop system subject to network attacks]
- $\Sigma_{ca}$  [Set of controllable events of  $\mathcal{T}_A$ ]
- $X_{US}^{TA} \subset X_{TA}$  [Set of unsafe states]

**Output:**  $NASafeCont \in \{TRUE, FALSE\}$

1: Compute  $\mathcal{T}_U$  and  $\mathcal{T}_S$  as follows:

1.1: Define  $X_{US}^{TA}$  as the set of marked states of  $\mathcal{T}_A$ .

1.2: Define  $\mathcal{T}_U = CoAc(\mathcal{T}_A) = (X_U, \Sigma_A, f_U, x_{0U}, X_{US}^{TA})$ , and unmark its states.

1.3: Define  $X_{TA} \setminus X_U$  as the set of marked states of  $\mathcal{T}_A$ .

1.4: Define  $\mathcal{T}'_S = CoAc(\mathcal{T}_A)$ , and unmark its states.

1.5: Construct automaton  $\mathcal{T}_S$  such that  $L(\mathcal{T}_S) = L(\mathcal{T}'_S) \cup L_T$ .

2: Construct automaton  $\mathcal{T}_{U,R} = (X_U, \Sigma_R, f_R, x_{0U})$ , where  $\Sigma_R = R(\Sigma_A)$  and  $f_R(x, R(\sigma)) = f(x, \sigma), \forall \sigma \in \Sigma_A$ .

3: Compute  $\mathcal{V} = (X_V, \Sigma_V, f_V, x_{0V}) = \mathcal{T}_S || \mathcal{T}_{U,R}$ .

4: Compute the unsafe region  $US_R$ , and unsafe boundary  $US_B$ , as follows:

4.1: Define the transpose of  $\mathcal{T}_A$  as  $\mathcal{T}_A^T = (X_{TA}, \Sigma_A, f_{TA}^T, x_{0TA})$  where  $f_{TA}^T(x, \sigma) = y \Leftrightarrow f_{TA}(y, \sigma) = x, \forall x, y \in X_{TA}$ , and  $\forall \sigma \in \Sigma_A$ .

4.2: Define the unsafe region from  $\mathcal{T}_A^T$  as  $US_R = UR(X_{US}^{TA}, \Sigma_{uca})$ , where  $\Sigma_{uca} = \Sigma \setminus \Sigma_{ca}$ .

4.3: Define the unsafe boundary from  $\mathcal{T}_A^T$  as  $US_B = \{x \in X_{TA} \setminus US_R : (\exists \sigma \in \Sigma_A) \wedge (\exists y \in US_R) [f_{TA}^T(y, \sigma) = x]\}$ .

5: If there exists a state  $x_V = (x_S, x_U) \in X_V$  such that  $x_U \in US_R \cup US_B$  then  $NASafeCont = FALSE$ , otherwise  $NASafeCont = TRUE$ .

---

In Algorithm 4, we describe a method for verifying the NA-Safe controllability of  $L_{TA}$ . The verification is based on the construction of the verifier automaton  $\mathcal{V} = \mathcal{T}_S \parallel \mathcal{T}_{U,R}$ , where the language generated by  $\mathcal{T}_S$  is  $L_T \cup \bar{L}_s(\mathcal{T}_A)$ , and the language generated by  $\mathcal{T}_{U,R}$  is composed of all renamed unsafe traces of  $L_{TA}$ . As shown in [47], all traces of  $\mathcal{V}$  are associated with a trace in  $L_T \cup \bar{L}_s(\mathcal{T}_A)$  and a trace in  $L_{us}(\mathcal{T}_A)$  that have the same projection. Thus, if a trace of  $\mathcal{V}$  reaches a state  $x_V = (x_S, x_U)$ , where  $x_U \in US_R \cup US_B$ , then there exist an unsafe trace in  $L_{us}(\mathcal{T}_A)$ , and a trace in  $L_T \cup \bar{L}_s(\mathcal{T}_A)$  that cannot be distinguished, and since state  $x_U$  is in the unsafe boundary or in the unsafe region, it is impossible to prevent reaching an unsafe state by disabling controllable events.

**Theorem 2** *The language generated by  $\mathcal{T}_A$ ,  $L_{TA}$ , is NA-safe controllable with respect of  $X_{US}^{TA}$  and  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  if, and only if,  $NASafeCont = TRUE$ .*

*Proof:* ( $\Rightarrow$ ) Let us first consider that  $NASafeCont = FALSE$ . Then, there exists a state  $x_V = (x_S, x_U) \in X_V$  such that either  $x_U \in US_R$ , or  $x_U \in US_B$ . Since  $\mathcal{V} = \mathcal{T}_S \parallel \mathcal{T}_{U,R}$ , then, in accordance with [47],  $\exists \omega \in L(\mathcal{T}_S)$  and  $\exists s_1 \in L(\mathcal{T}_U)$  such that  $P_o(\omega) = P_o(s_1)$ . If  $x_U \in US_B$ , then there exists an event  $\sigma \in \Sigma_{ca}$  such that  $f_{TA}^e(x_{0TA}, s_1\sigma) \cap US_R \neq \emptyset$ . According to the construction of  $US_R$ , there exists a trace  $\bar{s}_2 \in \Sigma_{uca}^*$  such that  $f_{TA}^e(x_{0TA}, s_1\sigma\bar{s}_2) \cap X_{US}^{TA} \neq \emptyset$ . Thus, defining trace  $s = s_1\sigma\bar{s}_2$ , we conclude that  $s$  violates Definition 7 since there exists  $\omega \in L_T \cup \bar{L}_s(\mathcal{T}_A)$  such that  $P_o(s_1) = P_o(\omega)$ , and  $\Sigma_{ca} \not\subseteq \bar{s}_2$ . Therefore, language  $L_{TA}$  is not NA-Safe controllable. On the other hand, if  $x_U \in US_R$ , then,  $\exists \omega \in L(\mathcal{T}_S)$  and  $\exists s_1 \in L(\mathcal{T}_U)$  such that  $P_o(\omega) = P_o(s_1)$  and  $f_{TA}^e(x_{0TA}, s_1) \cap US_R \neq \emptyset$ . According to the definition of  $US_R$ , there exists a trace  $s_2 \in \Sigma_{uca}^*$ , such that  $f_{TA}^e(x_{0TA}, s_1s_2) \cap X_{US}^{TA} \neq \emptyset$ , which violates Definition 7, and language  $L_{TA}$  is not NA-Safe controllable.

( $\Leftarrow$ ) Let us consider now that  $NASafeCont = TRUE$ . Then, for all  $x_V = (x_S, x_U) \in X_V$ , we have that  $x_U \notin US_B$  and  $x_U \notin US_R$ . Since, as shown in [47], the verifier represents only the traces of  $L(\mathcal{T}_S)$  and  $L(\mathcal{T}_U)$  that have the same projection, we conclude that for all traces  $s_1 \in L(\mathcal{T}_U)$  that reaches a state in  $US_B$ ,  $P_o(s_1) \notin P_o(L(\mathcal{T}_S))$ . From the constructions of  $US_R$  and  $US_B$ , each state  $x \in X_{US}^{TA}$  of  $\mathcal{T}_A$  is reached from a state in  $US_B$  through a trace  $s_2 = \sigma\bar{s}_2$ , where  $\sigma \in \Sigma_{ca}$  and  $\bar{s}_2 \in \Sigma_{uca}^*$ . Therefore, for all trace  $s \in L_{TA}$ ,  $f_{TA}^e(x_{0TA}, s) \cap X_{US}^{TA} \neq \emptyset \Rightarrow (s = s_1s_2)[(\forall \omega \in L(\mathcal{T}_S))[P_o(s_1) \neq P_o(\omega)] \wedge (\Sigma_{ca} \in s_2)]$ . Since,  $L(\mathcal{T}_S) = L_T \cup \bar{L}_s(\mathcal{T}_A)$ , we conclude that language  $L_{TA}$  is NA-Safe controllable. ■

