



## SAFETY OF FEEDBACK DISCRETE EVENT SYSTEMS WITH UNKNOWN CONTROLLERS

Tiago Cardoso França

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

Orientador: Marcos Vicente de Brito Moreira

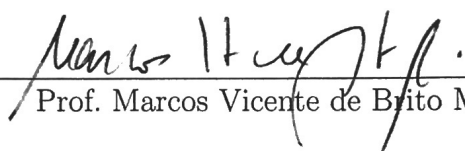
Rio de Janeiro  
Junho de 2017

SAFETY OF FEEDBACK DISCRETE EVENT SYSTEMS WITH UNKNOWN  
CONTROLLERS

Tiago Cardoso França

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. Antônio Eduardo Carrilho da Cunha, D.Sc.

  
Prof. Lilian Kawakami Carvalho, D.Sc.

RIO DE JANEIRO, RJ – BRASIL  
JUNHO DE 2017

França, Tiago Cardoso

Safety of feedback discrete event systems with unknown controllers/Tiago Cardoso França. – Rio de Janeiro: UFRJ/COPPE, 2017.

XI, 70 p.: il.; 29,7cm.

Orientador: Marcos Vicente de Brito Moreira

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

Referências Bibliográficas: p. 58 – 63.

1. Safety Device. 2. Safety Level. 3. Safety on Discrete Event Systems. 4. Cyber Physical Systems. 5. Supervisory Control. I. Moreira, Marcos Vicente de Brito. 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 DE SISTEMAS A EVENTOS DISCRETOS REALIMENTADOS COM CONTROLADORES DESCONHECIDOS

Tiago Cardoso França

Junho/2017

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

Apresenta-se, nesta dissertação, um dispositivo de segurança que lida com ataques cibernéticos e implementação incorreta de controladores em sistemas a eventos discretos. Um ataque cibernético modifica o comportamento do sistema, e pode levá-lo a estados inseguros, isto é, estados que podem causar danos aos componentes do sistema ou aos seus operadores. Além disso, uma implementação ou um projeto de controlador incorreto também pode levar o sistema para estados inseguros. Neste trabalho, considera-se que as modificações são realizadas na lógica de controle implementada, o que altera o comportamento do sistema e faz com que o controlador torne-se desconhecido. Tal dispositivo de segurança é desenvolvido para prevenir que a planta controlada modificada atinja estados inseguros. Entretanto, em alguns casos, o dispositivo de segurança pode bloquear comportamentos seguros do sistema. Portanto, é apresentado um algoritmo de verificação do nível de segurança da interação entre o dispositivo e a planta. Os resultados apresentados neste trabalho são ilustrados em uma planta mecatrônica do Laboratório de Controle e Automação da Universidade Federal do Rio de Janeiro (UFRJ).



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

## SAFETY OF FEEDBACK DISCRETE EVENT SYSTEMS WITH UNKNOWN CONTROLLERS

Tiago Cardoso França

June/2017

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

In this work, we present a safety device that deals with the problem of cyber-attacks and incorrect implementation of controllers on discrete event systems. A cyber-attack modifies the behavior of the system, and may lead it to unsafe states, *i.e.*, states that may cause damage to the system components or its operator. In addition, an incorrect implementation or design of controllers may also lead the system to unsafe states. In this work, we assume those changes in the implemented logic of the controller, which modify the behavior of the system, and thus we assume the controller to be unknown. The safety device is designed to prevent the modified controlled plant from reaching unsafe states. However, in some cases, the device may block safe behaviors of the system. For this reason, we present an algorithm to verify the safety level of the interaction between the device and the plant. The results shown in this work are illustrated by a case study in a mechatronic plant in the Control and Automation Laboratory at the Federal University of Rio de Janeiro.

# Contents

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Preliminaries</b>	<b>4</b>
2.1 Languages . . . . .	4
2.1.1 Notations and definitions . . . . .	4
2.1.2 Language operations . . . . .	5
2.2 Automata . . . . .	8
2.2.1 Operations on automata . . . . .	9
2.2.2 Automata under partial observation of events . . . . .	14
2.3 Controlled discrete event system . . . . .	17
2.4 Supervisory control theory . . . . .	18
2.4.1 Supervisor with partial controllability . . . . .	21
2.4.2 Supervisor under partial observation . . . . .	26
2.5 Final comments . . . . .	27
<b>3 Computation of the safety device</b>	<b>29</b>
3.1 Formulation of the problem . . . . .	29
3.2 Computation of a safety device realization . . . . .	34
3.3 Final comments . . . . .	40
<b>4 Implementation of the safety device in a mechatronic system</b>	<b>42</b>
4.1 Mechatronic system . . . . .	42
4.2 Plant model . . . . .	42
4.3 Implementation and final comments . . . . .	51
<b>5 Conclusions and future works</b>	<b>57</b>
<b>Bibliography</b>	<b>58</b>



# List of Figures

2.1	State transition diagram of automaton $G$ of Example 2.5. . . . .	9
2.2	Automaton $G$ , Example 2.6. . . . .	12
2.3	$Ac(G)$ , $CoAc(G)$ and $Trim(G)$ , respectively, of automaton $G$ depicted in Figure 2.2. . . . .	12
2.4	Automata $G_1$ and $G_2$ , Example 2.7. . . . .	14
2.5	Resulting automata after product and parallel composition of $G_1$ and $G_2$ , Example 2.7. . . . .	15
2.6	Automata $G$ with unobservable events and $Obs(G, \Sigma_o)$ respectively, Example 2.8. . . . .	16
2.7	Interaction between all four elements. . . . .	17
2.8	Four basic elements of a feedback discrete event system. . . . .	17
2.9	Behavior of a discrete event system. . . . .	18
2.10	Block diagram of a plant being controlled. . . . .	18
2.11	Block diagram of the interaction of plant, controller and supervisor. . . . .	19
2.12	Block diagram of the interaction supervisor and a controlled plant. . . . .	19
2.13	DES $G$ , Example 2.9. . . . .	21
2.14	Input automata for Algorithm 2.2 in Example 2.11. . . . .	24
2.15	Steps of Algorithm 2.2 in Example 2.11. . . . .	24
2.16	Automaton $H_{aug}$ from Example 2.12. . . . .	25
2.17	Automaton $G$ from Example 2.13. . . . .	26
2.18	Architecture of modular supervisory control. . . . .	28
3.1	Working tracks of robots and robot behavior automata of Example 3.1. . . . .	30
3.2	Independent automata models of Example 3.1. . . . .	30
3.3	Parallel composition of the two automata that model the behavior of the robots, described in Example 3.1. . . . .	31
3.4	Automaton of supervised plant behavior, described in Example 3.2. . . . .	32
3.5	Automaton of supervised plant behavior after cyber-attack, described in Example 3.2. . . . .	32
3.6	Automaton of a realization of a possible supervisor for the plant in Example 3.2. . . . .	33

3.7	Architecture of interaction between plant, supervisor and safety device.	34
3.8	Automaton $G$ of Example 3.3. . . . .	36
3.9	Final steps of Algorithm 3.1 in Example 3.3. . . . .	36
3.10	Automata $G_1$ , $G_2$ , $G_3$ , and $G_4$ used in Example 3.4. . . . .	41
3.11	Monitors $M_1$ , $M_2$ , $M_3$ , and $M_4$ of $G_1$ , $G_2$ , $G_3$ , and $G_4$ , respectively, as presented in Example 3.4. . . . .	41
4.1	Module division of Cube Assembly plant. . . . .	43
4.2	Processing module of Cube Assembly plant. . . . .	43
4.3	Pneumatic actuators of module 2 of the plant and robotic arm vacuum gripper models. . . . .	44
4.4	Automaton model of the robotic arm rotation. . . . .	45
4.5	State transition diagram of safety device realization $W$ , obtained by Algorithm 3.1 on the automaton of the robotic arm rotation subsys- tem, as described in Example 4.1. . . . .	50
4.6	Parallel composition of automata depicted in Figures 4.3d and 4.3e, as described in Example 4.2. . . . .	50
4.7	State transition diagram of safety device realization $W$ , obtained by Algorithm 3.1 on the automaton of the press horizontal and vertical pneumatic actuators subsystem, as described in Example 4.2. . . . .	51
4.8	Parallel composition of automata depicted in Figures 4.3a, 4.3b, and 4.4, as described in Example 4.3. . . . .	52
4.9	Automaton of a state splitting procedure on the automaton depicted in Figure 4.8, and the unsafe transitions $ArmForward$ from state ( $Re, Re, 4$ ), and $ArmBack$ from state ( $Re, Ex, 4$ ), as described in Example 4.3. . . . .	53
4.10	State transition diagram of safety device realization $W$ , obtained by Algorithm 3.1 on the automaton of the robotic arm horizontal and vertical pneumatic actuators, and robotic arm rotation subsystem, as described in Example 4.3. . . . .	54
A.1	Initialization module of Ladder logic of Example 4.4. . . . .	64
A.2	Events module of Ladder logic of Example 4.4. . . . .	64
A.3	Condition module of Ladder logic of Example 4.4. . . . .	65
A.4	Condition module of Ladder logic of Example 4.4. . . . .	66
A.5	Dynamics module of Ladder logic of Example 4.4. . . . .	67
A.6	Dynamics module of Ladder logic of Example 4.4. . . . .	68
A.7	Dynamics module of Ladder logic of Example 4.4. . . . .	69
A.8	Safety device implemented in Ladder logic, for Example 4.4. . . . .	70

A.9 Action module of Ladder logic with the safety device restrictions of  
Example 4.4. . . . . 70

# List of Tables

4.1	Physical meaning of state transition labels of the automata in Figure 4.3. . . . .	45
4.2	Physical meaning of states of the automata in Figure 4.4. . . . .	46
4.3	Physical meaning of state transition label abbreviations of the automata in Figure 4.4. . . . .	46
4.4	Unsafe state combinations of the plant. . . . .	47
4.5	Unsafe transitions of the plant. . . . .	49

# Chapter 1

## Introduction

In a certain level of abstraction, systems can be modeled as discrete event systems (DES) [1, 2], whose state space is discrete, and whose evolution is event-driven [2]. Those DESs are controlled by discrete event controllers (DEC), that operate by processing the signals received from sensors, and commanding signals to actuators. In order to synthesize a DEC, there are three main approaches in the literature [3]: *(i)* logic controller approach [4, 5], *(ii)* controlled behavior approach [6–8], and *(iii)* supervisory control approach [9–12]. Using the *(i)* logic controller approach, the designer synthesizes the controller by setting input and output variables to read the actual state of the system, and command actuators in such a way to achieve the desired controlled behavior. This approach is suitable to small systems and is based on the experience of the designer, requiring simulations for code validation. In order to synthesize the DEC, the *(ii)* controlled behavior approach is based on the model of the desired behavior for the controlled plant. In this approach, the controller is obtained by using an algorithm based on the controlled plant model. The *(iii)* supervisory control approach is based on algorithms that use the controlled plant model, and the objective is to design a supervisor that only disables events, *i.e.*, actuators of the plant, in order to satisfy the specified behavior, and thus, the supervisory control approach assumes that the plant generates all events. There are several examples of DESs that can be controlled using the aforementioned theories, such as assembling stations, chemical processes, and telephony systems. Regardless the approach used for designing the controller, it is important to assure the security of the system, *i.e.*, prevent that attacks or control logic modifications cause damages to the system or its operators.

Security is an important topic of research, and it can be categorized in two main classes [13]: *(i)* information flow from the system to the intruder; and *(ii)* capabilities of the intruders. These two categories are included in the area of cyber physical systems. Information flow from the system to the intruder is related with opacity and aims to hide the behavior of the system to intruders [14–23]. Capability



of the intruders is related to cyber-attacks and aims to mitigate the damage that an intruder can cause to the system [24–26]. Cyber-attacks change the controlled system behavior by modifying the communication between plant and controller, which makes the system behavior to be different from the expected one, causing damages to the system components or to its operators, *i.e.*, the system reaches unsafe states.

In this work, we deal with the safety problem [27, 28], which is a critical property of cyber physical systems that needs to be ensured, and safety violation is related to the incorrect implementation or design of controllers. Notice that, the incorrect implementation of a controller has the same effect as an intrusion, since it also changes the controlled system behavior.

Several works have been proposed in the area of cyber physical systems [29–37]. Recently, a DES approach to deal with cyber system security has been presented [24–26]. In these works, cyber-attacks are considered in the communication channels between plant, supervisor, actuators, and sensors. Moreover, those works do not consider the effects of a cyber-attack that can change the control logic, and consider the model of the plant and controller to be known.

In this work, the main objective is to deal with safety against incorrect implementation or design of controllers and cyber-attacks, which may completely change the control logic programmed. Notice that, this kind of attack is different from the attacks considered in [24–26], where the supervisor is not attacked and is known. Since, we assume that the controller can be attacked, its behavior can be completely changed by the intruder, being, therefore, unknown to the security device. Thus, the safety device must not block the normal behaviors programmed by the user in the unknown implemented controller. In this work, we show how to synthesize a safety device that is capable of preventing the system from reaching unsafe states, independently of the control logic implemented.

From the perspective of preventing the system from reaching unsafe states, several works in the supervisory control theory has been proposed [9, 38–45]. However, in these works it is considered that the control logic is known and that it cannot be changed by a cyber-attack. Thus, despite the fact that the safety device to be presented in this work is a supervisor with anti-permissive control policy [46, 47], it deals with a different approach from those in the literature. Moreover, safety levels are defined based on the interaction of the safety device with the plant, and which behaviors it may block. In addition, we present the implementation of a safety device on a mechatronic plant of the Control and Automation Laboratory (LCA) at the Federal University of Rio de Janeiro (UFRJ).

This work is organized as follows. In Chapter 2, we present all theoretical concepts used for the development of this work. Then, in Chapter 3, we present the

problem to be solved and the contributions of this work, such as the algorithm to synthesize the safety device and the algorithm for verification of safety levels. In Chapter 4, we present the mechatronic plant, synthesize a safety device and show the results of the implementation of the safety device in the system. Finally, in Chapter 5, we present the final considerations and future works.

# Chapter 2

## Preliminaries

A Discrete Event System (DES) is a dynamic system whose state space is a discrete set and the state transition mechanism is event driven. Any system can be modeled as a DES considering a higher level of abstraction. Also, a DES is a dynamic system that evolves according to event occurrences, and thus, it is needed to formalize a structure to describe this class of systems. This formalization should determine the current state of the system, and define a rule to the state evolution based on event occurrences.

Considering the set of events as the alphabet of the system, then a trace of events form a word and a set of traces forms a language. Thus, the set of all possible traces of a given system is the language generated by this system. The knowledge of the language of the system and its initial state is sufficient to model a DES, but this formalization might be too complex in a practical sense. To simplify, DES are commonly modeled as labeled graph structures. In this work, DESs will be modeled as automata.

### 2.1 Languages

A language is a set of traces of events of finite length generated by the system. Thus, a language is a math formalism that can be used to describe a DES [2].

#### 2.1.1 Notations and definitions

In this work, the notation  $\Sigma$  represents the set of events of a DES. The symbol  $\sigma$  will be used to represent a generic event, and  $\varepsilon$  represents the empty event trace. Let  $s$  be a trace, then its length is denoted by  $\|s\|$ . By convention, the length of the empty trace  $\varepsilon$  is zero. The formal definition of a language is given in the sequel.

**Definition 2.1** (*Language*) *A language defined over a set of events  $\Sigma$  is a set of traces of events of finite length formed using events in  $\Sigma$ .* ■

**Example 2.1** *The language  $L = \{\varepsilon, a, b, ac, acb\}$  is formed with events of  $\Sigma = \{a, b, c\}$  and has five traces, including the empty trace.*

There are some operations involving traces and events that can be used to generate new traces, and therefore, languages. The main operation is the concatenation of one or more traces, aiming to generate a single one. The trace  $acb$  is obtained by the concatenation of trace  $ac$  with event  $b$ . Moreover, trace  $ac$  is formed by the concatenation of events  $a$  and  $c$ . The empty trace  $\varepsilon$  is the neutral element of concatenation, then  $\varepsilon s = s\varepsilon = s$ , where  $s$  is a trace of events.

The set formed by all traces of finite length constructed using elements of  $\Sigma$ , including the empty trace  $\varepsilon$ , is called the Kleene Closure of  $\Sigma$ , denoted by  $\Sigma^*$ . Thus, any language generated by a DES, whose event set is  $\Sigma$ , is a subset of  $\Sigma^*$ . Notice that,  $\emptyset$ ,  $\Sigma$  and  $\Sigma^*$  are languages.