**Example 19** *Let us consider again the plant automaton of Example 16, depicted in Figure 3.3, where  $\Sigma_o = \{a, b, c, d, e\}$  and  $\Sigma_c = \{b, c, d, e\}$ , and assume that  $\Sigma_{vs} = \{c\}$  and  $\Sigma_{va} = \{e\}$ . Then, automata  $T$  and  $\mathcal{T}_A$ , shown in Figures 3.7 and 3.8, respectively, are computed as presented in Example 18. In this case  $\Sigma_{ca} = \Sigma_c \setminus \Sigma_{va} = \{b, c, d\}$ , and  $X_{US}^{TA} = \{(6, z)\}$ .*

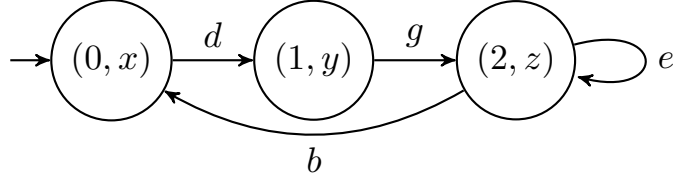


Figure 3.9: Automaton  $\mathcal{T}_S$ .

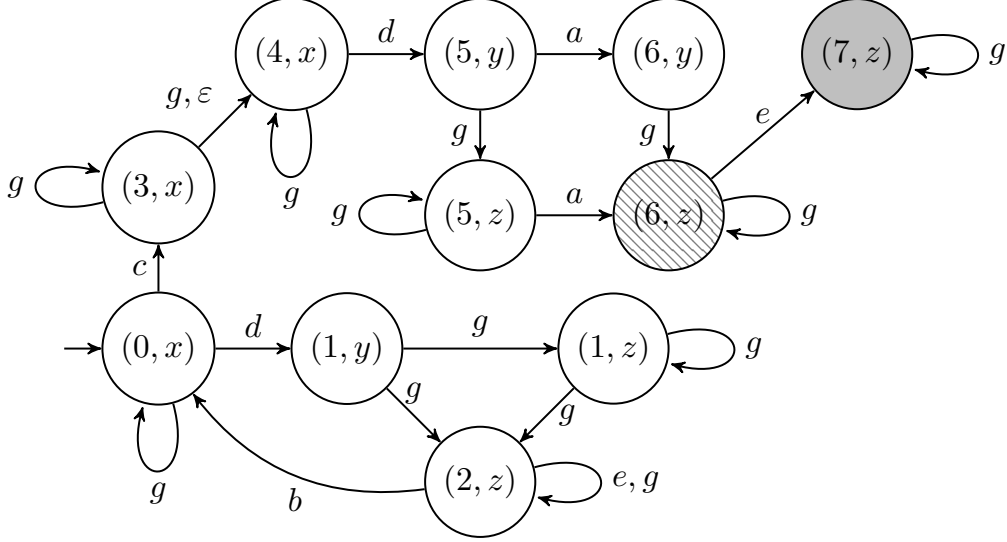


Figure 3.10: Automaton  $\mathcal{T}_U$ .

According to the first step of Algorithm 4, automata  $\mathcal{T}_S$  and  $\mathcal{T}_U$ , depicted in Figures 3.9 and 3.10, respectively, are computed, in this work the unsafe region is represented by a gray filled state, and the unsafe boundary by a state with gray hashed state. Then, in Step 2, automaton  $\mathcal{T}_{U,R}$ , depicted in Figure 3.11 is computed by renaming the events in  $\Sigma_{uo}$  of  $\mathcal{T}_U$ , and in Step 3, the verifier automaton  $\mathcal{V}$ , shown in Figure 3.12, is computed. In order to verify the NA-Safe controllability of  $L_{TA}$ , it is important to find the unsafe region  $US_R$  and unsafe boundary  $US_B$  of  $\mathcal{T}_A$ . By following the procedure presented in Step 4, we obtain  $US_R = \{(5, y), (6, z), (4, y)\}$  and,  $US_B = \{(3, x)\}$ . Since there is a state  $((0, x), (3, x))$  in verifier  $\mathcal{V}$ , and  $(3, x)$  is in the unsafe boundary  $US_B$ ,  $L_{TA}$  is not NA-Safe controllable. Notice that it is easy to find the traces  $s$  and  $\omega$  that violate the NA-Safe controllability condition. In this example,  $s = \varepsilon dae \in L(\mathcal{T}_U)$ , associated with path  $((0, x), \varepsilon, (3, x), d, (4, y), a, (5, y), e, (6, z))$ , has prefix  $s_1 = \varepsilon d$  with the same projection as the trace  $\omega = d \in L(\mathcal{T}_S)$ , associated path  $((0, x), d, (1, y))$ . Since  $a, e \in \Sigma_{uca}$ , the suffix of  $s$ ,  $s_2 = ae \in \Sigma_{uca}^*$ , then  $L_{TA}$  is not NA-Safe controllable.

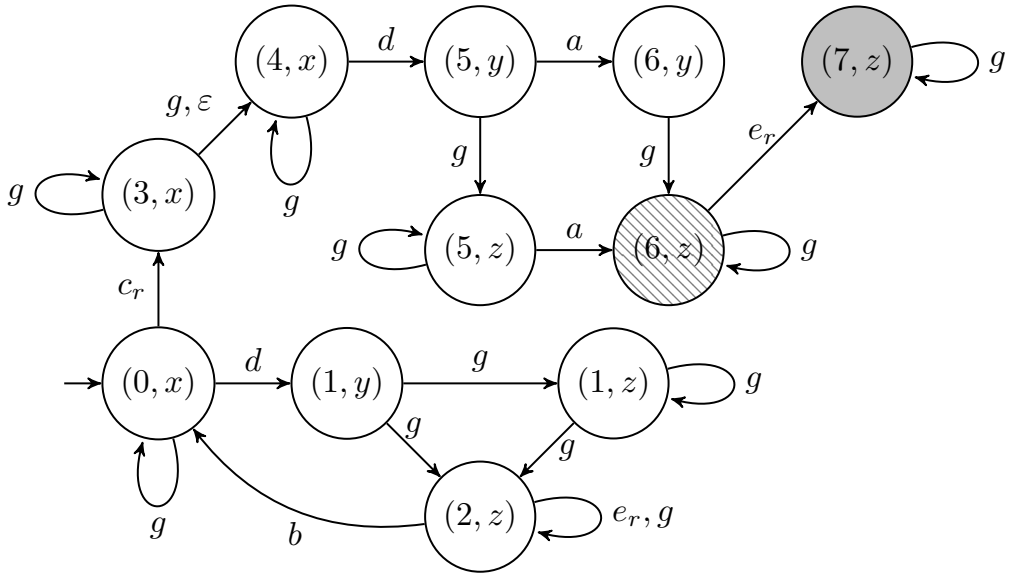


Figure 3.11: Automaton  $\mathcal{T}_{U,R}$ .

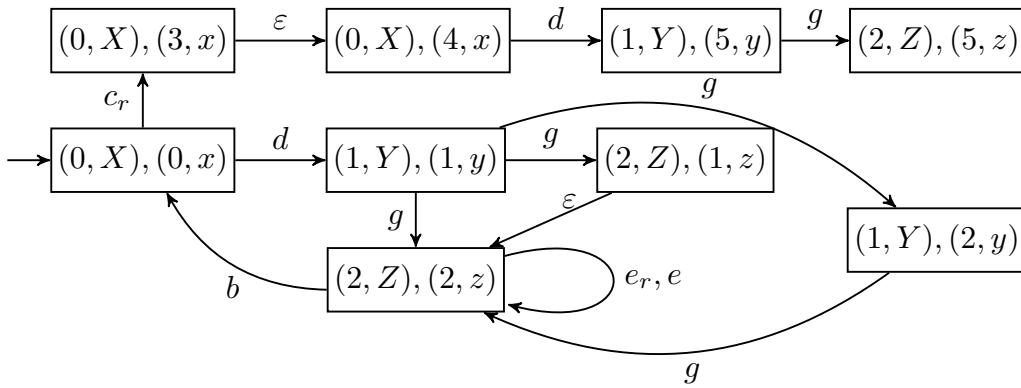


Figure 3.12: Verifier automaton  $\mathcal{V}$ .

## 3.5 Implementation of the Intrusion Detection Module

We show in the sequel that the NA-Safe controllability of  $L_{TA}$  is a necessary and sufficient condition for the existence of the Intrusion Detection Module.

**Theorem 3** *Let  $G$  be the automaton model of the plant with set of unsafe states  $X_{US}$ , controlled by a supervisor  $S$ , and subject to network attacks. Then, there exists an Intrusion Detection Module that is capable of preventing  $G$  from reaching an unsafe state in  $X_{US}$  if, and only if,  $L_{TA}$  is NA-Safe controllable with respect to  $P_o : \Sigma^* \rightarrow \Sigma_o$  and  $X_{US}^{TA}$ .*

*Proof:* ( $\Rightarrow$ ) Assume that  $L_{TA}$  is not NA-safe controllable. Then, according to Definition 7, there exists  $s \in L_{TA}$  such that  $f_{TA}^e(x_{0A}, s) \cap X_{US}^{TA} \neq \emptyset$ , and, for all  $s_1$  and  $s_2$  such that  $s = s_1s_2$ , either (i) there exists  $w \in L_T \cup \bar{L}_s(\mathcal{T}_A)$  such that  $P_o(s_1) = P_o(w)$ , or; (ii)  $s_2 \in \Sigma_{uca}^*$ . In case (i), it is impossible to obtain a module that can force the supervisor to disable the controllable events after observing  $P_o(s_1)$ , because this observation cannot be distinguished from the observation of a trace  $w$  generated by the closed-loop system without the occurrence of attacks ( $w \in L_T$ ), or a trace  $w$  that belongs to the prefix closure of the safe language ( $w \in \bar{L}_s(\mathcal{T}_A)$ ). In case (ii), even if a module forces the supervisor to disable all controllable events, the system can reach an unsafe state, since  $s_2$  is composed only of uncontrollable and/or vulnerable controllable events.

( $\Leftarrow$ ) Assume now that  $L_{TA}$  is NA-Safe controllable, and construct automaton  $\mathcal{T}_S$  as presented in Step 1 of Algorithm 4. Consider an Intrusion Detection Module that checks if the trace observed by the supervisor belongs to  $P_o[L(\mathcal{T}_S)]$  and, when this does not occur, the Intrusion Detection Module forces the supervisor to disable all controllable events. According to Definition 7, every trace  $s \in L_{TA}$  such that  $f_{TA}^e(x_{0A}, s) \cap X_{US}^{TA} \neq \emptyset$  can be partitioned as  $s = s_1s_2$  where, for all  $w \in L_T \cup \bar{L}_s(\mathcal{T}_A)$ ,  $P_o(w) \neq P_o(s_1)$  and  $\Sigma_{ca} \in s_2$ . Therefore, the Intrusion Detection Module will force the supervisor to disable all controllable events after the observation of  $P_o(s_1)$ , since  $P_o(s_1) \notin P_o[L_T \cup \bar{L}_s(\mathcal{T}_A)] = P_o[L(\mathcal{T}_S)]$ . As a consequence, the system will not be able to generate the unsafe trace  $s$ , because  $s_2$  has at least one controllable event that can be disabled by the supervisor, and cannot be enabled by the intruder. Therefore, the Intrusion Detection Module, based on  $\mathcal{T}_S$ , is capable of preventing  $G$  from reaching unsafe states in the presence of network attacks. ■

According to Theorem 3, the Intrusion Detection Module must observe the traces observed by the supervisor, and verify if this trace belongs to  $P_o(L(\mathcal{T}_S))$ . If the observed trace does not belong to  $P_o(L(\mathcal{T}_S))$ , then the Intrusion Detection Module must send an information to the supervisor to disable all controllable events.

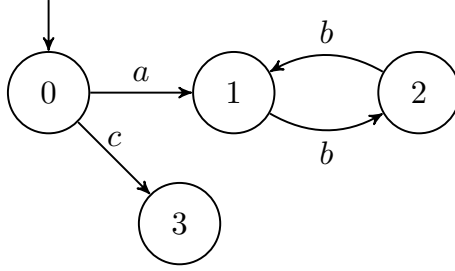


Figure 3.13: Automaton  $G$  of Example 20



Figure 3.14: Supervisor automaton  $H$  of Example 20.

In [25], a Petri net diagnoser is proposed for online failure diagnosis. The Petri net diagnoser is constructed based on the normal behavior of the system, and the occurrence of the failure event is diagnosed when the observed trace does not belong to the observation of the normal language of the system. Thus, since the Intrusion Detection Module must verify if the observed trace belongs to the observation of language  $L(\mathcal{T}_S)$ , the same method proposed in [25] can be used to obtain the Intrusion Detection Module, considering  $L(\mathcal{T}_S)$  as the normal language of the system. We refer to [25] for further details on the implementation of the Intrusion Detection Module as a Petri net.

It is critical for the correct construction of the Intrusion Detection Model that the system be modeled according to assumption A1. If A1 is violated the model of the system subject to an attack may enable a controllable event incorrectly, as shown in Example 20.

**Example 20** Consider automaton  $G$  represented in Figure 3.13. Let  $\Sigma_c = \{a, c\}$  be the set of controllable events, and  $\Sigma_o = \{c\}$  be the set of observable events.

Let automaton  $H$ , presented in Figure 3.14 be a supervisor for  $G$ .

Notice that, in this case, performing a sensor attack with compromised events  $\Sigma_{vs} = \{c\}$  should not disable nor enable any controllable events, since it only alters the observation of  $c$ , and the supervisor does not enable  $c$  in any of its states. However, the attacked plant  $G$ , represented in Figure 3.15, will have a transition to state 3 with event  $\varepsilon$ .

In this case, the closed-loop system under attack  $T_a$  shown in 3.16, has a state  $(3, 0)$  which represents that an event  $c$  has been executed although it should be disabled



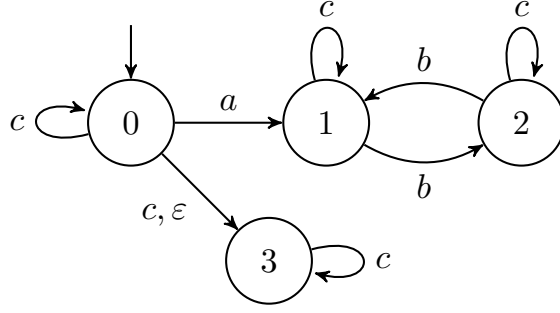


Figure 3.15: Automaton  $G$  subject to sensor attack

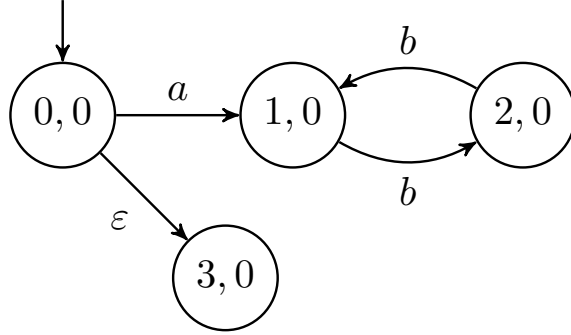


Figure 3.16: Closed-loop system subject to attack  $T_a$ .

by the supervisor.