Let us consider a trace  $s = tuv$ , where  $t, u, v \in \Sigma^*$ . Thus,  $t$  is called the prefix of  $s$ ,  $u$  is a subtrace of  $s$ , and  $v$  is called the suffix of  $s$ . In addition, the notation  $s/t$  will be used to denote the suffix of  $s$  after  $t$ . If  $t$  is not a prefix of  $s$ , then  $s/t$  is not defined.

## 2.1.2 Language operations

The *Concatenation* and *Kleene Closure* operations are formally defined over languages in Definitions 2.2 and 2.3, respectively [2].

**Definition 2.2** (*Concatenation*) *Let  $L_a, L_b \subseteq \Sigma^*$ , then the concatenation  $L_a L_b$  is defined as:*

$$L_a L_b = \{s \in \Sigma^* : (s = s_a s_b) \wedge (s_a \in L_a) \wedge (s_b \in L_b)\}.$$

■

A trace  $s$  is in  $L_a L_b$  if, and only if,  $s$  can be formed by the concatenation of a trace in  $L_a$  with a trace in  $L_b$ .

**Definition 2.3** (*Kleene Closure*) *Let  $L \subseteq \Sigma^*$ , then the Kleene closure of  $L$ ,  $L^*$ , is defined as:*

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

■

The elements of  $L^*$  are generated by the concatenation of a finite amount of elements of  $L$ . Notice that the Kleene Closure operation is idempotent, that means  $(L^*)^* = L^*$ .

In the sequel, some other operations over languages are defined [2].

**Definition 2.4** (*Prefix Closure*) Let  $L \subseteq \Sigma^*$ , then

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

■

The prefix closure of a language  $L$ ,  $\bar{L}$ , is the set of all prefixes of all traces of  $L$ . Notice that  $L \subseteq \bar{L}$ , and  $L$  is said to be prefix-closed if  $L = \bar{L}$ .

**Definition 2.5** (*Post Language*) Let  $L \subseteq \Sigma^*$  and  $s \in L$ . The post language of  $L$  after  $s$ , denoted by  $L/s$ , is defined as:

$$L/s = \{t \in \Sigma^* : st \in L\}.$$

By definition,  $L/s = \emptyset$ , if  $s \notin \bar{L}$ .

■

The following example illustrates the concatenation and prefix closure operations applied to languages.

**Example 2.2** Let  $\Sigma = \{a, b, c\}$ ,  $L_1 = \{\varepsilon, a, ab, aab\}$ , and  $L_2 = \{c\}$ . Notice that, since  $aa \notin L_1$  and  $\varepsilon \notin L_2$ ,  $L_1$  and  $L_2$  are not prefix-closed, then,  $L_1L_2 = \{c, ac, abc, aabc\}$ ,  $\bar{L}_1 = \{\varepsilon, a, ab, aa, aab\}$ ,  $\bar{L}_2 = \{\varepsilon, c\}$ ,  $L_1^* = \{\varepsilon, a, ab, aab, aa, aaab, aba, aaba, \dots\}$  and  $L_2^* = \{\varepsilon, c, cc, ccc, \dots\}$ .

**Remark 2.1** If  $L = \emptyset$ , then  $\bar{L} = \emptyset$ , and if  $L \neq \emptyset$ , then  $\varepsilon \in \bar{L}$  necessarily. Moreover,  $\emptyset^* = \{\varepsilon\}$  and  $\{\varepsilon\}^* = \{\varepsilon\}$ , and the concatenation of a language and the empty set results in an empty set, i.e.,  $\emptyset L = L\emptyset = \emptyset$ .

■

**Definition 2.6** (*Projection*) The projection  $P_s : \Sigma_l^* \rightarrow \Sigma_s^*$ , considering  $\Sigma_s \subset \Sigma_l$ , is defined recursively as:

$$\begin{aligned} P_s(\varepsilon) &= \varepsilon, \\ P_s(\sigma) &= \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_s \\ \varepsilon, & \text{if } \sigma \in \Sigma_l \setminus \Sigma_s \end{cases}, \\ P_s(s\sigma) &= P(s)P(\sigma), \forall s \in \Sigma_l^*, \sigma \in \Sigma_l. \end{aligned}$$

with  $\setminus$  denoting set difference.

■

Notice that the projection operation erases all events  $\sigma \in \Sigma_l \setminus \Sigma_s$  of the traces  $s \in \Sigma_l^*$ .

**Example 2.3** Consider the sets  $\Sigma_l = \{a, b, c\}$  and  $\Sigma_s = \{b\}$  and the event traces  $s_1 = ac$  and  $s_2 = acb$ , with  $s_1, s_2 \in \Sigma_l^*$ . Then, the projection  $P_s : \Sigma_l^* \rightarrow \Sigma_s^*$  applied to  $s_1$  and  $s_2$  equals to  $P_s(s_1) = \varepsilon$  and  $P_s(s_2) = b$ , respectively.

The inverse projection operation is defined as follows.

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

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

■

For a given trace  $t$  formed with events of  $\Sigma_s$ , the inverse projection operation  $P_s^{-1}$  applied to  $t$  generates the set of all traces  $s \in \Sigma_l^*$  such that  $P_s(s) = t$ .

Both projection  $P_s$  and inverse projection  $P_s^{-1}$  operations can be defined over languages, applying the operation to all traces in the language. Formally, these operations applied to a language are defined as follows [2].

**Definition 2.8** (Projection Over Languages) Let  $L \subseteq \Sigma_l^*$ , then  $P_s(L)$  is defined as:

$$P_s(L) = \{t \in \Sigma_s^* : (\exists s \in L)[P_s(s) = t]\}$$

■

**Definition 2.9** (Inverse Projection Over Languages) Let  $L_s \subseteq \Sigma_s^*$ , then  $P_s^{-1}(L_s)$  is defined as:

$$P_s^{-1}(L_s) = \{s \in \Sigma_l^* : (\exists t \in L_s)[P_s(s) = t]\}$$

■

The projection operation can be used to represent the observed language of a system from an observer that only recognizes events from sensors and controller commands. Unobservable events are erased from all traces of the language generated by the system by applying the projection operation, obtaining the observable language of the system.

The following example illustrates the projection and inverse projection operations applied to languages.

**Example 2.4** Let  $\Sigma_l = \{a, b, c\}$ ,  $\Sigma_s = \{a, b\}$ , and the language  $L = \{a, b, c, ac, abc\} \subseteq \Sigma_l^*$ . Let  $P_s : \Sigma_l^* \rightarrow \Sigma_s^*$ . By applying projection  $P_s$  to language  $L$ , the events  $\sigma \in \Sigma_l \setminus \Sigma_s$  are erased from all traces  $s \in L$ . In this case,  $\Sigma_l \setminus \Sigma_s = \{c\}$ , which implies that  $P_s(L) = \{\varepsilon, a, b, ab\}$ , and  $P_s^{-1}(\{ab\}) = \{c\}^*a\{c\}^*b\{c\}^*$ .

## 2.2 Automata

An automaton is a device that is capable of representing a DES. The formal definition of a deterministic automaton is presented as follows [2].

**Definition 2.10** (*Deterministic Automaton*) A deterministic automaton, denoted by  $G$ , is a six-tuple:

$$G = (X, \Sigma, f, \Gamma, x_0, X_m),$$

where  $X$  is the set of states,  $\Sigma$  is the set of events,  $f : X \times \Sigma \rightarrow X$  is the transition function,  $\Gamma : X \rightarrow 2^\Sigma$  is the active event function,  $x_0$  is the initial state, and  $X_m \subseteq X$  is the set of marked states. ■

Notice that  $f$  can be extended from domain  $X \times \Sigma$  to domain  $X \times \Sigma^*$  in the following recursive manner:

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

An automaton can be graphically represented by a directed graph called the state transition diagram. The graph nodes, drawn as circles, are the states, and the labeled arcs represent the transitions between those states. The events of  $\Sigma$  label the transitions of the graph.

The initial state is indicated by an arrow and the marked states are represented by two concentric circles. The arcs graphically represent the transition function  $f : X \times \Sigma \rightarrow X$ . In Example 2.5, the graphical representation of an automaton is illustrated.

**Example 2.5** Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $X = \{0, 1, 2\}$ ,  $\Sigma = \{a, b\}$ , the transition function is defined as  $f(0, a) = 1; f(0, b) = 0; f(1, b) = 2; f(2, a) = 2; f(2, b) = 0$ , the active event function is defined as  $\Gamma(0) = \Gamma(2) = \{a, b\}; \Gamma(1) = \{b\}$ ,  $x_0 = 0$  and  $X_m = \{2\}$ .  $G$  can be graphically represented as the state transition diagram shown in Figure 2.1.

The generated language and the marked language of an automaton are defined as follows.

**Definition 2.11** (*Generated Language*) The generated language of a given automaton  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  is defined as:

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

■

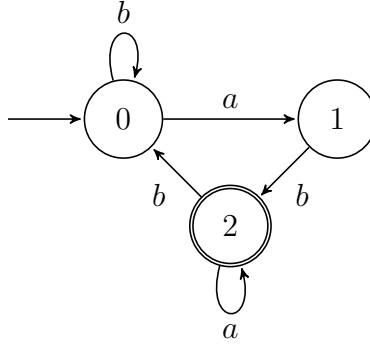


Figure 2.1: State transition diagram of automaton  $G$  of Example 2.5.

**Definition 2.12** (*Marked Language*) The marked language of automaton  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  is defined as:

$$\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\}.$$

■

The language generated by automaton  $G$ ,  $\mathcal{L}(G)$ , is composed of all feasible traces from the initial state. It is important to notice that  $\mathcal{L}(G)$  is prefix-closed by definition, since a path is feasible only if all its prefixes are also feasible. Moreover, some events in  $\Sigma$  might not be used in the state transition diagram of  $G$  and, henceforth, do not belong to  $\mathcal{L}(G)$ .

The marked language of automaton  $G$ ,  $\mathcal{L}_m(G)$ , is a subset of  $\mathcal{L}(G)$  that contains all traces  $s$  such that  $f(x_0, s) \in X_m$ , *i.e.*, all traces that start in the initial state and lead to a marked state on the state transition diagram of  $G$ . It is important to notice that, since the states of  $X$  do not need to be marked,  $\mathcal{L}_m(G)$  is not necessarily prefix-closed.

### 2.2.1 Operations on automata

In order to analyze a discrete event system modeled as a finite state automaton, it is necessary to define operations that are capable of modifying its state transition diagram. Moreover, it is necessary to define some operations that allow to combine two or more automata, in order to obtain more complex models from the models of the components of the system.

The transpose  $G^T$  of a DES  $G$  is an automaton that is obtained by inverting all transitions of  $G$ , and is defined as follows [48].

**Definition 2.13** (*Transpose*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ . The transpose automaton  $G^T$  is defined as:

$$G^T = (X, \Sigma, f_T, \Gamma_T, x_0, X_m),$$



where, for all  $x \in X$ , and for all  $\sigma \in \Sigma$ ,  $f_T$  is defined as:

$$f_T(f(x, \sigma), \sigma) = \begin{cases} x, & \text{if } f(x, \sigma) \text{ is defined,} \\ \text{undefined,} & \text{otherwise.} \end{cases},$$

and for all  $x \in X$ ,  $\Gamma_T$  is defined as:

$$\Gamma_T(x) = \{\sigma \in \Sigma : (\exists y \in X)[f(y, \sigma) = x]\}$$

■

The complement of  $G$  forms an automaton  $G^{comp}$  such that  $\mathcal{L}_m(G^{comp}) = \Sigma^* \setminus \mathcal{L}_m(G)$ . This operations is defined as follows [2].

**Definition 2.14** (*Complement*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  be an automaton such that  $\mathcal{L}_m(G) \subseteq \mathcal{L}(G) \subseteq \Sigma^*$ . Then, the complement automaton  $G^{comp}$ , such that  $\mathcal{L}_m(G^{comp}) = \Sigma^* \setminus \mathcal{L}_m(G)$  is formed by the following steps:

$$1: (\forall x \in X \cup \{x_d\})(\forall \sigma \in \Sigma), \text{ define } f_{tot}(x, \sigma) = \begin{cases} f(x, \sigma), & \text{if } \sigma \in \Gamma(x) \\ x_d, & \text{otherwise} \end{cases}.$$

$$2: \forall x \in X \cup \{x_d\}, \text{ define } \Gamma_{tot}(x) = \Sigma.$$

$$3: G^{comp} \leftarrow (X \cup x_d, \Sigma, f_{tot}, \Gamma_{tot}, x_0, (X \cup x_d) \setminus X_m).$$

■

Basically, in the complement operation  $\mathcal{L}(G)$  is completed to  $\Sigma^*$  by adding state  $x_d$  to  $G$ , and adding transitions from all states  $x \in X \cup \{x_d\}$  to  $x_d$  labeled by events in  $\Sigma \setminus \Gamma(x)$ . Then, all marked states became unmarked states and all unmarked states became marked states.

The accessible part of an automaton  $G$  is a unary operation that erases all states of  $G$  that are not reachable from the initial state  $x_0$  and their associated transitions. The formal definition of the accessible part of an automaton is presented in the sequel [2].

**Definition 2.15** (*Accessible Part*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ . The accessible part of  $G$ , denoted by  $Ac(G)$ , is defined as:

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

where  $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$ ,  $f_{ac} : X_{ac} \times \Sigma^* \rightarrow X_{ac}$ ,  $X_{ac,m} = X_m \cap X_{ac}$ , and  $\Gamma_{ac} : X_{ac} \rightarrow 2^\Sigma$ . ■

It is important to notice that, after applying the accessible part operation to an automaton, the transition function is restricted to the smaller domain,  $X_{ac} \times \Sigma$ . In addition, the accessible part operation does not change the languages  $\mathcal{L}(G)$  and  $\mathcal{L}_m(G)$ .

A state  $x \in X$  is said to be coaccessible if there is a subtrace from  $x$  to a marked state. The coaccessible part operation erases all states in  $G$ , that are not coaccessible, and their corresponding transitions. The formal definition of the coaccessible part of an automaton is presented in the sequel [2].

**Definition 2.16** (*Coaccessible Part*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ . The coaccessible part of  $G$ , denoted by  $CoAc(G)$ , is defined as:

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

where  $X_{coac} = \{x \in X : (\exists s \in \Sigma^*)[f(x, s) \in X_m]\}$ ;  $x_{0,coac} = x_0$ , if  $x_0 \in X_{coac}$ , and  $x_{0,coac}$  is undefined, if  $x_0 \notin X_{coac}$ ;  $f_{coac} : X_{coac} \times \Sigma \rightarrow X_{coac}$ ; and  $\Gamma_{coac} : X_{coac} \rightarrow 2^\Sigma$ . ■

It is important to remark that the coaccessible part operation restricts the transition function to the smaller domain,  $X_{coac} \times \Sigma$ . Notice that  $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$ , and  $\mathcal{L}_m(CoAc(G)) = \mathcal{L}_m(G)$ .

An automaton that is both accessible and coaccessible is called Trim. The formal definition of the trim operation is presented as follows [2].

**Definition 2.17** (*Trim Operation*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ . The Trim operation is defined as:

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

■

The following example shows the results of the accessible part, coaccessible part and trim operations applied to an automaton  $G$ .

**Example 2.6** Consider automaton  $G$  depicted in Figure 2.2. Figures 2.3a, 2.3b and 2.3c show the resulting automata after the accessible part, coaccessible part and trim operations, respectively.

It is possible to combine two or more automata in order to obtain a new automaton model. In this work, two composition operations are presented: the product composition and the parallel composition.

The product composition is also called completely synchronous composition, and is denoted by  $\times$ . The formal definition of the product composition is presented in the sequel [2].

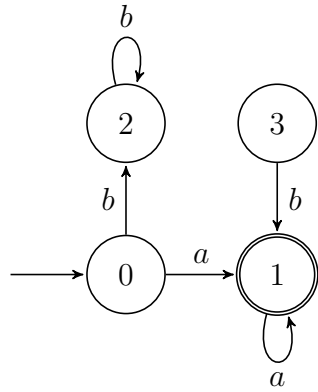


Figure 2.2: Automaton  $G$ , Example 2.6.

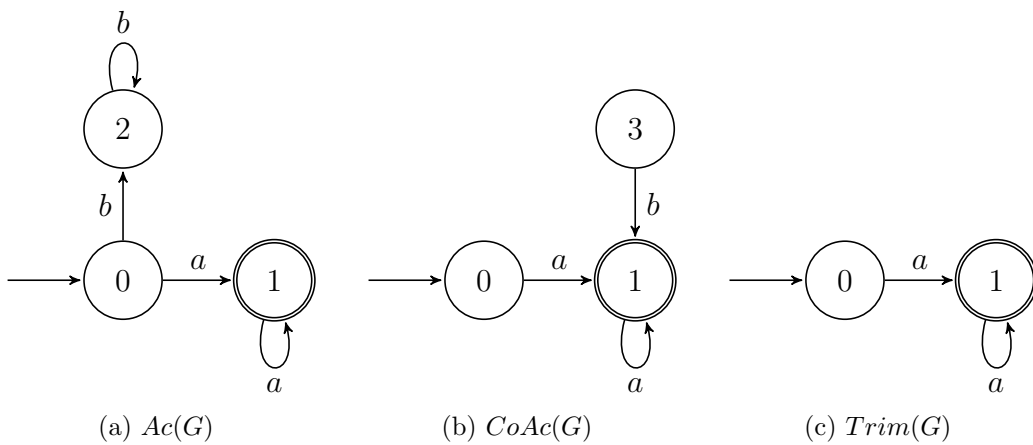


Figure 2.3:  $Ac(G)$ ,  $CoAc(G)$  and  $Trim(G)$ , respectively, of automaton  $G$  depicted in Figure 2.2.

**Definition 2.18** (*Product Composition*) Let  $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{01}, X_{m1})$  and  $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{02}, X_{m2})$ . Then, the product composition between  $G_1$  and  $G_2$ ,  $G_1 \times G_2$ , is given by:

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1 \times 2}, \Gamma_{1 \times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}),$$

where:

$$f_{1 \times 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

$$\Gamma_{1 \times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2).$$

■

According to Definition 2.18, the transitions of both automata need to be synchronized with a common event, *i.e.*, an event in  $\Sigma_1 \cap \Sigma_2$ . Thus, an event occurs in  $G_1 \times G_2$  if, and only if, the event occurs in  $G_1$  and  $G_2$ , simultaneously.

The states of  $G_1 \times G_2$  are pairs of the form  $(x_1, x_2)$ , where the first component is a state of  $G_1$  and the second one is a state of  $G_2$ . Moreover, considering that the product composition synchronizes the automata evolution, the generated language and the marked language of  $G_1 \times G_2$  are  $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$  and  $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$ , respectively.

The parallel composition, also called synchronous composition, is denoted by  $\parallel$ . Each automaton in the parallel composition can evolve with transitions that are labeled by private events, instead of only evolving if the event is feasible in both automata, as in the product composition. The formal definition of parallel composition is presented as follows [2].

**Definition 2.19** (*Parallel Composition*) Let  $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{01}, X_{m1})$  and  $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{02}, X_{m2})$ . The parallel composition between  $G_1$  and  $G_2$ , denoted by  $G_1 \parallel G_2$ , is given by:

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{\parallel 2}, \Gamma_{\parallel 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}),$$

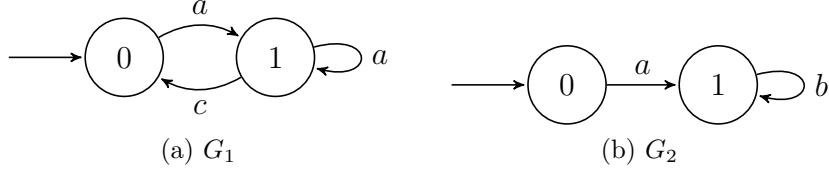


Figure 2.4: Automata  $G_1$  and  $G_2$ , Example 2.7.

where:

$$f_{1\parallel 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1, \sigma), x_2), & \text{if } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2 \\ (x_1, f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_2(x_2) \setminus \Sigma_1 \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

$$\Gamma_{1\parallel 2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus \Sigma_2] \cup [\Gamma_2(x_2) \setminus \Sigma_1].$$

■

According to Definition 2.19, the transitions of both automata need to be synchronized if the event is in  $\Sigma_1 \cap \Sigma_2$ , as in the product composition. However, if the event is in  $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$ , it can occur independently of the synchronization of the automata. Thus, an event  $\sigma \in \Sigma_1 \cap \Sigma_2$  can occur in  $G_1 \parallel G_2$  if  $\sigma$  occurs in  $G_1$  and  $G_2$  at the same time, and an event  $\sigma \in (\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$  can occur as long as it is feasible in  $G_1$  or  $G_2$ .

In order to obtain the generated and marked languages of  $G_1 \parallel G_2$ , it is needed to define the following projections:

$$P_i : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^* \text{ para } i = 1, 2.$$

The generated and marked languages of the parallel composition are given by  $\mathcal{L}(G_1 \parallel G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$  and  $\mathcal{L}_m(G_1 \parallel G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]$ , respectively. The product and parallel compositions are illustrated in the following example.

**Example 2.7** *Let  $G_1$  and  $G_2$  be shown in Figures 2.4a and 2.4b, respectively. Figures 2.5a and 2.5b show the automata computed from the product and parallel compositions of  $G_1$  and  $G_2$ , respectively.*

## 2.2.2 Automata under partial observation of events

The set of event in  $G$  can be partitioned as  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ , where  $\Sigma_o$  denotes the set of observable events and  $\Sigma_{uo}$  denotes the set of unobservable events. Observable events

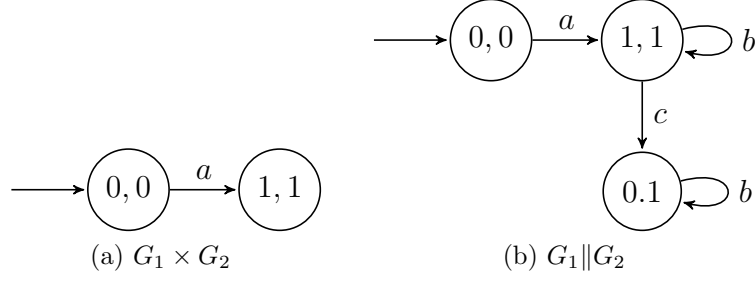


Figure 2.5: Resulting automata after product and parallel composition of  $G_1$  and  $G_2$ , Example 2.7.

are events that can be recognized by a sensor. On the other hand, unobservable events have no signal referring to it.

For a system  $G$  with unobservable events, the observable language generated by  $G$  is obtained by applying the projection  $P_o : \Sigma^* \rightarrow \Sigma_o^*$ , resulting in language  $P_o[\mathcal{L}(G)]$ . In order to obtain the possible states of the system after the observation of an event trace. It is necessary to construct an automaton called the observer of  $G$ , denoted by  $Obs(G)$ . Before presenting the algorithm for the construction of  $Obs(G)$ , it is necessary to define the unobservable reach function.

**Definition 2.20** (*Unobservable Reach*) Let the  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ . The unobservable reach of an state  $x \in X$ , denoted by  $UR(x)$ , is defined as:

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

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

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

■

The unobservable reach of a state  $x$  provides the set of states that are reached after the occurrence of a trace  $s \in \Sigma_{uo}^*$  from  $x$ .

**Definition 2.21** (*Observer*) The observer of an automaton  $G$  whose observable events set is  $\Sigma_o$ , denoted by  $Obs(G, \Sigma_o)$ , is given by:

$$Obs(G, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs}),$$

where  $X_{obs} \subseteq 2^X$  and  $X_{m,obs} = \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$ .  $f_{obs}$ ,  $\Gamma_{obs}$  and  $x_{0,obs}$  are obtained, by following the steps of Algorithm 2.1 [49]. ■

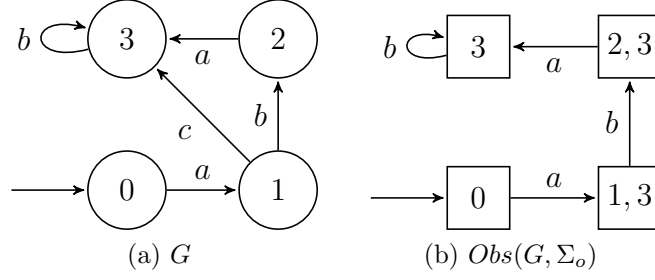


Figure 2.6: Automata  $G$  with unobservable events and  $Obs(G, \Sigma_o)$  respectively, Example 2.8.

---

**Algorithm 2.1** *Computation of the observer  $Obs(G, \Sigma_o)$*

---

**Input:**  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  and  $\Sigma_o$ , where  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ .

**Output:**  $Obs(G) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs})$ .

- 1: Define  $x_{0,obs} = UR(x_0)$ . Do  $X_{obs} = \{x_{0,obs}\}$  and  $\tilde{X}_{obs} = X_{obs}$ .
  - 2:  $\bar{X}_{obs} = \tilde{X}_{obs}$  and  $\tilde{X}_{obs} = \emptyset$ .
  - 3: For each  $B \in \bar{X}_{obs}$ ,
    - 3.1:  $\Gamma_{obs}(B) = (\bigcup_{x \in B} \Gamma(x)) \cap \Sigma_o$ .
    - 3.2: For each  $\sigma \in \Gamma_{obs}(B)$ ,
      - 3.2.1:  $f_{obs}(B, \sigma) = UR(x \in X : (\exists y \in B)[x = f(y, \sigma)])$ .
      - 3.2.2:  $\tilde{X}_{obs} \leftarrow \tilde{X}_{obs} \cup f_{obs}(B, \sigma)$ .
  - 4:  $X_{obs} \leftarrow X_{obs} \cup \tilde{X}_{obs}$ .
  - 5: Repeat steps 2 to 4 until the conclusion of the accessible part of  $Obs(G)$ .
  - 6:  $X_{m,obs} = \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$ .
- 

Notice that the generated and marked language of automaton  $Obs(G, \Sigma_o)$  are  $\mathcal{L}(Obs(G, \Sigma_o)) = P_o[\mathcal{L}(G)]$  and  $\mathcal{L}_m(Obs(G, \Sigma_o)) = P_o[\mathcal{L}_m(G)]$ , respectively.

**Example 2.8** Let  $G$  be the automaton shown in Figure 2.6a.  $X = \{0, 1, 2, 3\}$  is the set of states, and  $\Sigma = \{a, b, c\}$  is the set of events of  $G$ , where  $\Sigma_o = \{a, b\}$  and  $\Sigma_{uo} = \{c\}$ . Then, according to Algorithm 2.1,  $Obs(G, \Sigma_o)$ , presented in Figure 2.6b, is computed. Notice that, if the system executes the trace  $t = acb$ , the observed trace is  $P_o(t) = ab$ , and the state estimate is  $\{2, 3\}$ .

Example 2.8 shows the observer  $Obs(G, \Sigma_o)$  of an automaton  $G$  with unobservable events. Notice that each state in  $Obs(G, \Sigma_o)$  is a set of estimated states in  $G$  after the observation of a trace of events.

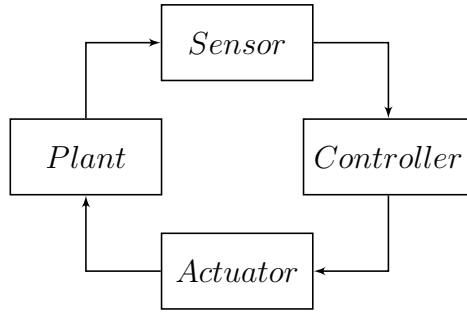


Figure 2.7: Interaction between all four elements.

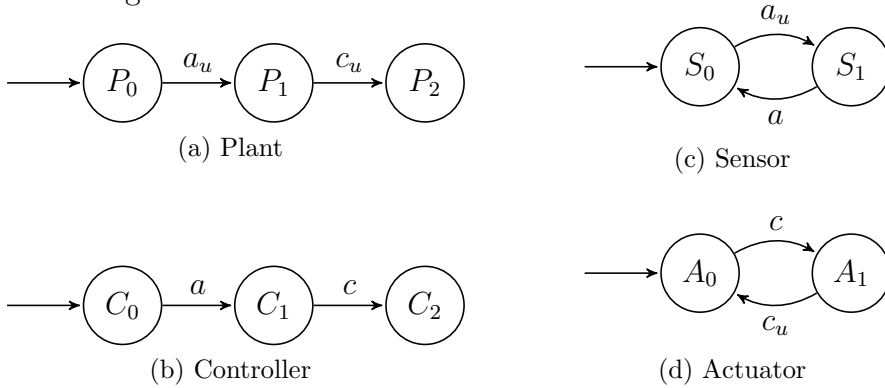


Figure 2.8: Four basic elements of a feedback discrete event system.

## 2.3 Controlled discrete event system

A controlled discrete event system is usually composed of four elements: plant, controller, sensors, and actuators. The plant and the controller interact as follows: the plant sends signals to the controller through sensors, and the controller sends command signals to the plant through actuators [12]. This relationship is represented in Figure 2.7.

Each of the four elements can be represented as an automaton, as illustrated in the example of Figure 2.8. In Figure 2.8a, the automaton of the plant has only unobservable events, referring to the lack of logical signals as an intrinsic event. On the other hand, the automaton of the controller (Figure 2.8b) has only observable events, referring to the communication with the controller being composed of logical signals. To allow the interaction, *i.e.*, synchronization between those two automata, the automata of the sensor and actuator, presented in Figures 2.8c and 2.8d, respectively, have events of both plant and controller automata.

The behavior of the controlled system can be generated by the parallel composition of all four automata, as shown in Figure 2.9. Such system first executes a physical action on the plant, then the sensor recognizes this event and communicates it to the controller. The controller, after considering the occurrence of the event, sends a logical signal to the actuator. This signal is, then, translated into a physical action to be applied in the plant.



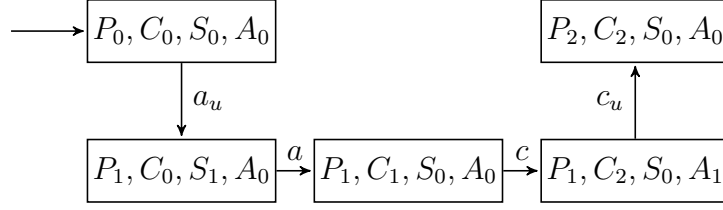


Figure 2.9: Behavior of a discrete event system.

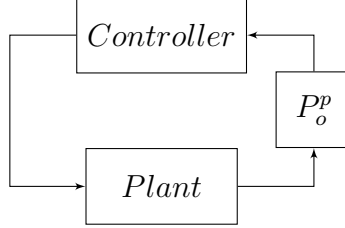


Figure 2.10: Block diagram of a plant being controlled.

The block diagram of a controlled DES with partial observation is represented in Figure 2.10, where  $P_o^p : \Sigma^* \rightarrow \Sigma_o^{p*}$  denotes a projection operation, where  $\Sigma_o^p$  is the set of observable events of the plant. The lack of sensors is the physical interpretation of unobservable events, as presented in Section 2.2.2.

The controller is a reactive element that receives signals from sensors and sends signals to actuators, and there exist several methods proposed in the literature for the design of discrete event controllers [3–5, 11, 50]. The controller is usually implemented on a Programmable Logic Controller (PLC) or the control action can be executed by a human operator that checks panels and press buttons. Depending on the PLC programming methodology, or the training of human operators, the plant might have undesired behaviors, specially if DES theory is not applied at the designing stage of the DEC.

## 2.4 Supervisory control theory

Differently from the controller, the supervisor is used to restrict behaviors on the system, in a higher level. This is done by disabling events in the system states. The interaction between plant and controller remains the same, and the supervisor observes events of the plant and controller. This interaction is represented in the block diagram of Figure 2.11. Notice that, since only part of the command events may be observable, then it is necessary to introduce the projection  $P_o^c : \Sigma^* \rightarrow \Sigma_o^{c*}$ , where  $\Sigma_o^c$  denotes the set of events of the controller that can be observed by the supervisor. Considering the abstraction of the controlled plant behavior, the block diagram of Figure 2.11 can be simplified leading to the block diagram of Figure 2.12, where  $P_o : \Sigma^* \rightarrow \Sigma_o^*$ , and  $\Sigma_o = \Sigma_o^p \cup \Sigma_o^c$ .