For any controlled system  $T = G \parallel H$  where set  $\Sigma$  has events controllable and observable, *i.e.*,  $\Sigma_c \cap \Sigma_o \neq \emptyset$ , it is possible to obtain a similar controlled system  $\tilde{T} = \tilde{G} \parallel \tilde{H}$  with disjoint sets of controllable events and observable events.

**Definition 8 (Similar events)** *Two events  $\alpha$  and  $\beta$  are said to be similar if they represent the same physical occurrence.*

**Definition 9 (Similar closed-loop models)** *Two closed-loop models,  $T = G \parallel H$  and  $\tilde{T} = \tilde{G} \parallel \tilde{H}$ , where  $G = (X, \Sigma, f, x_0)$ ,  $H = (X_H, \Sigma, f_H, x_{0H})$ ,  $\tilde{G} = (\tilde{X}, \tilde{\Sigma}, \tilde{f}, \tilde{x}_0)$  and  $\tilde{H} = (\tilde{X}_H, \tilde{\Sigma}, \tilde{f}_H, \tilde{x}_{0H})$ , with the same set of observable events  $\Sigma_o$ , are said to be similar if two conditions are satisfied.*

- i. The two systems have the same observation, *i.e.*,  $P_o(L(T)) = \tilde{P}_o(L(\tilde{T}))$  where  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  and  $\tilde{P}_o : \tilde{\Sigma}^* \rightarrow \Sigma_o^*$ .*
- ii. If a controllable event is enabled after a sequence  $s$  in  $T$  it should have a similar event enabled in  $\tilde{T}$  after a sequence with same projection as  $s$ ,  $\tilde{s}$ , *i.e.*,  $P_o(s) = \tilde{P}_o(\tilde{s})$ .*

A similar system with  $\Sigma_o \cap \Sigma_c = \emptyset$  from any system can be obtained by Algorithm 5.

---

**Algorithm 5** *Computation of automaton  $\tilde{G}$  and  $\tilde{H}$  where  $\tilde{T} = \tilde{G} \parallel \tilde{H}$  models a similar system as system  $T = G \parallel H$  composed by automata  $G$  and  $H$ .*

---

**Input:**

- $G = (X, \Sigma, f, x_0)$  [Plant model]
- $H = (X_H, \Sigma, f_H, x_{0H})$  [Supervisor model]
- $\Sigma_c \subseteq \Sigma$  [Set of controllable events]
- $\Sigma_o \subseteq \Sigma$  [Set of observable events]

**Output:**

- $\tilde{G} = (\tilde{X}, \tilde{\Sigma}, \tilde{f}, \tilde{x}_0)$ . [New model of the plant]
- $\tilde{H} = (\tilde{X}_H, \tilde{\Sigma}, \tilde{f}_H, \tilde{x}_{0H})$ . [New model of the supervisor]
- $\tilde{\Sigma}_c$  [New set of controllable events]
- $\tilde{\Sigma}_o$  [New set of observable events]

1: Define  $\tilde{\Sigma}_c = \{\sigma_c : \forall \sigma \in \Sigma_c\}$

2: Define  $\tilde{\Sigma}_o = \Sigma_o$

3: Define  $\tilde{G}$  as:

3.1: Define  $\tilde{X} = X$

3.2: Update  $\tilde{X}$  by  $(\forall x \in X), (\forall \sigma \in \Gamma_G(x)), \text{ if } [\sigma \in (\Sigma_c \cap \Sigma_o)] \rightarrow \tilde{X} = \tilde{X} \cup \{x_\sigma\}$

3.2: Define  $\tilde{\Sigma} = \Sigma \setminus (\Sigma_c \setminus \Sigma_o) \cup \tilde{\Sigma}_c$

3.3: Define  $\tilde{f}$  as follows:

$\forall x \in X, \forall \sigma \in \Gamma_G(x) :$

if  $\sigma \notin \Sigma_c$  then  $\tilde{f}(x, \sigma) = f(x, \sigma)$

if  $\sigma \in \Sigma_c \setminus \Sigma_o$  then  $\tilde{f}(x, \sigma_c) = f(x, \sigma)$

if  $\sigma \in \Sigma_c \cap \Sigma_o$  then  $\tilde{f}(x, \sigma_c) = x_\sigma$  and  $\tilde{f}(x_\sigma, \sigma) = f(x, \sigma)$

4: Define  $\tilde{H}$  as:

4.1: Define  $\tilde{X}_H = X_H$ .

4.2: Define  $\tilde{f}_H$  as follows:

$$\forall x \in X_H, \forall \sigma \in \Gamma_H(x) :$$

if  $\sigma \notin \Sigma_c$  then  $\tilde{f}_H(x, \sigma) = f_H(x, \sigma)$

if  $\sigma \in \Sigma_c \setminus \Sigma_o$  then  $\tilde{f}_H(x, \sigma_c) = x$

if  $\sigma \in \Sigma_c \cap \Sigma_o$  then  $\tilde{f}_H(x, \sigma_c) = x$  and  $\tilde{f}_H(x, \sigma) = f_H(x, \sigma)$

Notice that, this algorithm represents the partition of all event  $\sigma$  that are controllable and observable into two events such that one is a similar controllable event  $\sigma_c$  but it is not observable, and one is observable  $\sigma$  but not controllable. Notice that, it is also needed to ensure that they always occur together, *i.e.*, there should not be events between  $\sigma_c$  and  $\sigma$ . In order to facilitate notation we also create similar events for events that are controllable and not observable in the original system. Notice that in step 1 of Algorithm 5 we rename events from the original system so any event  $\sigma_c$  is similar to the event  $\sigma$  in the original system.

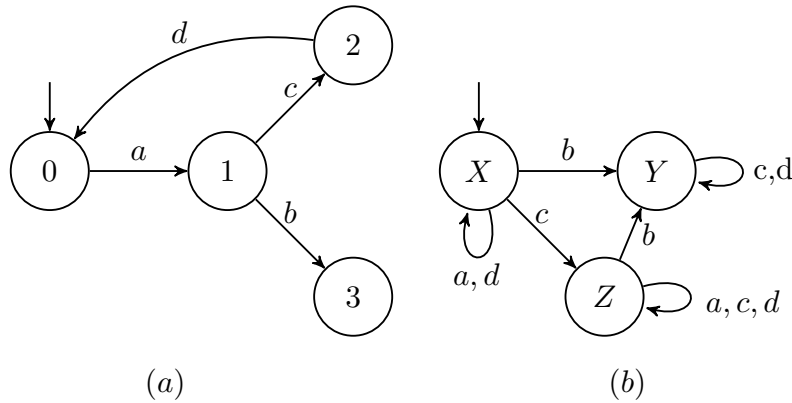


Figure 3.17: Automata  $G$  and  $H$  of Example 21.