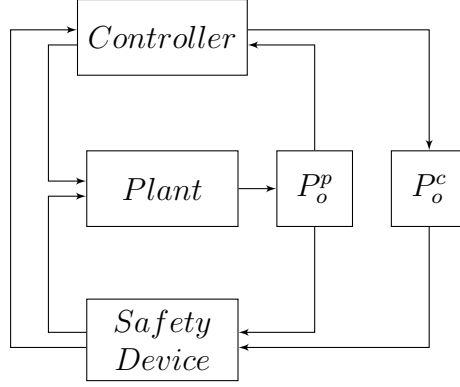


Figure 2.11: Block diagram of the interaction of plant, controller and supervisor.

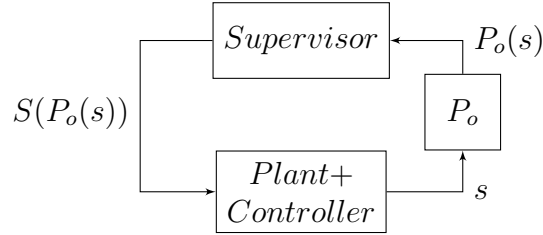


Figure 2.12: Block diagram of the interaction supervisor and a controlled plant.

Let us consider that automaton  $G$  models the controlled behavior of the system, *i.e.*,  $G$  is obtained by the parallel composition of the plant, controller, sensors and actuators automata. Then, language  $\mathcal{L}(G)$  may contain traces that lead the system to undesirable states, *i.e.*, states that might be unsafe or block the system [2]. The sublanguage of  $\mathcal{L}(G)$  that contains all admissible traces is called admissible language, and is denoted by  $L_a$ . The behavior described by  $L_a$  is the maximal admissible behavior of the system. Usually the supervisor cannot force the controlled plant to execute only traces that are in the admissible behavior, but can guarantee that only traces in a subset of  $L_a$  be executed. In addition, the controlled plant to be supervised have some behavior requirements to follow, and thus, it is, in general, defined a sublanguage of  $L_a$ , denoted by  $L_r$ , that contains all traces that satisfy this minimal required behavior. Summarizing, languages  $L_a$ ,  $L_r$ , and  $\mathcal{L}(G)$ , relate to each other as  $L_r \subseteq L_a \subseteq \mathcal{L}(G)$ .

In order to design a supervisor, the desired behavior, described by specification language  $K$ , need to contain  $L_r$  and also be a subset of  $L_a$ , *i.e.*,  $L_r \subseteq K \subseteq L_a \subseteq \mathcal{L}(G)$ . Moreover, different bounds for the specification language  $K$  can be defined as desired language  $L_{des}$  and tolerated language  $L_{tol}$ . Then, specification language  $K$  must achieve as much as possible of  $L_{des}$  without ever exceeding  $L_{tol}$ . Thus,  $L_{tol}$  is essentially equal to  $L_a$ , but  $L_{des}$  is different from  $L_r$  in such a way that  $K \cap L_{des}$  is not necessarily equal to  $L_{des}$ , and there are no  $K' \subseteq L_{tol}$  such that  $K'$  can be realized as a supervisor and  $(K \cap L_{des}) \subset (K' \cap L_{des})$ .

After defining specification language  $K$ , the next step is to synthesize a supervisor

$S$ . However, some events cannot be disabled. Thus,  $\Sigma$  needs to be partitioned in a set of controllable events  $\Sigma_c$ , and a set of uncontrollable events  $\Sigma_{uc}$ . Hence, the supervisor can be represented as a function defined as  $S : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$ , as shown in Figure 2.12, where  $G$  stands for the automaton of the *Plant+Controller*. Thus, for each  $s \in \mathcal{L}(G)$ , the set of enabled events in  $S$  controlling  $G$ , denoted by  $S/G$ , is  $S(P_o(s)) \cap \Gamma(f(x_0, s))$ . Also,  $S$  cannot disable uncontrollable events, which means that  $\Gamma(f(x_0, s)) \cap \Sigma_{uc} \subseteq S(P_o(s))$  for every  $s \in \mathcal{L}(G)$ .

Considering function  $S$ , the generated and marked languages of  $S/G$  are defined as follows:

**Definition 2.22** (*Languages generated and marked by  $S/G$* ) *The language generated by  $S/G$  is defined recursively as:*

1.  $\varepsilon \in \mathcal{L}(S/G)$ ;
2.  $[(s \in \mathcal{L}(S/G)) \wedge (s\sigma \in \mathcal{L}(G)) \wedge (\sigma \in S(s))] \Leftrightarrow [s\sigma \in \mathcal{L}(S/G)]$ .

*The language marked by  $S/G$  is defined as:*

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

■

A supervised system is said to be blocking if there is a state that cannot reach a marked state due to a disabled event or a characteristic of the system. A blocking supervised system is defined in the sequel.

**Definition 2.23** (*Blocking supervised system*) *The DES  $S/G$  is blocking if*

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

*and nonblocking when*

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

■

**Example 2.9** *Let  $G$  be the automaton depicted in Figure 2.13a, and consider that  $L_a = \mathcal{L}(G)$ . Let the required language be defined as  $L_r = \{\varepsilon, a, aa, aaa, \dots\}$ , and  $K = \mathcal{L}_m(G)$ . Thus, the supervisor  $S$ , such that  $S/G = K$ , can be realized as automaton  $H$ , shown in Figure 2.13b, and the state transition diagram of  $G||H$  is depicted in Figure 2.13c.*

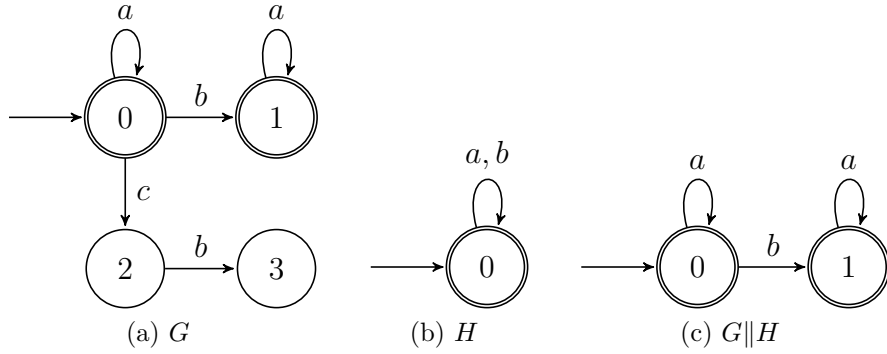


Figure 2.13: DES  $G$ , Example 2.9.

### 2.4.1 Supervisor with partial controllability

In this section, the controllability problem [2] is addressed. The *observability problem* will be addressed in another section, and thus, in this section, all events are considered observable.

**Definition 2.24** (*Controllability*) Consider  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  and let  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ , where  $\Sigma_{uc}$  is the set of uncontrollable events. Let  $K \subseteq \mathcal{L}(G)$ , where  $K \neq \emptyset$ . Then  $K$  is said to be controllable with respect to  $\mathcal{L}(G)$ , and  $\Sigma_{uc}$  if

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

■

According to Definition 2.24 [2], if there exists an event  $\sigma_{uc} \in \Sigma_{uc}$  such that  $s \in \overline{K}$ ,  $s\sigma_{uc} \in \mathcal{L}(G)$ , and  $s\sigma_{uc} \notin \overline{K}$ , then there does not exist a supervisor  $S$  such that  $S/G = K$ . Notice that, by definition, if  $K$  is controllable, then  $\overline{K}$  is controllable.

**Example 2.10** Let  $G$  be the automaton depicted in Figure 2.13a, where  $\Sigma_{uc} = \{b\}$ . If  $K = \mathcal{L}_m(G)$ , then the supervisor of Example 2.9, depicted in Figure 2.13b, is capable of keeping the system inside the specification language. However, if the specification language is set to also reach state 2, then  $K$  is uncontrollable with respect to  $\mathcal{L}(G)$  and  $\Sigma_{uc}$ , since the supervisor enables event  $c$  in state 0 but cannot disable event  $b$  after the occurrence of  $c$ , which leads the system out of the specification language.

Controllable languages in  $\mathcal{L}(G)$  have the following properties [2].

1. If  $K_1$  and  $K_2$  are controllable, then  $K_1 \cup K_2$  is controllable.
2. If  $K_1$  and  $K_2$  are controllable, then  $K_1 \cap K_2$  is not necessarily controllable.

3. If  $\overline{K_1} \cap \overline{K_2} = \overline{(K_1 \cap K_2)}$  and  $K_1$  and  $K_2$  are controllable, then  $K_1 \cap K_2$  is controllable.
4. If  $K_1$  and  $K_2$  are prefix-closed and controllable, then  $K_1 \cap K_2$  is prefix-closed and controllable.

Let us also define the class of controllable sublanguages of  $K$ ,  $\mathcal{C}_{in}(K)$ , and the class of prefix-closed and controllable superlanguages of  $K$ ,  $\mathcal{CC}_{out}(K)$ , as follows:

$$\begin{aligned}\mathcal{C}_{in}(K) &:= \{L \subseteq K : \overline{L}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{L}\} \\ \mathcal{CC}_{out}(K) &:= \{L \subseteq \Sigma^* : (K \subseteq L \subseteq \mathcal{L}(G)) \wedge (\overline{L} = L) \wedge (\overline{L}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{L})\}\end{aligned}$$

From the controllability definition and its properties, it is possible to define two other languages derived from a specification  $K \subseteq \mathcal{L}(G)$ :

- $K^{\uparrow C}$ , the *supremal controllable sublanguage* of  $K$ ;
- $K^{\downarrow C}$ , the *infimal prefix-closed and controllable superlanguage* of  $K$ .

The supremal controllable sublanguage of  $K$ ,  $K^{\uparrow C}$ , refers to a subset of  $K$  that is controllable and contains all controllable sublanguages of  $K$ . Thus,  $K^{\uparrow C}$  can be defined as follows:

$$K^{\uparrow C} := \bigcup_{L \in \mathcal{C}_{in}(K)} L.$$

Notice that, since, according to property 1, the union of controllable languages is a controllable language, then  $K^{\uparrow C} \in \mathcal{C}_{in}(K)$ . In the worst case,  $K^{\uparrow C} = \emptyset$ , since  $\emptyset \in \mathcal{C}_{in}(K)$ , and, if  $K$  is controllable, then  $K^{\uparrow C} = K$ . The standard algorithm to obtain  $K^{\uparrow C}$  of a language  $K$  is presented in the sequel [2].

**Algorithm 2.2** *Algorithm for the computation of  $K^{\uparrow C}$*

**Inputs:**  $G = (X, \Sigma, f, \Gamma, x_0)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ , the specification language  $K \subseteq \mathcal{L}(G)$ , and  $H = (Y, \Sigma, g, \Gamma_H, y_0, Y_m)$ , where  $\mathcal{L}_m(H) = K$  and  $\mathcal{L}(H) = \overline{K}$ .

**Outputs:**  $K^{\uparrow C}$  and  $H_K$ , where  $\mathcal{L}(H_K) = \overline{K^{\uparrow C}}$ .

- 1: Mark all states of  $G$ .
- 2: Define  $H_0 = (Y_0, \Sigma, g_0, \Gamma_{H_0}, (y_0, x_0), Y_{0,m}) = H \times G$ , where  $Y_0 \subseteq Y \times X$ . Define  $i = 0$ .
- 3: Calculate (The notation “ $\upharpoonright$ ” stands for “restricted to”)

3.1:

$$\begin{aligned}
Y'_i &= \{(y, x) \in Y_i : \Gamma(x) \cap \Sigma_{uc} \subseteq \Gamma_{H_i}(y, x)\} \\
g'_i &= g_i|_{Y'_i} \\
\Gamma'_{H_i} &= \Gamma_{H_i}|_{g'_i} \\
Y'_{i,m} &= Y_{i,m} \cap Y'_i
\end{aligned}$$

3.2: Define  $H_{i+1} = \text{Trim}(Y'_i, \Sigma, g'_i, (y_0, x_0), Y'_{i,m})$ .

If  $H_{i+1}$  is the empty automaton, i.e.,  $(y_0, x_0)$  is deleted in the above calculation, then  $K^{\uparrow C} = \emptyset$ ,  $H_K \leftarrow H_{i+1}$  and STOP.

Otherwise, define  $(Y_{i+1}, \Sigma, g_{i+1}, \Gamma_{H_{i+1}}, (y_0, x_0), Y_{i+1,m})$  as  $H_{i+1}$ .

4: If  $H_{i+1} = H_i$ , then  $H_K \leftarrow (Y_{i+1}, \Sigma, g_{i+1}, \Gamma_{H_{i+1}}, (y_0, x_0), Y_{i+1})$  and  $K^{\uparrow C} \leftarrow \mathcal{L}_m(H_{i+1})$  and STOP.

Otherwise,  $i \leftarrow i + 1$  and go to Step 3.

**Example 2.11** Let  $G$  be the automaton depicted in Figure 2.14a. Let the specification  $K = \{a\}^*\{b\}\{a\}^* \cup \{a\}^*\{c\}$  and the set of uncontrollable events  $\Sigma_{uc} = \{d\}$ . Thus, an automaton  $H$ , such that  $\mathcal{L}_m(H) = K$  and  $\mathcal{L}(H) = \overline{K}$ , is shown in Figure 2.14b. According to Algorithm 2.2, in the first step all states of  $G$  are marked, as shown in Figure 2.15a. Then,  $H_0$  is defined as the product composition  $H \times G$ , which, in this case, has the same marked and generated language as  $H$ , as depicted in Figure 2.15b. Thus,  $Y_0 = \{(0, 0), (1, 1), (2, 2)\}$  and  $\Gamma(0) \cap \Sigma_{uc} = \emptyset \subseteq \Gamma_{H_0}((0, 0))$ ,  $\Gamma(1) \cap \Sigma_{uc} = \emptyset \subseteq \Gamma_{H_0}((1, 1))$ , and  $\Gamma(2) \cap \Sigma_{uc} = \{d\} \not\subseteq \Gamma_{H_0}((2, 2)) = \emptyset$ . This implies that state  $(2, 2)$  is not in  $Y_1$ . The Trim operation leads to the automaton depicted in Figure 2.15c. Since  $H_1 \neq H_0$ , then step 3 is repeated, which leads to  $H_2 = H_1$ , and, therefore, the output of algorithm is  $K^{\uparrow C} = \mathcal{L}_m(H_1)$ , where  $H_K$  is shown in Figure 2.15d.

The infimal prefix-closed and controllable superlanguage of  $K$ ,  $K^{\downarrow C}$ , refer to a superset of  $K$  that is either prefix-closed, controllable, and is a subset of all prefix-closed and controllable superlanguages of  $K$ . Thus,  $K^{\downarrow C}$  can be defined as follows:

$$K^{\downarrow C} := \bigcap_{L \in \mathcal{CC}_{out}(K)} L.$$

According to property 4, all languages in  $\mathcal{CC}_{out}(K)$  are prefix-closed and controllable, then,  $K^{\downarrow C}$  is prefix-closed and controllable. Notice that, in the worst case,  $K^{\downarrow C} = \mathcal{L}(G)$ , since  $\mathcal{L}(G) \in \mathcal{CC}_{out}(K)$  by definition, and, if  $K$  is controllable, then

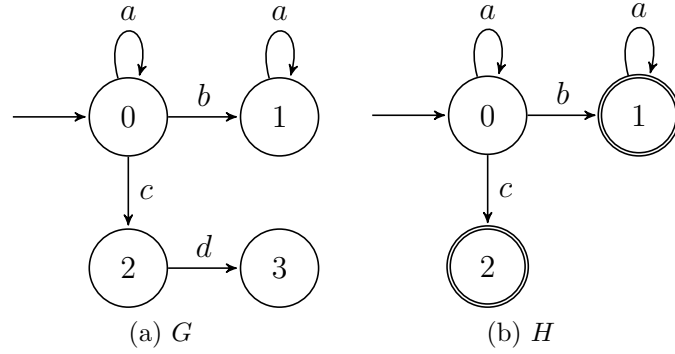


Figure 2.14: Input automata for Algorithm 2.2 in Example 2.11.

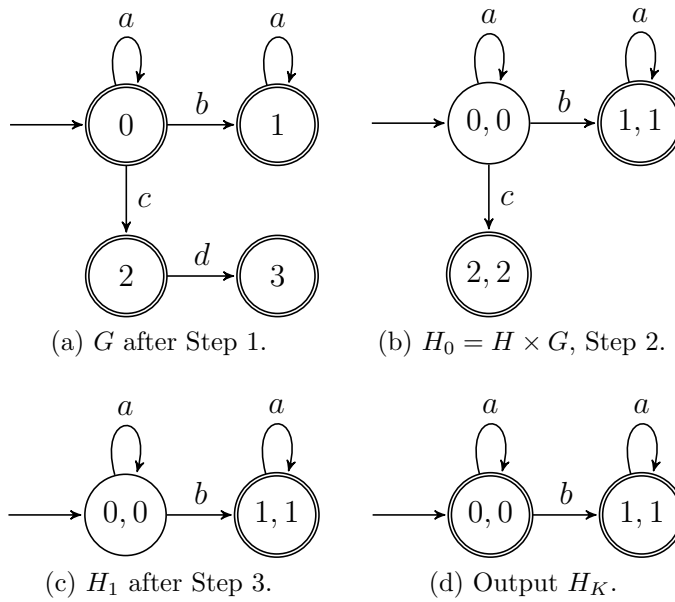


Figure 2.15: Steps of Algorithm 2.2 in Example 2.11.

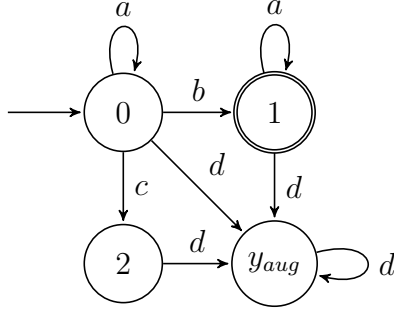


Figure 2.16: Automaton  $H_{aug}$  from Example 2.12.

$K^{\downarrow C} = \overline{K}$ . The standard algorithm to obtain  $K^{\downarrow C}$  of a language  $K$  is shown in the sequel [2].

---

**Algorithm 2.3** *Algorithm for Computation of  $K^{\downarrow C}$*

---

**Input:**  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ , the specification language  $K \in \mathcal{L}(G)$  and  $H = (Y, \Sigma, g, \Gamma_H, y_0, Y_m)$ , where  $\mathcal{L}(H) = \overline{K}$ .

**Output:**  $K^{\downarrow C}$  and  $H_K$ , where  $\mathcal{L}(H_K) = K^{\downarrow C}$ .

- 1: Define  $Y_{aug} = Y \cup \{y_{aug}\}$ .
- 2: For all  $y \in Y_{aug}$ , define  $\Gamma_{aug}(y) = \Gamma_H(y) \cup \Sigma_{uc}$ .
- 3:  $(\forall y \in Y_{aug})(\forall \sigma \in \Sigma)$  Define:

$$g_{aug}(y, \sigma) = \begin{cases} g(y, \sigma), & \text{if } g(y, \sigma) \text{ is defined} \\ y_{aug}, & \text{if } g(y, \sigma) \text{ is undefined and } \sigma \in \Sigma_{uc} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

- 4: Define  $H_{aug} = (Y_{aug}, \Sigma, g_{aug}, \Gamma_{aug}, y_0, Y_m)$ .
  - 5: Set  $H_K = H_{aug} \times G$ , and  $K^{\downarrow C} = \mathcal{L}(H_K)$ .
- 

**Example 2.12** Let  $G$ ,  $K$  and  $\Sigma_{uc}$  be the same presented in Example 2.11. Thus,  $H$  is shown in Figure 2.14b. According to Algorithm 2.3, add state  $y_{aug}$  to the set of states in  $H$  and create transitions from all states to  $y_{aug}$  labeled by  $d$ , as depicted in Figure 2.16 and call this automaton  $H_{aug}$ . The product composition  $H_{aug} \times G$ , in this example, have the same generated language as  $G$ , i.e.,  $\mathcal{L}(H_{aug} \times G) = \mathcal{L}(G)$ . Then,  $H_K = G$  and  $K^{\downarrow C} = \mathcal{L}(G)$ .



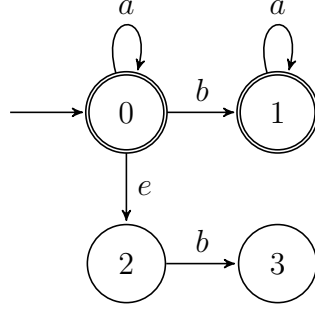


Figure 2.17: Automaton  $G$  from Example 2.13.

## 2.4.2 Supervisor under partial observation

As mentioned in Section 2.2.2, an event can be unobservable. In supervisory control, an unobservable event can lead the supervisor to disable events that should not be disabled, according to the specification. In order to prevent the supervisor from disabling an specific behavior, the specification language  $K$  have to satisfy the observability condition. The observability condition can be defined as follows [2]:

**Definition 2.25** (*Observability*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  and  $K \subseteq \Sigma^*$ . Let  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and let  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  be a projection function. Then,  $K$  is said to be observable with respect to  $\mathcal{L}(G)$ ,  $\Sigma_o$ , and  $\Sigma_c$  if,  $\forall s \in \overline{K}$  and  $\forall \sigma \in \Sigma_c$ ,

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

■

According to Definition 2.25, if the supervisor cannot distinguish between two traces, then these traces should require the same control action, thus, if an event must be disabled after the observation of a trace, then this event should not be part of specification language. If the specified language does not satisfy the observability condition, then there are traces  $s, t \in K$  such that  $P(s) = P(t)$  and  $s\sigma \in K$ , but  $t\sigma \notin K$ . Thus, a supervisor cannot decide to disable or not event  $\sigma$ .

**Example 2.13** Let  $G$  be an automaton as depicted in Figure 2.17. Let  $K = \mathcal{L}_m(G) \cup \{a\}^*\{e\}$ , and  $\Sigma_{uo} = \{e\}$  be the set of unobservable events. Thus, since the supervisor does not distinguish between states 0 and 2, then the supervisor cannot decide to disable event  $b$ .

In order to guarantee the existence of a supervisor for unobservable languages  $K$ , it is possible to define control policies to determine which events should be disabled by the supervisor. There are two possible control polices that can be used to classify a supervisor as permissive or anti-permissive. These control policies are defined as follows [46, 47].

**Definition 2.26** (*Control Policy*) Let  $G$  be an automaton, where  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ . Let  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  be the projection function,  $K \subseteq \mathcal{L}(G)$  be the specification language, and  $S : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$  be a supervisor for automaton  $G$ . Then, a supervisor is said to be permissive if, for each  $s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$ ,

$$P_o^{-1}(s)\{\sigma\} \cap K \neq \emptyset \Leftrightarrow \sigma \in S(s),$$

and a supervisor is said to be anti-permissive if, for each  $s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$ ,

$$(P_o^{-1}(s) \cap K)\{\sigma\} \cap (\mathcal{L}(G) \setminus K) \neq \emptyset \Leftrightarrow \sigma \notin S(s).$$

■

If the supervisor cannot distinguish between two traces, then these traces should require the same control action. In addition, if this control action may exceed the specification language, then the control policy determine if the supervisor should or should not disable this event. A permissive supervisor does not disable the event, and an anti-permissive supervisor disables the event. Thus, it is important to remark that, specification language  $K$  has a set inclusion relation with  $\mathcal{L}(S/G)$  depending on the control policy [46, 47].

**Proposition 2.1** Let  $G$  be an automaton, where  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ . Let  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  be the projection function, and  $K \subseteq \mathcal{L}(G)$  be the specification language. Let the supervisors  $S_P : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$  and  $S_A : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$  be permissive and anti-permissive, respectively. Then,

$$\mathcal{L}(S_A/G) \subseteq K \subseteq \mathcal{L}(S_P/G).$$

■

It is important to remark that the control of the discrete event system is also possible with two or more supervisors. Thus, it is necessary to define the architecture of the interaction between supervisors and plant. There are plenty of presented architectures that are possible to implement [2], and, in this work, we use the modular supervisory control architecture, as depicted in Figure 2.18 [2]. In the modular supervisory control, an event in the plant is feasible only if both supervisors allow its triggering.

## 2.5 Final comments

In this chapter, we presented the basic theory of discrete event systems that is used in this work. First we introduced the notions of DES, languages, and defined

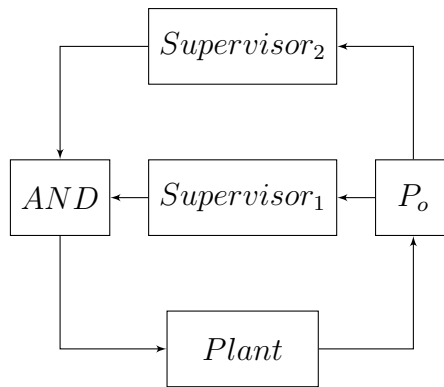


Figure 2.18: Architecture of modular supervisory control.

language operations. Then, we presented the automata as a DES formalism, and also presented automata under partial observation of events. Moreover, we presented the control on discrete event systems and its architecture. The supervisory control theory with partial controllability and under partial observability of events have also been presented. In the next chapter, we present and formalize the safety device for preventing the system from reaching unsafe states, and we show how to obtain a safety device realization from the automaton plant model.

# Chapter 3

## Computation of the safety device

In the supervisory control of DESs, the main objective is to restrict the system behavior in such a way that the system satisfies a set of specifications. In this chapter, the safety device is introduced, as a device that restricts the behavior of controlled discrete event systems, so it cannot reach unsafe or critical states when the system suffers a cyber-attack that change the control action provided by the supervisor, or when it is not implemented or designed correctly.

### 3.1 Formulation of the problem

A DES has, in general, several components, and these components interact in order to complete tasks. This interaction, if incorrectly controlled or if an attack occurs in the system, may damage the system components or may harm operators that are close to the system. These states, associated with dangerous operations of the system, are called in this work unsafe states. Hence, the set of unsafe states is defined as follows.

**Definition 3.1** (*Set of Unsafe States*) *Let  $X$  be the set of states of a given DES. Then, the set of unsafe states  $X_{US}$  is defined as:*

$$X_{US} := \{x \in X : x \text{ is unsafe}\}.$$

■

In addition, behaviors that leads to an unsafe state are called in this work unsafe behaviors, and behaviors that does not leads to unsafe states are called normal behaviors.

**Example 3.1** *Let us consider two robots working on individual tracks with three stations, that share a central station, as depicted in Figure 3.1. The positions of*

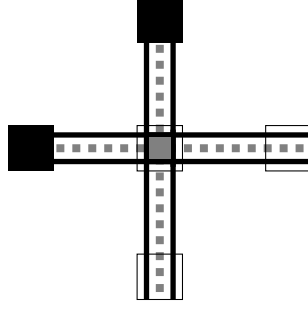
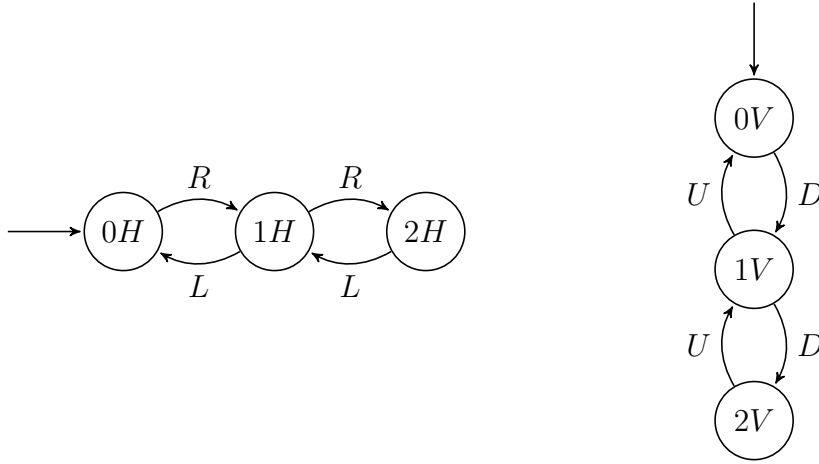


Figure 3.1: Working tracks of robots and robot behavior automata of Example 3.1.



(a) Automaton model of the robot in the horizontal track of Example 3.1. (b) Automaton model of the robot in the vertical track of Example 3.1.

Figure 3.2: Independent automata models of Example 3.1.

each robot can be modeled as in the automata shown in Figure 3.2, where  $0H$ ,  $1H$  and  $2H$  are possible stations of the robot that moves in horizontal direction,  $0V$ ,  $1V$  and  $2V$  are the stations of the robot that moves in vertical direction, and  $R$ ,  $L$ ,  $U$  and  $D$  represents the robots moving right, left, up and down, respectively. The parallel composition of these automata is depicted in Figure 3.3. Notice that state  $(1H, 1V)$  represents a collision between the robots, and thus, in this example,  $X_{US} = \{(1H, 1V)\}$ .

Notice that, in order to prevent the system from reaching unsafe states, one solution is to model a supervisor that disable the last controllable event before all unsafe states. However, depending on the structure of the supervisor and controlled plant, the system can be susceptible to cyber-attacks, which modifies the programmed behavior and may lead the plant to reach unsafe states. Moreover, if the supervisor or controller is frequently reprogrammed, a mistake on the programming may also lead to unsafe states.

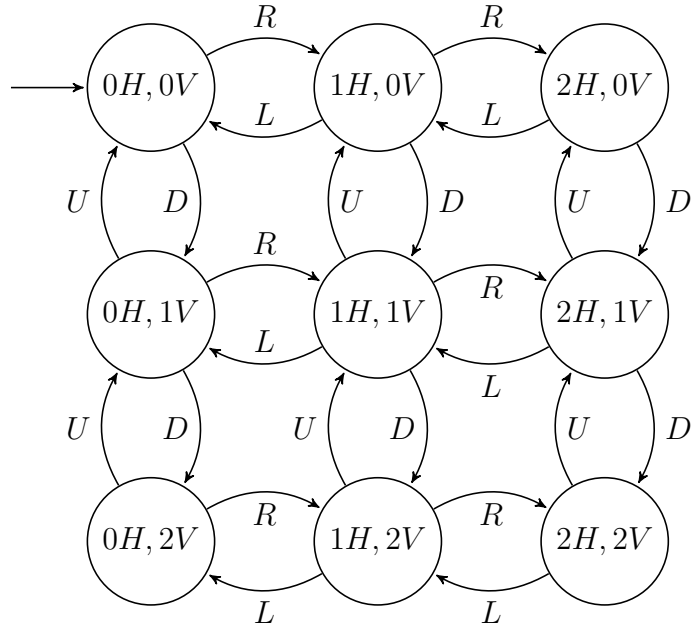


Figure 3.3: Parallel composition of the two automata that model the behavior of the robots, described in Example 3.1.

**Example 3.2** Consider the DES described in Example 3.1. Let  $S$  be a supervisor that restricts the behavior of the robots in such a way that the robots take turns to cross the track, starting with the horizontal robot. Then, the automaton of the supervised plant is shown in Figure 3.4. Hence, if the supervised plant is attacked in such a way that event  $L$  is not disabled in state  $(2H, 1V)$ , then, the plant can reach the unsafe state  $(1H, 1V)$ , as shown in Figure 3.5.

In addition, consider a supervisor attached to the plant that could not be attacked, and assume that this supervisor has been programmed to never let the robots collide, in Example 3.2. Thus, the state transition diagram of a realization of such supervisor is depicted in Figure 3.6. Notice that, if such supervisor is implemented as described, then, the plant never reach the unsafe state, despite the behavior programmed in the controller. Such supervisor can be invulnerable to attacks by a cryptography technique, and can only be modified with a password.

In this work we address the problem of computing a realization of a device that disable transitions that reach unsafe states, and does not prevent the evolution of the programmed control, despite knowing the control behavior designed. By hypothesis, the control behavior is designed to never reach an unsafe state. Then, we call such device as safety device and present how to compute it from the model of the plant.

It is important to remark that since the safety device is directly connected to the plant, it recognizes every observable event and disable every actuator without any intervention of a cyber-attack. This assumption is reasonable, since a control logic can be protected by cryptography, and is only accessed locally and in possession of

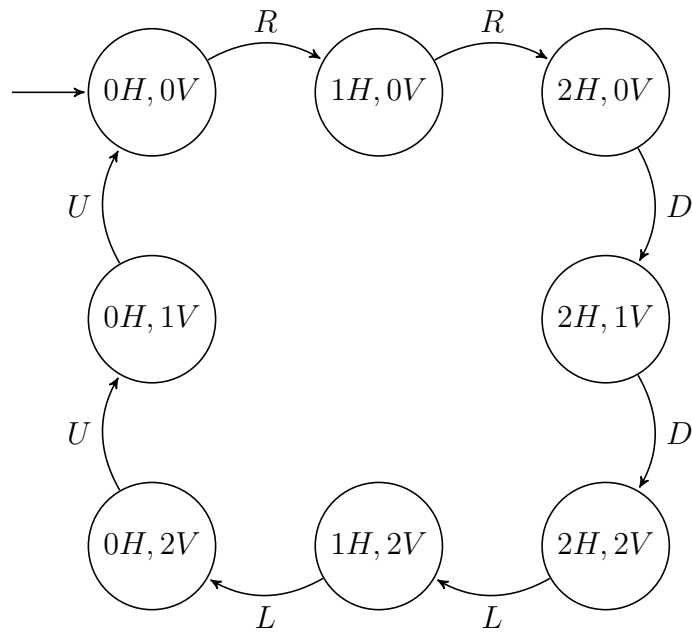


Figure 3.4: Automaton of supervised plant behavior, described in Example 3.2.