**Example 21** Let us consider two automata  $G$  and  $H$  represented in Figure 3.17 (a) and (b), respectively, let  $\Sigma_c = \{a, b\}$  be the set of controllable events, and  $\Sigma_o = \{b, c\}$  be the set of observable events. Following the Algorithm 5 we construct  $\tilde{G}$  as an image of  $G$  where for every event  $\alpha$  that is both controllable and observable we create a new state  $x_\alpha$ , and then separate event  $\alpha$  in its controllable part  $\alpha_c$  and observable part  $\alpha$ . In addition, we rename any controllable event  $\beta$  that are not observable as  $\beta_c$ . The new automata  $\tilde{G}$  and  $\tilde{H}$ , obtained by following Algorithm 5, with inputs  $G, H, \Sigma_c$  and  $\Sigma_o$  are presented in Figure 3.18, where the controllable events are  $\tilde{\Sigma}_c = \{a_c, b_c\}$  and observable events are  $\tilde{\Sigma}_o = \{b, c\}$ .

**Proposition 4** A system  $T = G \parallel H$  composed by automata  $G$  and  $H$ , and a system  $\tilde{T}$  composed by  $\tilde{G}$  and  $\tilde{H}$  generated by Algorithm 5, are similar.

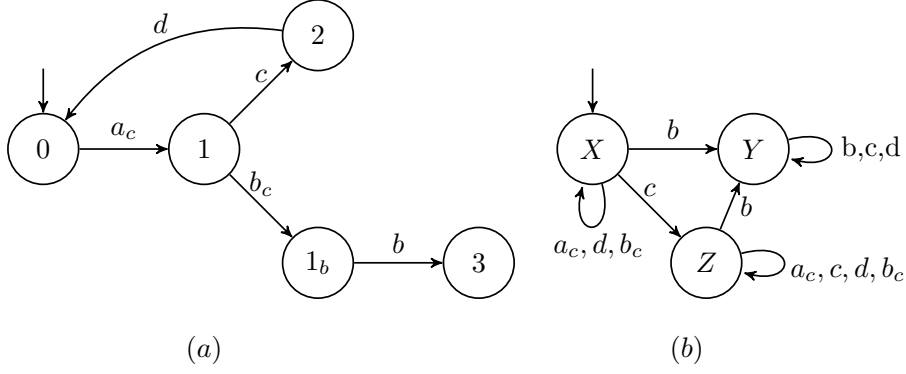


Figure 3.18: Automata  $\tilde{G}$  and  $\tilde{H}$  of Example 21.

*Proof:* The proof is done in two steps. First we prove that the systems are similar if either  $G$  or  $H$  are empty automata, and then we use induction to prove that for non-empty  $G$  and  $H$  the systems are also similar.

Since automata  $G$  and  $H$  have the same event  $\Sigma$ , then if any of them are empty the composition will also be empty. By construction if  $G$  is empty, then  $\tilde{G}$  will be empty, and if  $H$  is empty, then  $\tilde{H}$  will be empty. Since  $\tilde{H}$  and  $\tilde{G}$  are also defined over the same  $\tilde{\Sigma}$  then the composition will also be empty, making the systems similar.

For non-empty  $G$  and  $H$  the proof is by induction in the size of a sequence in the generated language by  $G \parallel H$ .

(i) *Basis step.* If both  $G$  and  $H$  have generated language equal to  $\{\varepsilon\}$  then by construction both  $\tilde{G}$  and  $\tilde{H}$  will also have the generated language equal to  $\{\varepsilon\}$ , and then they will have the same empty observation and will always enable no controllable events, so the systems are similar.

(ii) *Induction hypothesis.* For all  $t \in \Sigma^*$  such that  $|t| \leq n$ , if  $t \in P_o(L(T))$  then  $t \in P_o(L(\tilde{T}))$  and if an event  $\alpha$  is enabled in  $T$  after  $t$  then there is a similar event enabled in  $\tilde{T}$  after a sequence with same projection as  $t$ .

(iii) *Inductive step.* Let us consider a trace  $t_{n+1} = t\sigma \in L(T)$  such that  $|t| = n$ . Then,  $|t_{n+1}| = n + 1$ . According to the induction hypothesis, if  $t \in P_o(L(T))$  then  $t \in \tilde{P}_o(L(\tilde{T}))$  and if an event  $\alpha$  is enabled in  $T$  after  $t$  then there is an similar event enabled in  $\tilde{T}$  after a sequence with same projection as  $t$ , by following a trace  $s\sigma$  such that  $\sigma \in \Sigma$ . The following cases can occur:

(a) If  $\sigma \in \Sigma_c \cap \Sigma_o$ , then by the construction of  $\tilde{G}$  there will be a trace  $t\sigma_c\sigma$  such that  $\sigma_c$  is controllable and  $\sigma$  is observable. In the supervisor on the other hand the occurrence  $\sigma_c$  will be enabled, and then the observable event  $\sigma$  will make the supervisor transit to the original state where the controllable events after an original sequence  $t\sigma$  are enabled. Since  $\sigma_c$  is not observable the observation of the sequence  $t\sigma_c\sigma$  will be  $t\sigma$  as the original trace.

(b) If  $\sigma \in \Sigma_c \setminus \Sigma_o$ , then by the construction of  $\tilde{G}$  there will be a trace  $t\sigma_c$  such

that  $\sigma_c$  is similar to  $\sigma$ , in the supervisor on the other hand the occurrence  $\sigma_c$  will be enabled as a loop and since  $\sigma$  is not observable in other to  $H$  be admissible the event  $\sigma$  cannot make the supervisor transit to a different state so the controllable events after an original sequence  $t\sigma$  will be enabled. In this regard since  $\sigma_c$  is not observable the observation of the sequence  $t\sigma_c$  will be  $t$  as the original trace.

(c) If  $\sigma \notin \Sigma_c$ . by construction of  $\tilde{G}$  and  $\tilde{H}$  the system will also have the same transition  $\sigma$  after sequence  $t$  with the same properties, therefore the system will be similar. ■

Since any system has a similar system such that  $\Sigma_c \cap \Sigma_o = \emptyset$ , *i.e.*, with set of controllable events and set of observable events disjoint, then, in this work, we consider systems in which the sets of controllable events and observable events do not have events in common.

### 3.6 Example

In order to illustrate this work, let us consider an example and obtain the Intrusion Detection Module for the system under MITM attacks. In order to do so, we will use a cat/mouse problem, which consists of a maze divided into rooms with doors, and the events are the passing of a mouse through a door. Each door may have a sensor to detect the passing of the mouse and/or actuators that close or open the door making events associated with these doors controllable. Sensors from different doors may be connected to the same channel, and then can be transmitted with the same observation, also a command from the control may open or close simultaneously more than one door. Thus, distinct doors may have same observation and same control signal. In these cases, there is no need to differentiate events, and then we will represent them by the same event. The maze in this example is shown in Figure 3.19. In the maze there are a cat, which always stays in the same room, a mouse, and food. The control objective is to remotely open or close the doors to allow the mouse to eat the food, and do not reach the same room of the cat. In this example, the set of controllable events is given by  $\Sigma_c = \{a, b, c\}$ , and the set of observable events is given by  $\Sigma_o = \{a, b, d, e\}$ .

This system can be modelled by automaton  $G$  presented in Figure 3.20 where each state represents the room that the mouse has reached. Then, if the system is in state 7, the mouse has reached the room where the cat is. Since this case should be avoided, we will consider the set of unsafe states as  $X_{US} = \{7\}$ . Notice that in this scenario there are two ways to feed the mouse. The mouse can either go through room 1 executing sequence  $dc$ , or through rooms 3 and 4, executing sequence  $adb$ , and then, after eating the food, the mouse can transit through rooms

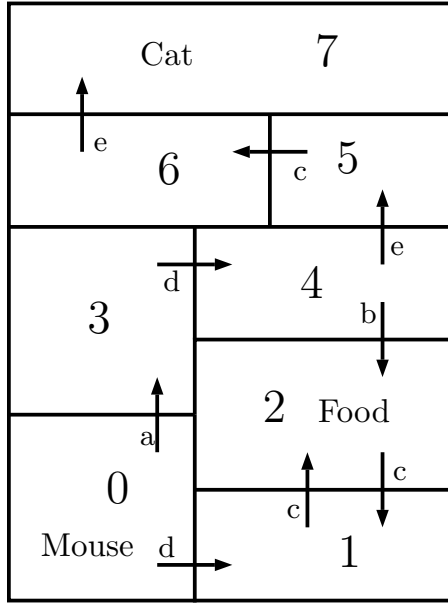


Figure 3.19: Initial house position for cat/mouse example.

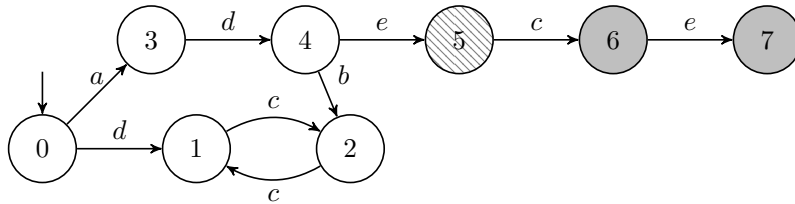


Figure 3.20: Automaton model for mouse/cat problem  $G$ .

1 and 2 executing event  $c$ .

Since the control objective is to feed the mouse, and avoid reaching room 7, we could just specify that the mouse can pass to room 2 and make a realization of a supervisor that enables all the ways to reach this room, through rooms 3 and 4, or through room 1. Notice that disabling the occurrence of event  $c$  after the observation of event  $e$  would avoid the mouse from reaching room 7. However, since event  $e$  is uncontrollable this would represent a risk, because the mouse would be able to move to room 5 and will not be able to return. In this regard, a more appropriate supervisor  $H$ , represented in Figure 3.21, would only enable the mouse to move from room 1 to room 2 where the food is, and then, there is no need to disable its return to room 1. Notice that this supervisor is admissible, since whenever an uncontrollable event is active in the plant it is enabled in the supervisor. Thus, it is important to enable uncontrollable events in every state of  $H$ .

In order to implement a Intrusion Detection Module, assumption A1 must be valid. Notice that in this case this assumption is violated, since events  $a$  and  $b$  are controllable and observable. Then, we can follow Algorithm 5 in order to ensure that sets of controllable and observable events are disjoint. We generate the sim-

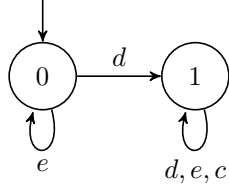


Figure 3.21: Supervisor  $H$  for cat/mouse example.

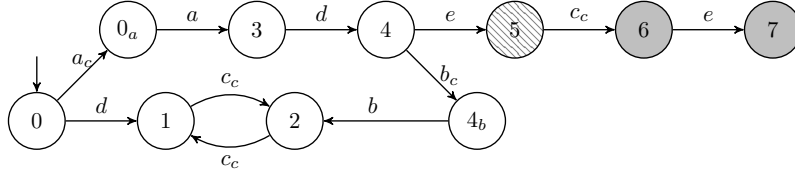


Figure 3.22: Similar plant system  $\tilde{G}$  from  $G$  in cat/mouse example.

ilar system  $\tilde{T}$  formed by automata  $\tilde{G}$  and  $\tilde{H}$  represented in Figures 3.22 and 3.23 respectively, with set of controllable events given by  $\tilde{\Sigma}_c = \{a_c, b_c, c_c\}$  and the set of observable events is given by  $\tilde{\Sigma}_o = \{a, b, d, e\}$ .

We present the closed-loop system  $\tilde{T} = \tilde{G} \parallel \tilde{H}$  in Figure 3.24.

We consider vulnerable communications channels and then model MITM attacks in this system. Let us consider these vulnerable sensor channels transmit the set of events  $\Sigma_{vs} = \{a\}$ . Thus, the attacked system  $\mathcal{G}_A$  is represented in Figure 3.25.

Let us also consider vulnerable supervisory control channels that enables set of events  $\Sigma_{va} = \{a_c\}$ . The supervisor under attack  $\mathcal{H}_A$  is represented in Figure 3.26.

If this supervisory control operates without an Intrusion Detection Module, the resulting attacked closed-loop system  $\mathcal{T}_A$  would be represented in Figure 3.27. Notice that the state  $(7, 1)$  represents that the mouse has reached the room where the cat is, *i.e.*, the system achieved unsafe state 7.

In order to verify if it is possible to avoid reaching the unsafe state, we generate the verifier proposed in 3.4. Following Algorithm 4 we define  $X_{US}^{TA} = \{(7, 1)\}$  as the set of marked states of  $\mathcal{T}_A$  and then generate  $\mathcal{T}_U$  as  $CoAc(\mathcal{T}_A)$ , for this case  $\mathcal{T}_U$  is represented in Figure 3.28.

We then generate  $\mathcal{T}_S$ . Notice that the automaton  $\mathcal{T}'_S = T$ , and then the automaton  $\mathcal{T}_S$  is also equal to  $T$ , represented in Figure 3.24.

With  $\mathcal{T}_S$  and  $\mathcal{T}_U$  defined, we rename unobservable events in  $\mathcal{T}_U$  generating au-

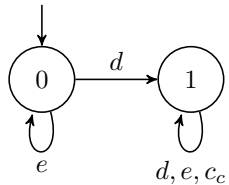


Figure 3.23: Similar supervisor system  $\tilde{H}$  from  $H$  in cat/mouse example.

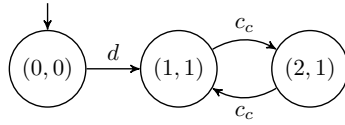


Figure 3.24: Closed-loop system  $T$  for cat/mouse example.

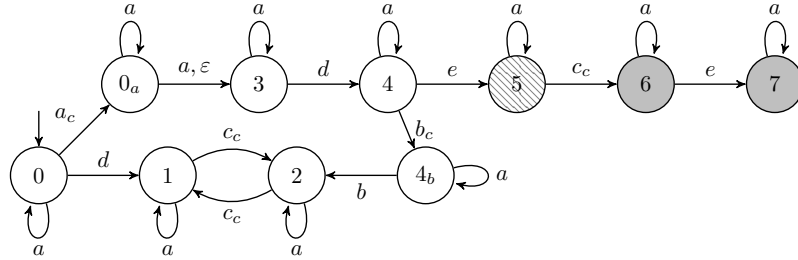


Figure 3.25: Plant model under attack  $\mathcal{G}_A$  for cat/mouse example.

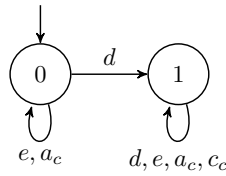


Figure 3.26: Supervisor model under attack  $\mathcal{H}_A$  for cat/mouse example.

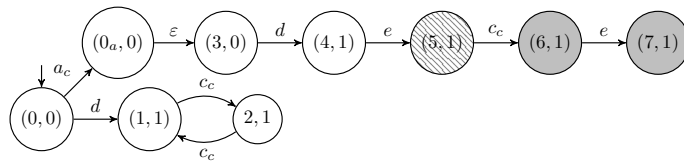


Figure 3.27: Closed-loop system model under attack  $\mathcal{T}_A$  for cat/mouse example.