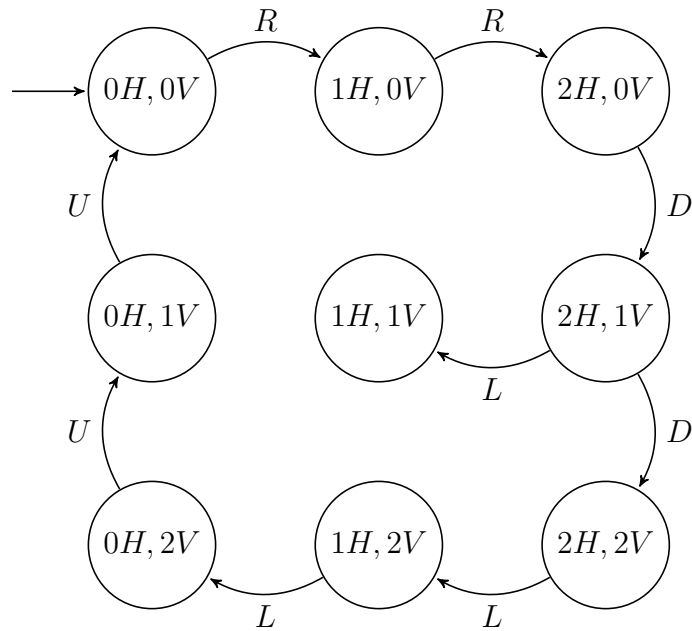


Figure 3.5: Automaton of supervised plant behavior after cyber-attack, described in Example 3.2.

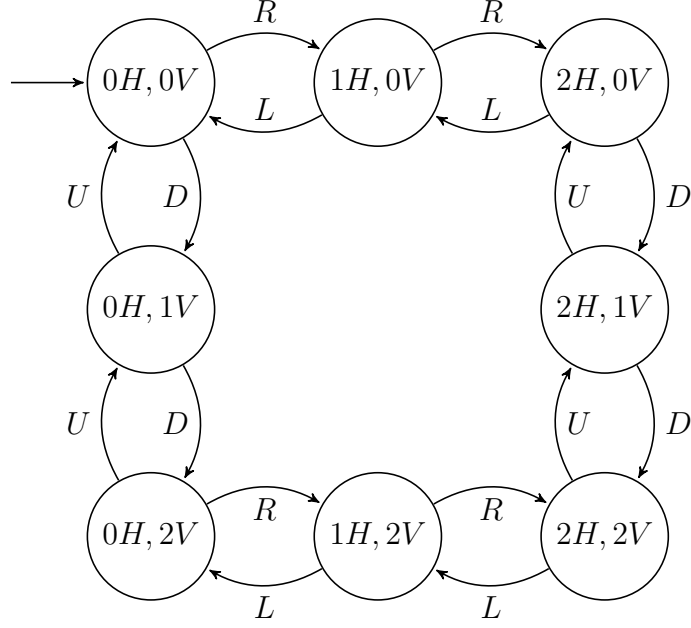


Figure 3.6: Automaton of a realization of a possible supervisor for the plant in Example 3.2.

a password.

The safety device prevents the plant from reaching unsafe states. Therefore, there is an unsafe language  $L_{US} \subseteq \mathcal{L}(G)$  such that for all  $s \in L_{US}$ ,  $\exists t \in \overline{\{s\}}$  such that  $f(x_0, t)$  is an unsafe state, *i.e.*, the unsafe language is formed by all feasible traces in the plant that reach or pass through an unsafe state. Analogously, the safe language is defined as  $L_S = \mathcal{L}(G) \setminus L_{US}$ , which is prefix-closed by definition.

Since the safety device can be implemented in a controlled system, it is necessary to define the architecture of the interaction between safety device, supervisor and plant. The architecture used in this work is the modular supervisory control architecture [2]. This scheme is depicted in Figure 3.7. Moreover, since the safety device is implemented to disable events that lead the system to unsafe states, no other feasible trace in the plant should be blocked. Thus, for all states  $x \in X$ ,  $M(x) \cap S(x) = S(x)$ , since the unknown supervisor, by hypothesis, does not allow the system to reach unsafe states.

The safety device is a function  $M : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$  where, for each trace in  $P_o(\mathcal{L}(G))$ , a combination of events in  $\Sigma$  is disabled by the safety device  $M$ , according to the following rule.

**Rule 3.1** *Let  $G$  be an automaton, where  $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and  $X_{US}$  are the set of unsafe states. Let  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  be a projection function,  $L_S = \mathcal{L}(G) \setminus L_{US}$  be the safe language, and consider the safety device  $M : P_o(\mathcal{L}(G)) \rightarrow 2^\Sigma$ . Then, for each*



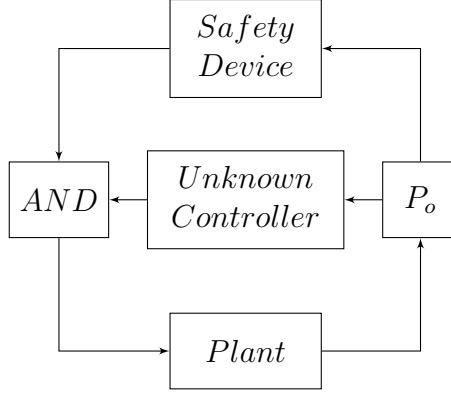


Figure 3.7: Architecture of interaction between plant, supervisor and safety device.

$s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$ ,

$$(P_o^{-1}(s) \cap L_S^{\uparrow C})\{\sigma\} \cap (\mathcal{L}(G) \setminus L_S^{\uparrow C}) \neq \emptyset \Leftrightarrow \sigma \notin M(s).$$

■

Notice that, since the objective of the safety device is to prevent the plant from reaching unsafe states, then a realization of  $M$  is based on the safe language  $L_S$ , and is considered to have an anti-permissive control policy. Thus, the language generated by  $M/G$  never exceeds  $L_S$ .

## 3.2 Computation of a safety device realization

Since the safety device is implemented to disable events that lead the system to unsafe states, no other feasible trace in the plant should be blocked. Thus, for all states  $x \in X$ ,  $M(x) \cap S(x) = S(x)$ , since the unknown supervisor, by hypothesis, does not allow the system to reach unsafe states. Thus, as in [2], the safety device must be permissive for every possible supervisory control applied to the plant.

Before obtaining the algorithm for computing a realization  $W$  for the safety device  $M$ , it is necessary to make the following definition.

**Definition 3.2** (*Reach*) Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma_s \subseteq \Sigma$ . The reach function  $R_{\Sigma_s}^G : X \rightarrow 2^X$  is defined as:

$$R_{\Sigma_s}^G(x) = \{y \in X : (\exists t \in \Sigma_s^*)[f(x, t) = y]\}$$

■

The algorithm for obtaining a realization  $W$  for  $M$  from  $L_S$  is shown in the sequel.

---

**Algorithm 3.1** *Algorithm for computing a realization  $W$  for the safety device*

---

**Input:**  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and the set of unsafe states  $X_{US}$ .

**Output:** Realization  $W$  for  $M$ .

- 1: Compute  $G^T$  as presented in Definition 2.13.
  - 2: Define the unsafe region as the set  $X_{UR} = \bigcup_{x \in X_{US}} R_{\Sigma_{uc}}^{G^T}(x)$ .
  - 3: Define the safe region as the set  $X_S = X \setminus X_{UR}$ .
  - 4: Define  $G_S = (X_S, \Sigma, f_S, \Gamma_S, x_0, X_S)$ , where  $f_S = f|_{X_S \times \Sigma \rightarrow X_S}$  and  $(\forall x \in X_S)(\forall \sigma \in \Sigma)[(\sigma \in \Gamma_S(x)) \iff (f_S(x, \sigma) \text{ is defined})]$ .
  - 5: Define the safe region boundary as the set  $X_B = \{x \in X_S : (\exists \sigma \in \Sigma_c)(\exists y \in X_{UR})[f(x, \sigma) = y]\}$ .
  - 6: Define  $f'_S(x, \sigma) = f_S(x, \sigma)$ ,  $\forall x \in X_S$  and  $\forall \sigma \in \Sigma$ , and  $\Gamma'_S(x) = \Gamma_S(x)$ ,  $\forall x \in X_S$ .
  - 7: For all  $x \in X_B$ , do:
    - 7.1:  $\Sigma' = \{\sigma \in \Sigma_c : (\exists y \in X_{UR})[f(x, \sigma) = y]\}$ .
    - 7.2: For all  $y \in R_{\Sigma_{uo}}^{G^T}(x)$ , do:
      - 7.2.1: For all  $\sigma \in \Sigma'$ , do:
        - 7.2.1.1:  $f'_S(y, \sigma)$  is undefined.
        - 7.2.1.2:  $\Gamma'_S(y) \leftarrow \Gamma'_S(y) \setminus \{\sigma\}$ .
      - 7.2.2: Redefine  $G'_S = (X_S, \Sigma, f'_S, \Gamma'_S, x_0, X_S)$ .
  - 8:  $W = Ac(G'_S)$ .
- 

In the first step of Algorithm 3.1, automaton  $G$  is transposed. In the next step the unsafe region  $X_{UR}$  is defined as the set of all states in  $G^T$  that are reachable from a state in  $X_{US}$  with a trace  $s \in \Sigma_{uc}^*$ . Notice that, from any state in the unsafe region, the safety device no longer have control to prevent the plant from reaching unsafe states. Thus, all states in the unsafe region are removed from automaton  $G$ , forming  $G_S$ . Moreover, in the same sense as a supervisor, the safety device must be controllable and observable. In order to do so, it is necessary to exist an observable event before the disabled controllable event. Thus, in Step 7 of Algorithm 3.1, from

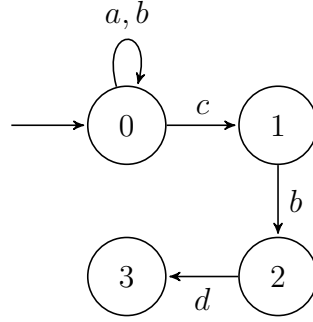


Figure 3.8: Automaton  $G$  of Example 3.3.

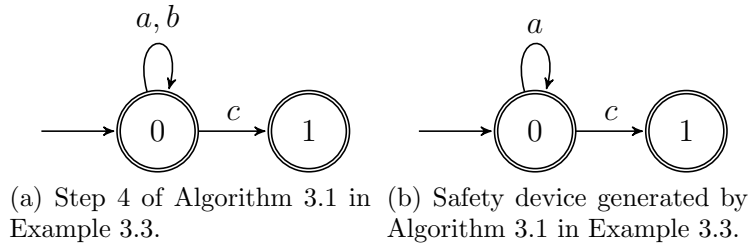


Figure 3.9: Final steps of Algorithm 3.1 in Example 3.3.

each state  $x$  in the set of boundary safe states, all states that are reachable from a trace  $s \in \Sigma_{uo}^*$  are found, and the same controllable events that the safety device disables in  $x$  are disabled. Then, state  $f(x, \sigma)$  is added to the set of boundary safe states and Step 7 is repeated. Notice that the event  $\sigma$  disabled in  $x$  prevents the plant from reaching unsafe states from state  $x$ . Moreover, if state  $x$  is reached after the occurrence of an unobservable event, then, the safety device does not recognize if the plant is in state  $x$ , and must disable  $\sigma$  before reaching state  $x$ . However, disabling  $\sigma$  may prevent the system from reaching states in the normal behavior, and this problem is addressed in the following sections.

**Example 3.3** Let  $G$  be the automaton depicted in Figure 3.8, where  $\Sigma_{uo} = \{c\}$  and  $\Sigma_{uc} = \{d\}$ , and let  $X_{US} = \{3\}$  be the set of unsafe states. Then, as described in Algorithm 3.1, define the unsafe region as  $X_{UR} = \{2, 3\}$ , and remove all states in the unsafe region from  $G$ , as shown in Figure 3.9a. Notice that the automaton in Figure 3.9a disables event  $b$  when on state 1. However,  $c$  is unobservable and the safety device would not be capable of distinguishing between states 1 and 0. If the safety device cannot distinguish states 1 and 0, then, it must also disable event  $b$  in state 0. Thus, in this example, the set of boundary safe states is  $\{0, 1\}$  and a realization  $W$  is depicted in Figure 3.9b.

In order to prove the existence of a safety device realization, and the correctness of Algorithm 3.1, a lemma is presented in the sequel.

**Lemma 3.1** Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and let  $X_{US} \subseteq X$  be the set of unsafe states. Let  $L_{US} = \{s \in \mathcal{L}(G) : (\exists t \in \overline{\{s\}})[f(x_0, t) \in X_{US}]\}$ ,  $L_S = \mathcal{L}(G) \setminus L_{US}$ , and  $W = (Y, \Sigma, g, \Gamma_W, y_0, Y)$  be a realization of the safety device  $M$  for plant  $G$ , obtained according to Algorithm 3.1. Then,  $\mathcal{L}(W) \subseteq L_S^{\uparrow C}$ .

*Proof:* Notice that, according to the definition of  $L_S$ , a realization for  $L_S$  should be obtained by eliminating all states in  $X_{UR}$  of  $G$ , and their associated transitions, and then, taking its accessible part. This procedure is carried out in Steps 1 to 4 of Algorithm 3.1, generating automaton  $G_S$ . Thus,  $L_S^{\uparrow C} = \mathcal{L}(G_S)$ .

Since  $W$  is computed from  $G_S$  by removing transitions labeled with uncontrollable events, and taking the accessible part of the resulting automaton, then  $\mathcal{L}(W) \subseteq \mathcal{L}(G_S) = L_S^{\uparrow C}$ , which concludes the proof. ■

**Theorem 3.1 (Existence)** Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and let  $X_{US} \subseteq X$  be the set of unsafe states. Let  $L_{US} = \{s \in \mathcal{L}(G) : (\exists t \in \overline{\{s\}})[f(x_0, t) \in X_{US}]\}$ , and  $L_S = \mathcal{L}(G) \setminus L_{US}$ , where  $L_S^{\uparrow C} \neq \emptyset$ . Then, there exists an automaton  $W = (Y, \Sigma, g, \Gamma_W, y_0, Y)$  as a realization of the safety device  $M$  for plant  $G$ , obtained according to Algorithm 3.1.

*Proof:* Notice that,  $L_S^{\uparrow C} = \emptyset$  if, and only if, exists  $s \in \Sigma_{uc}^*$  such that  $f(x_0, s) \in L_{US}$ . According to Step 3 of Algorithm 3.1,  $x_0 \in X_S$ , since  $L_S^{\uparrow C} \neq \emptyset$  by hypothesis. Moreover, notice that, in Algorithm 3.1, no other state is removed from the automaton, and then, in the worst case,  $y_0 = x_0$ , which concludes the proof. ■

**Theorem 3.2 (Correctness)** Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , and let  $X_{US} \subseteq X$  be the set of unsafe states. Let  $P_o : \Sigma^* \rightarrow \Sigma_o^*$  be a projection function,  $L_{US} = \{s \in \mathcal{L}(G) : (\exists t \in \overline{\{s\}})[f(x_0, t) \in X_{US}]\}$ ,  $L_S = \mathcal{L}(G) \setminus L_{US}$ , and  $W = (Y, \Sigma, g, \Gamma_W, y_0, Y)$  be a realization of the safety device  $M$  for plant  $G$ , obtained according to Algorithm 3.1. Then, for each  $s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$ ,

$$(P_o^{-1}(s) \cap L_S^{\uparrow C})\{\sigma\} \cap (\mathcal{L}(G) \setminus L_S^{\uparrow C}) \neq \emptyset \Leftrightarrow (\forall t \in P_o^{-1}(s) \cap \mathcal{L}(G))[\sigma \notin \Gamma_W(g(y_0, t))]$$

■

*Proof:* ( $\Rightarrow$ ) Assume that there exist  $s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$  such that  $(P_o^{-1}(s) \cap L_S^{\uparrow C})\{\sigma\} \cap (\mathcal{L}(G) \setminus L_S^{\uparrow C}) \neq \emptyset$ . Then, there exists  $u \in P_o^{-1}(s)$  such that  $u \in L_S^{\uparrow C}$ , and  $u\sigma \in \mathcal{L}(G) \setminus L_S^{\uparrow C}$ . Let us assume now that there exists  $t \in P_o^{-1}(s) \cap \mathcal{L}(G)$  such that  $\sigma \in \Gamma_W(g(y_0, t))$ . Then, according to Step 5 of Algorithm 3.1,  $f(x_0, u) \in X_B$ , i.e., state  $f(x_0, u)$  is in the safe region boundary, which implies that  $f(x_0, u\sigma) \in X_{UR}$ . According to Step 7 of Algorithm 3.1, all states reached from traces with the same

observation as  $u$ , must have event  $\sigma$  disabled, which contradicts the assumption that there exists  $t \in P_o^{-1}(s) \cap \mathcal{L}(G)$  such that  $\sigma \in \Gamma_W(g(y_0, t))$ .

( $\Leftarrow$ ) If there exists  $s \in P_o(\mathcal{L}(G))$  and  $\sigma \in \Sigma$  such that  $(P_o^{-1}(s) \cap L_S^{\uparrow C})\{\sigma\} \cap (\mathcal{L}(G) \setminus L_S^{\uparrow C}) = \emptyset$ , then, according to Algorithm 3.1,  $[(P_o^{-1}(s) \cap L_S^{\uparrow C})\{\sigma\}] \cap \mathcal{L}(G) \subseteq \mathcal{L}(W)$ . Thus,  $(\forall t \in P_o^{-1}(s) \cap \mathcal{L}(G)), \sigma \in \Gamma_W(g(y_0, t))$ , which concludes the proof. ■

Notice that the sufficient condition is proved by showing that the safety device is anti-permissive, *i.e.*, disable event  $\sigma$  after all traces  $t \in P_o^{-1}(s) \cap \mathcal{L}(G)$  such that  $s$  exceeds the safe language  $L_S$ . Moreover, the necessary condition is proved by showing that the safety device only disables event  $\sigma$  if there exists a trace  $t \in P_o^{-1}(s) \cap \mathcal{L}(G)$ , such that  $t\sigma \notin L_S^{\uparrow C}$ .

According to Lemma 3.1, the safety device can block traces in  $L_S$ . This blocking can be detrimental or not for the system. To analyze the effect of the safety device on the system, we define in the sequel levels of influence of the safety device over the safe language of the system. These levels are defined over the automaton of the plant and its marked states. There are four levels, defined as follows:

**Definition 3.3** (*Safety Levels*) *Let  $G$  be an automaton, where  $X_m$  is the set of marked states, and let  $X_{US}$  be the set of unsafe states. Notice that, by definition, all safety devices block all traces in  $L_{US}$  and, by hypothesis,  $L_{US} \cap \mathcal{L}_m(G) = \emptyset$ . Then, the following safety levels can be defined:*

- *Level 1: the safety device blocks only traces in  $L_{US}$ ;*
- *Level 2: the safety device might block some traces in  $\mathcal{L}(G) \setminus [\mathcal{L}_m(G) \cup L_{US}]$ ;*
- *Level 3: the safety device might block some traces in  $\mathcal{L}_m(G)$ , but not all traces in  $L_m(G)$ ; and*
- *Level 4: the safety device blocks all traces in  $\mathcal{L}_m(G)$ .*

■

Notice that a safety device in level 1 is also in levels 2 and 3, and a safety device in level 2 is also in level 3. However, there are safety devices that are in level 2 and are not in level 1, and also there exist safety devices that are in level 3 and not in level 2. Moreover, safety device in level 4 are neither in levels 1, 2 nor 3. This relation defines an inclusion characteristic.

Notice that, it is necessary to verify the safety level of device  $M$  after computing a safety device realization  $W$  from Algorithm 3.1, to conclude the application on the plant. Due to the inclusion characteristic of levels, the safety level verifications must be done from level 1 to level 4.

The verification of the safety level is computed as follows:

---

**Algorithm 3.2** *Algorithm for safety level verification of realization  $W$*

---

**Input:**  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ , where  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$ , the set of unsafe states  $X_{US}$ , and safety device realization  $W$  obtained from Algorithm 3.1.

**Output:** Level 1, 2, 3 or 4.

1: Define  $f' = f|_{X \setminus X_S \times \Sigma \rightarrow X \setminus X_{US}}$ ,  $\Gamma' = \Gamma|_{X \setminus X_{US} \rightarrow \Sigma}$ , and  $G' = (X \setminus X_{US}, \Sigma, f', \Gamma', x_0, X_m)$

2: Define  $G_S = Ac(G')$

3: Define  $W^{(m)} =, G_S^{(m)} =, Trim(G)^{(m)}$

4: If  $(W^{(m)})^{comp} \times G_S^{(m)}$  and  $W^{(m)} \times (G_S^{(m)})^{comp}$  have no marked states, then:

4.1: Return Layer 1.

4.2: STOP.

5: If  $(Trim(G)^{(m)})^{comp} \times W^{(m)}$  has nor marked states, then:

5.1: Return Layer 2.

5.2: STOP.

6: If  $W \parallel G$  does not have marked states, then:

6.1: Return Layer 3.

6.2: STOP.

7: Return Layer 4.

---

Algorithm 3.2 verifies the safety level of realization  $W$ . In the first step of Algorithm 3.2, all unsafe states and its associated transitions are removed from plant automaton  $G$ , and the accessible part  $G_S = Ac(G')$  of the reduced automaton  $G'$  is taken. In the next step, it is verified if languages  $\mathcal{L}(W)$  and  $L_S$  are the same, by computing its respective automata with all states marked, and the product composition is computed between  $(W^{(m)})^{comp}$  and  $G_S^{(m)}$ , then, between  $W^{(m)}$  and  $(G_S^{(m)})^{comp}$ . If there are no marked states in  $(W^{(m)})^{comp} \times G_S^{(m)}$  and  $W^{(m)} \times (G_S^{(m)})^{comp}$ , which certify that  $\mathcal{L}(W) = L_S$ , *i.e.*, the safety device only blocks traces in  $L_{US}$ . Then, safety device realization  $W$  is concluded to be in safety level 1. Else, the set inclusion  $\mathcal{L}_m(G) \subseteq \mathcal{L}(W)$  is verified. Automata  $W$  and  $Trim(G)$  are used for this verification, since  $\mathcal{L}(Trim(G)) = \overline{\mathcal{L}_m(G)}$ . Then, the product composition between  $(Trim(G)^{(m)})^{comp}$  and  $W^{(m)}$  is computed. If  $(Trim(G)^{(m)})^{comp} \times W^{(m)}$  has no marked

states, then, realization safety device  $W$  is confirmed to be in safety level 2, since this concludes that all traces in  $\mathcal{L}_m(G)$  are feasible in  $W$ . However, in case there are marked states in  $(Trim(G)^{(m)})^{comp} \times W^{(m)}$ , then,  $W\|G$  is verified. If  $W\|G$  has no marked states, then  $W$  is considered to be in safety level 4, else,  $W$  is in safety level 3.

**Example 3.4** *Let  $G_1, G_2, G_3,$  and  $G_4$  be the automata depicted in Figures 3.10a, 3.10b, 3.10c, and 3.10d, respectively. Moreover, consider that all four automata have the event set defined as  $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo} = \{a, b, c, d, e\}$ , where  $\Sigma_{uc} = \{d\}$ , and  $\Sigma_{uo} = \{e\}$ , and also, consider that state 3 is unsafe. Then, their respective safety device realizations  $W_1, W_2, W_3,$  and  $W_4$ , shown in Figure 3.11, are on safety levels 1, 2, 3, and 4, respectively. Notice that,  $G_1$  has a controllable event before the unsafe state, and, then, all traces in the safe language are feasible. Also, safety device  $W_1$  allows  $G_1$  to reach state 2, which is a deadlock, since restricting deadlock behaviors is not the intention of the safety device. Automaton  $G_2$  has unsafe state 3 and unsafe region state  $\{2, 3\}$ , which are blocked by  $W_2$ , and state 2 is reachable by the safe language. In addition, automaton  $G_3$  also has a controllable event before the unsafe state, but event  $e$  is unobservable, and it is impossible to distinguish between states 0 and 2. Then,  $W_3$  disables event  $b$  in states 0 and 2, which blocks some traces in  $\mathcal{L}_m(G_3)$ . Finally, realization  $W_4$  also disables event  $b$  in states 0 and 2 in automaton  $G_4$ , however all traces in  $\mathcal{L}_m(G_4)$  are blocked.*

### 3.3 Final comments

In this chapter, we presented the safety device for preventing the controlled plant to reach unsafe states, where the control of the plant is unknown. Then, we presented the architecture used for safety device, control, supervisor and plant interaction. Also, we formalized a realization of the safety device and showed how to obtain this realization from the plant model. Moreover, safety devices can be classified in safety levels and the algorithm for verification of safety levels is also presented in this chapter. In the next chapter we test the developed theory in a mechatronic system and implement the safety device for preventing control configurations from damaging the plant.

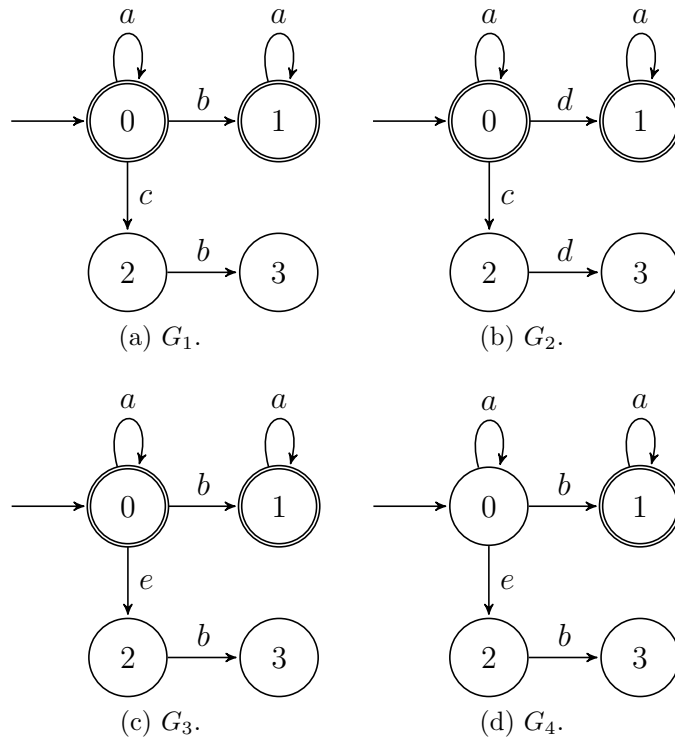


Figure 3.10: Automata  $G_1$ ,  $G_2$ ,  $G_3$ , and  $G_4$  used in Example 3.4.

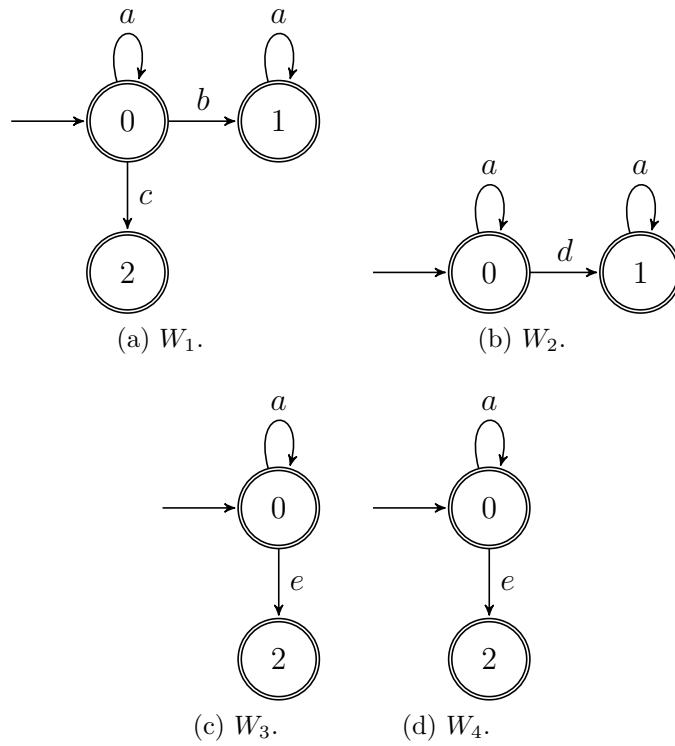


Figure 3.11: Monitors  $M_1$ ,  $M_2$ ,  $M_3$ , and  $M_4$  of  $G_1$ ,  $G_2$ ,  $G_3$ , and  $G_4$ , respectively, as presented in Example 3.4.



# Chapter 4

## Implementation of the safety device in a mechatronic system

In this chapter, we describe the cube assembling mechatronic system used as the illustrative example for the application of the safety device. This mechatronic system is a module of the industrial cube assembly and storage system that can be found at the Control and Automation Laboratory (LCA) of the Federal University of Rio de Janeiro (UFRJ). The safety device was implemented using the Siemens software, Totally Integrated Automation Portal (TIA Portal).

### 4.1 Mechatronic system

The plant is a Cube Assembly mechatronic system, built by Christiani Technical Institute for Vocation Training [51]. This plant has three connected modules, as shown in Figure 4.1, which represent functionalities that simulate industrial processes, as raw material selection, processing, and storage. However, only module 2, depicted in Figure 4.2, is used in this work. Module 2 is responsible for transporting the cube pieces between modules, and assembling cubes. The transportation is done by a robotic arm, which has a pneumatic mechanism that activates a suction cup in order to pick up, transport and deliver cube pieces. The assembling of two pieces is done by a press, which also has pneumatic mechanisms.

### 4.2 Plant model

In order to build a safety device to the plant, it is necessary to model all feasible events of the plant without control. Thus, the automaton model must represent physical restrictions of actuators, and space limitations of queues. The processing module has six pneumatic actuators, three in the press and three in the robotic arm.

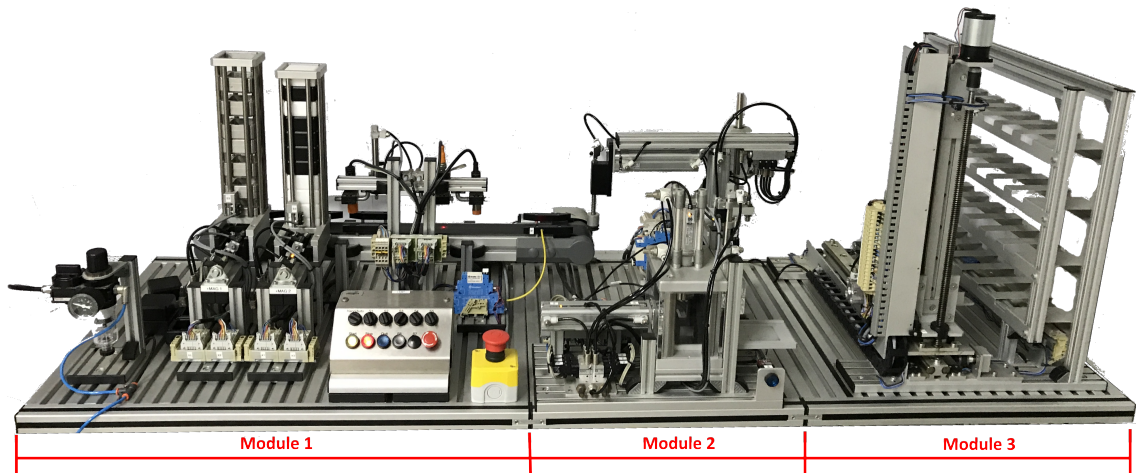


Figure 4.1: Module division of Cube Assembly plant.