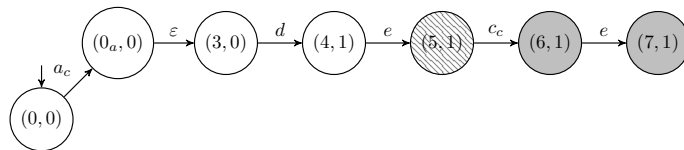


Figure 3.28: Closed-loop system model unsafe part  $\mathcal{T}_U$  for cat/mouse example.



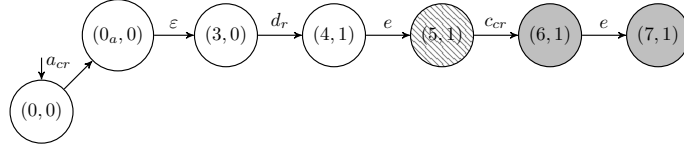


Figure 3.29: Renamed model of unsafe part  $\mathcal{T}_{U,R}$  for cat/mouse example.

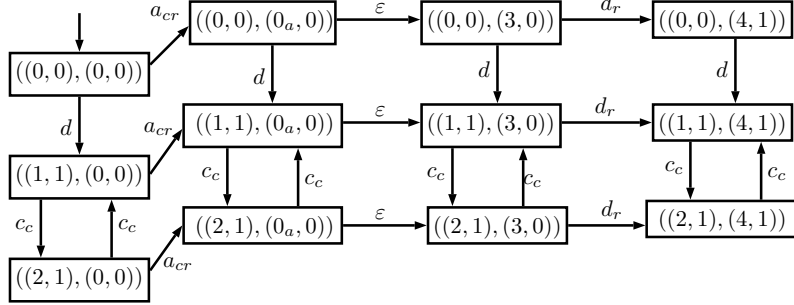


Figure 3.30: Verifier model  $\mathcal{V}$  for cat/mouse example

tomaton  $\mathcal{T}_{U,R}$  represented in Figure 3.29.

The verifier is then given by  $\mathcal{V} = \mathcal{T}_S \parallel \mathcal{T}_{U,R}$ , represented in Figure 3.30. In this case the unsafe region is given by  $US_R = \{(7, 1), (6, 1)\}$  and the unsafe boundary is given by  $US_b = \{(5, 1)\}$ .

Notice that, the verifier does not reach a state on the unsafe boundary nor a state in the unsafe region, then, an Intrusion Detection Module can be successfully implemented in this plant, avoiding the attacker to lead the mouse to reach room 7.

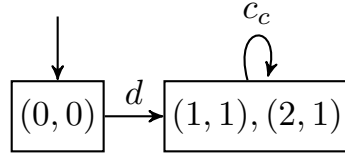


Figure 3.31: Observer of safe part for cat/mouse example.

The Intrusion Detection Module would be implemented by using the safe part  $T_S$  depicted in Figure 3.24, and then by taking the observer of this automaton shown in Figure 3.31. If the system generates an observed trace that is not represented in  $T_S$ , the Intrusion Detection Module blocks all controllable events. Notice that, in this example the Intrusion Detection Module may not see an occurrence of event  $a$ , since it is attacked. The mouse would, then, be able to go to room 5, however, once observed event  $e$  the module would block all controllable events preventing  $c$  from being executed, so the mouse would never reach room 7.

# Chapter 4

## Conclusion and Future work

In this work, we propose the implementation of an Intrusion Detection Module that is capable of preventing damages caused by attacks in sensor and/or control communication channels in networked supervisory control systems.

In order to implement this device, models of the plant subject to sensor channel attacks, and of the supervisor subject to control channel attacks are needed. We consider man-in-the-middle attacks (MITM) for these models, and in Sections 3.1 and 3.2 we present a way to properly modify the plant and supervisor in order to generate these models subject to this type of attack. We also present, in Section 3.3, a way to properly combine these models in order to construct the model of the closed-loop system subject to attack.

In Section 3.4 we present the definition of NA-Safe controllability, which is a property of the language of a DES subject to attacks, that leads to a necessary and sufficient condition for the existence of the Intrusion Detection Module. We also propose an algorithm to verify this property.

In Section 3.5 we refer to [25], and indicate a way to use the authors result in order to implement the Intrusion Detection Module proposed in this work in an online application. Also in Section 3.5 we define similar systems and propose an algorithm to generate a similar system with disjoint sets of controllable and observable events in order to relax assumption A1.

In Section 3.6 we presented an example to illustrate the use of the Intrusion Detection Module, where it is shown that, without the module, the system would be unsafe after an attack, *i.e.*, if a malicious agent performs a successful attack in the system, it could get the system to an unsafe state. However, after the implementation of the Intrusion Detection Module controllable events of the system would be disabled preventing unsafe states to be reached. In this regard the Intrusion Detection Module may prevent damage to real systems under attack.

In summary this work contributes with the literature by:

- Generating the plant model under MITM attack in the sensor channel.
- Generating the supervisor model under MITM attack in the supervisory control channel.
- Defining NA-Safe Controllability, which ensures the possibility to create an Intrusion Detection Module.
- Creating a verification algorithm to ensure that a system under attack is NA-Safe Controllable.
- Indicating how to implement the Intrusion Detection Module.

### **Future Works**

CPS security is a relatively new area of study, in this regard the continuation of this work may cover various different areas. An approach as a future work would be to study other types of network based attacks and/or how to prevent or identify them before they reach the supervisor or plant improving the communication system. This work can also develop a DES approach to improve the Intrusion Detection Module so it may be even more permissive and just block the system before the last controllable event out of the unsafe boundary.

Moreover, another way to improve the Intrusion Detection Module would be to study self-healing, *i.e.*, return to the normal operation of the system after an attack. Notice that in this implementation of the Intrusion Detection Module the system is blocked once an attack is detected. However, in some systems, it is possible to enable some events in order to recover from the attack and return to the specification.

# Bibliography

- [1] SHI, J., WAN, J., YAN, H., et al. “A survey of cyber-physical systems”. In: *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pp. 1–6. IEEE, 2011.
- [2] BAHETI, R., GILL, H. “Cyber-physical Systems”. In: Samad, T., Annaswamy, A. (Eds.), *The Impact of Control Technology*, 2011.
- [3] MO, Y., KIM, T. H.-J., BRANCIK, K., et al. “Cyber-physical security of a smart grid infrastructure”, *Proceedings of the IEEE*, v. 100, n. 1, pp. 195–209, 2012.
- [4] MO, Y., SINOPOLI, B. “False data injection attacks in control systems”. In: *Preprints of the 1st Workshop on Secure Control Systems*, pp. 1–6, 2010.
- [5] TEIXEIRA, A., AMIN, S., SANDBERG, H., et al. “Cyber security analysis of state estimators in electric power systems”. In: *49th IEEE Conference on Decision and Control (CDC)*, pp. 5991–5998, 2010.
- [6] FAWZI, H., TABUADA, P., DIGGAVI, S. “Secure state-estimation for dynamical systems under active adversaries”. In: *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 337–344, Sept 2011.
- [7] SMITH, R. S. “A decoupled feedback structure for covertly appropriating networked control systems”, *IFAC Proceedings Volumes*, v. 44, n. 1, pp. 90–95, 2011.
- [8] SUNDARAM, S., HADJICOSTIS, C. N. “Distributed function calculation via linear iterative strategies in the presence of malicious agents”, *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1495–1508, 2011.
- [9] AMIN, S., LITRICO, X., SASTRY, S., et al. “Cyber Security of Water SCADA Systems - Part I: Analysis and Experimentation of Stealthy Deception Attacks”, *IEEE Transactions on Control Systems Technology*, v. 21, n. 5,

pp. 1963–1970, Sept 2013. ISSN: 1063-6536. doi: 10.1109/TCST.2012.2211873.

- [10] PASQUALETTI, F., DÖRFLER, F., BULLO, F. “Attack detection and identification in cyber-physical systems”, *IEEE Transactions on Automatic Control*, v. 58, n. 11, pp. 2715–2729, 2013.
- [11] FAWZI, H., TABUADA, P., DIGGAVI, S. “Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks”, *IEEE Transactions on Automatic Control*, v. 59, n. 6, pp. 1454–1467, June 2014. ISSN: 0018-9286. doi: 10.1109/TAC.2014.2303233.
- [12] ZHANG, H., CHENG, P., WU, J., et al. “Online deception attack against remote state estimation”, *IFAC Proceedings Volumes*, v. 47, n. 3, pp. 128–133, 2014.
- [13] THORSLEY, D., TENEKETZIS, D. “Intrusion detection in controlled discrete event systems”. In: *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 6047–6054. IEEE, 2006.
- [14] CARVALHO, L. K., WU, Y.-C., KWONG, R., et al. “Detection and prevention of actuator enablement attacks in supervisory control systems”. In: *2016 13th International Workshop on Discrete Event Systems (WODES)*, pp. 298–305. IEEE, 2016.
- [15] KUMAR, V., SRIVASTAVA, J., LAZAREVIC, A. *Managing cyber threats*. Springer, 2005.
- [16] COMER, D. *Computer networks and internets*. Pearson/Prentice Hall, 2009.
- [17] LIMA, P. M., ALVES, M. V. S., MOREIRA, M. V., et al. “Security Against Network Attacks in Supervisory Control Systems”. In: *The 20th World Congress of the International Federation of Automatic Control (IFAC)*, accepted for publication, 2017.
- [18] ZAD, S., KWONG, R., WONHAM, W. “Fault diagnosis in discrete-event systems: framework and model reduction”, *IEEE Trans. on Automatic Control*, v. 48, n. 7, pp. 1199–1212, 2003.
- [19] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. “Coordinated decentralized protocols for failure diagnosis of discrete event systems”. In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, v. 4, pp. 3763–3768. IEEE, 1998.

- [20] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. “Failure diagnosis using discrete-event models”, *IEEE transactions on control systems technology*, v. 4, n. 2, pp. 105–124, 1996.
- [21] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. “Diagnosability of discrete-event systems”, *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.
- [22] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. “Generalized robust diagnosability of discrete event systems”, *IFAC Proceedings Volumes*, v. 44, n. 1, pp. 8737–8742, 2011.
- [23] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V. “Robust diagnosis of discrete event systems against intermittent loss of observations”, *Automatica*, v. 48, n. 9, pp. 2068–2078, 2012.
- [24] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., et al. “Robust diagnosis of discrete-event systems against permanent loss of observations”, *Automatica*, v. 49, n. 1, pp. 223–231, 2013.
- [25] CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. “A Petri net diagnoser for discrete event systems modeled by finite state automata”, *IEEE Transactions on Automatic Control*, v. 60, n. 1, pp. 59–71, 2015.
- [26] LIMA, S. T., BASILIO, J. C., LAFORTUNE, S., et al. “Robust diagnosis of discrete-event systems subject to permanent sensor failures”, *IFAC Proceedings Volumes*, v. 43, n. 12, pp. 90–97, 2010.
- [27] JESUS, T. C., MOREIRA, M. V., BASILIO, J. C., et al. “Diagnóstico de falhas em tempo real de sistemas a eventos discretos descritos por autômatos finitos”. In: *Proceedings of the XVIII Congresso Brasileiro de Automática, Bonito*, pp. 712–719, 2010.
- [28] TOMOLA, J. H., DE OLIVEIRA CABRAL, F. G., CARVALHO, L. K., et al. “Robust Disjunctive-Codiagnosability of Discrete-Event Systems Against Permanent Loss of Observations”, *IEEE Transactions on Automatic Control*, 2016.
- [29] SANTORO, L. P. M., MOREIRA, M. V., BASILIO, J. C. “Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers.” v. 77, pp. 93–102, 2017.
- [30] MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. “Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems“

Versus "Decentralized Failure Diagnosis of Discrete Event Systems": A Critical Appraisal", *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2016.

- [31] CASSANDRAS, CHRISTOS GLAFORTUNE, S. *Introduction to discrete event systems*. Springer, 2008.
- [32] RAMADGE, P. J., WONHAM, W. M. "The control of discrete event systems". In: *Proc. IEEE, Special Issue on Discrete Event Systems*, v. 77, pp. 81–98, 1989.
- [33] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Revista Controle & Automação*, v. 21, n. 5, pp. 510–533, 2010.
- [34] DE QUEIROZ, M. H., CURY, J. E. "Modular supervisory control of large scale discrete event systems". In: *Discrete Event Systems*, Springer, pp. 103–110, 2000.
- [35] HILL, R., TILBURY, D., LAFORTUNE, S. "Modular supervisory control with equivalence-based conflict resolution". In: *American Control Conference, 2008*, pp. 491–498. IEEE, 2008.
- [36] PENA, P. N., CURY, J. E., LAFORTUNE, S. "Verification of nonconflict of supervisors using abstractions", *IEEE Transactions on Automatic Control*, v. 54, n. 12, pp. 2803–2815, 2009.
- [37] HILL, R. C., TILBURY, D. M. "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction". In: *Discrete Event Systems, 2006 8th International Workshop on*, pp. 399–406. IEEE, 2006.
- [38] SCHMIDT, K., MOOR, T., PERK, S. "Nonblocking hierarchical control of decentralized discrete event systems", *IEEE Transactions on Automatic Control*, v. 53, n. 10, pp. 2252–2265, 2008.
- [39] DA CUNHA, A. E. C., CURY, J. E. R. "Hierarchical supervisory control based on discrete event systems with flexible marking", *IEEE Transactions on Automatic Control*, v. 52, n. 12, pp. 2242–2253, 2007.
- [40] DETKEN, K.-O., GENZEL, C.-H., RUDOLPH, D. C., et al. "Integrity Protection in a Smart Grid Environment for Wireless Access of Smart Meters". In: *The 2nd IEEE International Symposium on Wireless Systems within*

*the Conferences on Intelligent Data Acquisition and Advanced Computing Systems*, pp. 79–86. IEEE, 2014.

- [41] GIANI, A., BITAR, E., GARCIA, M., et al. “Smart Grid Data Integrity Attacks”, *IEEE Transactions on Smart Grid*, v. 4, n. 3, pp. 1244–1253, 2013.
- [42] RAWAT, D. B., BAJRACHARYA, C. “Cyber security for smart grid systems: Status, challenges and perspectives”. In: *SoutheastCon 2015*, pp. 1–6. IEEE, 2015.
- [43] YANG, T. C. “Networked control system: a brief survey”, *IEE Proceedings-Control Theory and Applications*, v. 153, n. 4, pp. 403–412, 2006.
- [44] GREENE, S. S. *Security policies and procedures*. New Jersey: Pearson Education, 2006.
- [45] BELLOVIN, A. E. K. E. “A Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise”. In: *Proceedings of the First ACM Conference on Computer and Communications Security*, pp. 244–250, 1993.
- [46] CONTI, M., DRAGONI, N., LESYK, V. “A Survey of Man In The Middle Attacks”, *IEEE Communications Surveys & Tutorials*, v. 18, n. 3, pp. 2027–2051, 2016.
- [47] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. “Polynomial time verification of decentralized diagnosability of discrete event systems”, *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684, 2011.