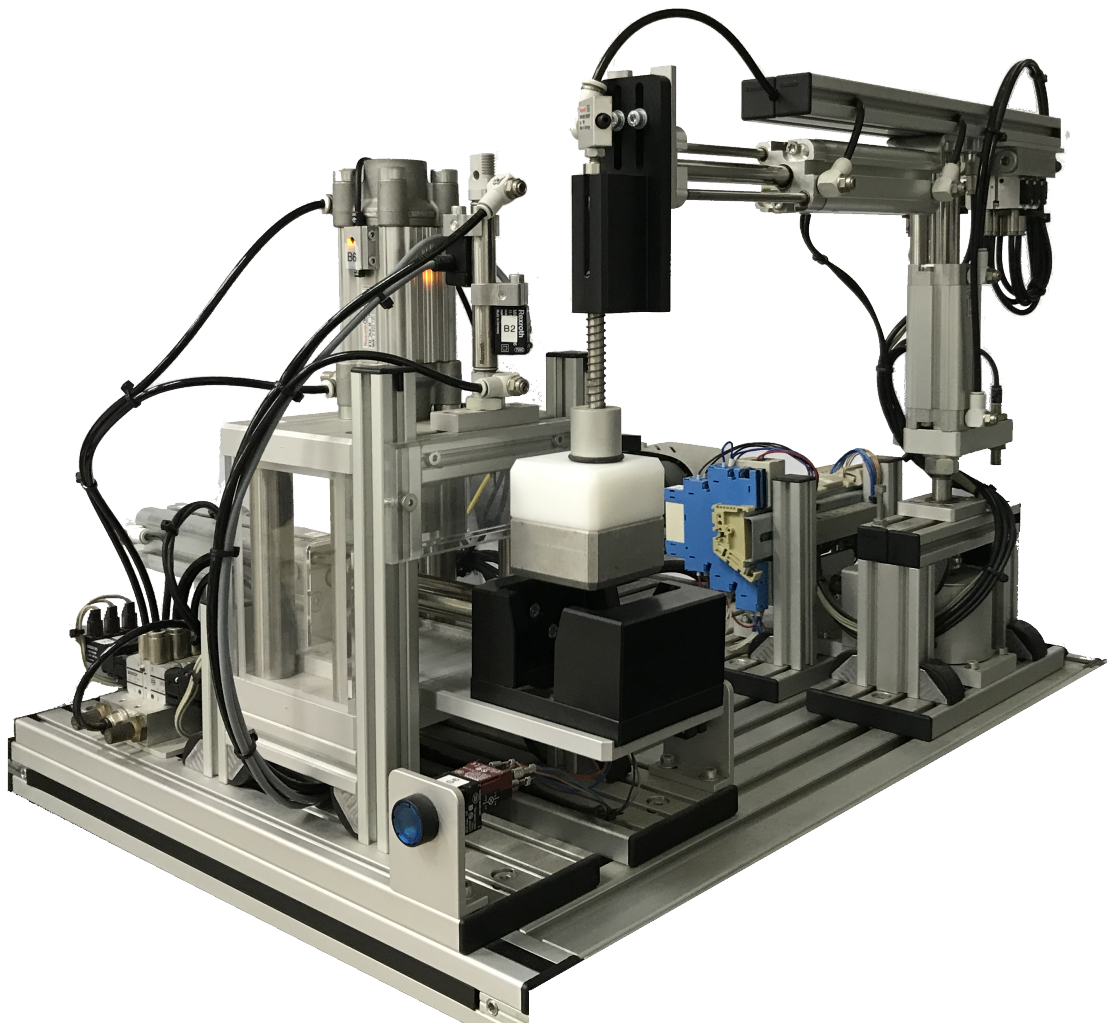


Figure 4.2: Processing module of Cube Assembly plant.

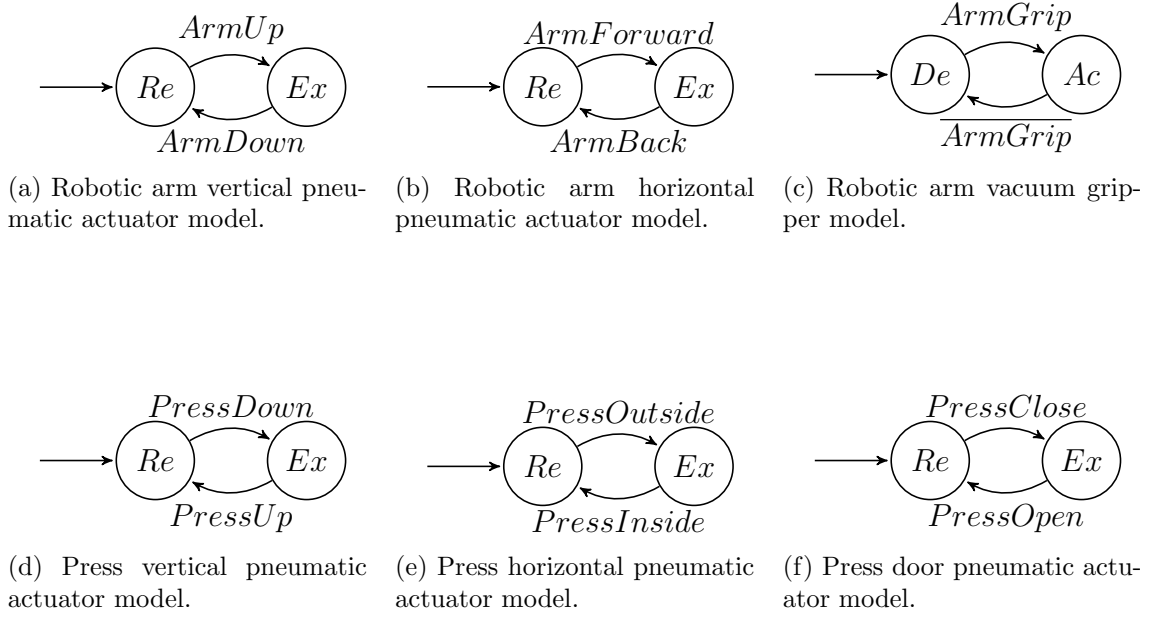


Figure 4.3: Pneumatic actuators of module 2 of the plant and robotic arm vacuum gripper models.

Each pneumatic actuator is modeled as an automaton with two states, extended and retracted, as depicted in Figure 4.3. In these automata, states  $Re$  and  $Ex$  represent that the pneumatic actuator is retracted and extended, respectively. Also, states  $De$  and  $Ac$  in Figure 4.3c represent the vacuum gripper deactivated and activated, respectively. Moreover, since the states are retracted or extended, and deactivated or activated, then, the label of the state transition is the signal emitted to the pneumatic actuator. Thus, the state transitions label abbreviations in Figure 4.3 are explained in Table 4.1.

The robotic arm rotates on its axis and has three important positions: calibrating, facing the conveyor, and facing the press. Each position has a relative angle that is measured using an encoder, which are  $0^\circ$ ,  $90^\circ$ , and  $180^\circ$ , respectively. As a redundancy to the encoder, there is an inductive sensor that identify a certain position of the robotic arm and calibrates the encoder to  $0^\circ$ . Thus, besides the states that refer to angle positions, the robotic arm rotates clockwise and counter-clockwise. Then, the automaton model of the robotic arm rotation is depicted in Figure 4.4. Each state and state transition label has a physical meaning that are described in Tables 4.2 and 4.3, respectively.

Algorithm 3.1 needs both automaton model of the plant and all unsafe states in the model. The complete automaton model of the plant is the parallel composition of the automata in Figures 4.3 and 4.4. However, the state transition diagram is

Table 4.1: Physical meaning of state transition labels of the automata in Figure 4.3.

Abbreviation	Meaning	Figure
<i>ArmUp</i>	Vertical pneumatic actuator moves the robotic arm up.	4.3a
<i>ArmDown</i>	Vertical pneumatic actuator moves the robotic arm down.	4.3a
<i>ArmForward</i>	Horizontal pneumatic actuator moves the robotic arm forward.	4.3b
<i>ArmBack</i>	Horizontal pneumatic actuator moves the robotic arm back.	4.3b
<i>ArmGrip</i>	Activates the vacuum gripper of the robotic arm.	4.3c
<i>PressDown</i>	Vertical pneumatic actuator moves the press down to assemble cube pieces.	4.3d
<i>PressUp</i>	Vertical pneumatic actuator moves the press up.	4.3d
<i>PressOutside</i>	Horizontal pneumatic actuator moves the press outside of the case.	4.3e
<i>PressInside</i>	Horizontal pneumatic actuator moves the press inside of the case.	4.3e
<i>PressClose</i>	Door's pneumatic actuator closes the case.	4.3f
<i>PressOpen</i>	Door's pneumatic actuator opens the case.	4.3f

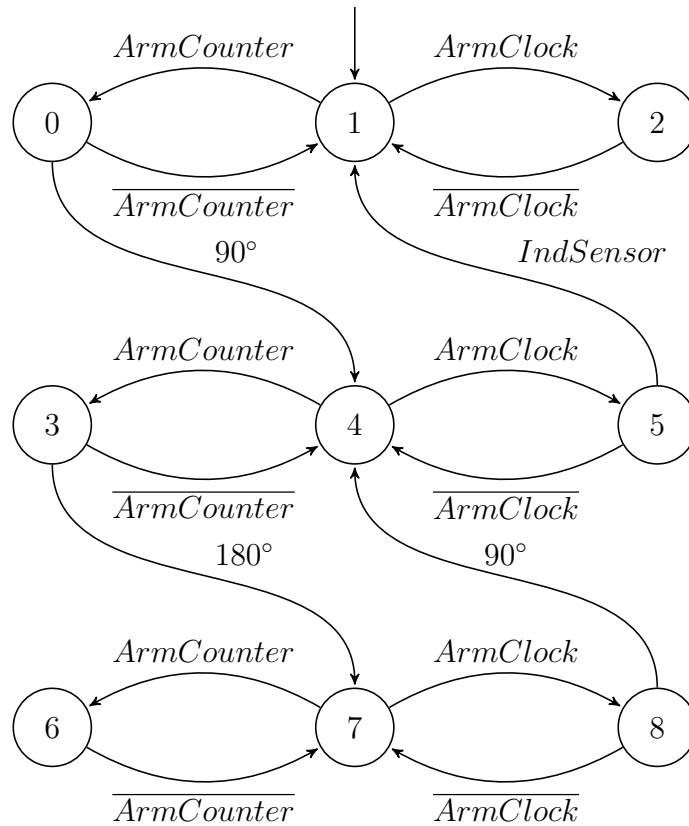


Figure 4.4: Automaton model of the robotic arm rotation.

Table 4.2: Physical meaning of states of the automata in Figure 4.4.

States	Physical Meaning
0, 1, and 2	Robotic arm last faced $0^\circ$ .
3, 4, and 5	Robotic arm last faced $90^\circ$ .
6, 7, and 8	Robotic arm last faced $180^\circ$ .
0, 3, and 6	Robotic arm rotating counter-clockwise.
1, 4, and 7	Robotic arm not rotating.
2, 5, and 8	Robotic arm rotating clockwise.

Table 4.3: Physical meaning of state transition label abbreviations of the automata in Figure 4.4.

Abbreviation	Physical Meaning
<i>ArmCounter</i>	Robotic arm starts rotating counter-clockwise.
<i>ArmClock</i>	Robotic arm starts rotating clockwise.
<i>IndSensor</i>	Inductive sensor identify a metallic totem that represents $0^\circ$ position.
$90^\circ$	Robotic arm encoder reach the value that represents $90^\circ$ position.
$180^\circ$	Robotic arm encoder reach the value that represents $180^\circ$ position.

not shown in this work, since this parallel composition results in an automaton with 576 states. Thus, the unsafe states are represented by a combination of states in different automata. All unsafe state combinations are listed in Table 4.4.

It is important to remark that some unsafe behaviors are based on transitions, and not on states. Those unsafe transitions lead to a state that might be reached also by safe transitions. An unsafe transition is defined as a state transition that may cause damage to the system components or may harm operators that are close to the system, even if it evolves the system from a safe state to another safe state. Thus, in order to distinguish those behaviors, the plant needs to be slightly changed using a state splitting procedure [2], as defined in the sequel.

**Definition 4.1** (*State Splitting*) *Let  $G = (X, \Sigma, f, \Gamma, x_0, X_m)$  and let  $(x, \sigma)$  be the unsafe transition, where  $x \in X$ ,  $\sigma \in \Sigma$ , and  $f(x, \sigma) \in X$ . Then, automaton  $G'$  that distinguish  $(x, \sigma)$  from other transitions that reach  $f(x, \sigma)$  is formed by the following steps:*

- 1: Define  $X' = X \cup \{x'\}$  and  $f'(y, \omega) = f(y, \omega)$ , for all  $y \in X$  and for all  $\omega \in \Sigma$ .
- 2:  $f'(x, \sigma) \leftarrow x'$ .
- 3:  $G' = (X', \Sigma, f', \Gamma, x_0, X_m)$

■

The state splitting procedure presented in Definition 4.1 adds an unsafe state that is only reachable by the unsafe transition. Hence, in order to obtain the state

Table 4.4: Unsafe state combinations of the plant.

State Combinations (Figure,State)	Physical Meaning	Consequence of Reaching Unsafe State Combination
(4.4,2)	Robotic arm in the $0^\circ$ region, and rotating clockwise.	Robotic arm collides with other components of the mechatronic system.
(4.4,6)	Robotic arm in the $180^\circ$ region, and rotating counter-clockwise.	Robotic arm collides with other components of the mechatronic system.
(4.3d,Ex) (4.3e,Ex)	Press horizontal and vertical pneumatic actuators extended.	Press pneumatic actuators collide.
(4.3e,Ex) (4.3f,Ex)	Press horizontal pneumatic actuator extended, and door closed.	Press horizontal actuator and door collide.
(4.3b,Ex) (4.4,3)	Robotic arm horizontal pneumatic actuator extended, and rotating counter-clockwise in the $90^\circ$ region.	Robotic arm collides with press structure.
(4.3b,Ex) (4.4,8)	Robotic arm horizontal pneumatic actuator extended, and rotating clockwise in the $180^\circ$ region.	Robotic arm collides with press structure.
(4.3a,Re) (4.3b,Ex) (4.4,5)	Robotic arm horizontal pneumatic actuator extended, vertical pneumatic actuator retracted and rotating clockwise in the $90^\circ$ region.	Robotic arm collides with other components of the mechatronic system.
(4.3a,Re) (4.3b,Ex) (4.4,0)	Robotic arm horizontal pneumatic actuator extended, vertical pneumatic actuator retracted and rotating counter-clockwise in the $0^\circ$ region.	Robotic arm collides with other components of the mechatronic system.

transition diagram of the plant that distinguish all unsafe behaviors by using states, then it is necessary to apply the state splitting procedure for each unsafe transition described in Table 4.5.

Since, all necessary inputs for Algorithm 3.1 are already computed and described, we can now obtain realization  $W$  for the interaction behavior of the robotic arm and the press. Since the complete automaton model of the system has 582 states, it is not depicted in this work. Then, for clarification reasons, we break the system model in three subsystems that are presented in the sequel.

**Example 4.1** *The first example is regarding the robotic arm rotation automaton subsystem depicted in Figure 4.4. There are two unsafe states in the robotic arm rotation behavior, states 2 and 6, according to Table 4.4. Thus, in order to compute realization  $W$  using Algorithm 3.1, we must first transpose the plant automaton and then find the unsafe region by the computation of the uncontrollable reach. Notice that, all events are controllable and observable, which implies that the safety device is at safety level 1. Hence, safety device realization  $W$ , depicted in Figure 4.5, is obtained by removing all unsafe states of the automaton. Notice that, the implemented safety device needs to disable events  $ArmClock$  in state 1, and  $ArmCounter$  in state 7.*

**Example 4.2** *The second example is regarding horizontal and vertical pneumatic actuators subsystems of the press. The automaton that models the behavior of this subsystem is the parallel composition of the automata depicted in Figures 4.3d and 4.3e, and the resulting automaton is shown in Figure 4.6. Only state  $(Ex, Ex)$  is an unsafe state in the press horizontal and vertical pneumatic actuators behavior, according to Table 4.4. Again, all events in the automaton plant are controllable and observable, which implies that the safety device is also at safety level 1. Hence, safety device realization  $W$ , depicted in Figure 4.7, is obtained by removing all unsafe states of the automaton. Notice that, the implemented safety device needs to disable events  $PressDown$  in state  $(Ex, Re)$ , and  $PressOutside$  in state  $(Re, Ex)$ .*

**Example 4.3** *The third example is regarding the robotic arm horizontal and vertical pneumatic actuators, and robotic arm rotation subsystem. The automaton that models the behavior of this subsystem is the parallel composition of the automata depicted in Figures 4.3a, 4.3b, and 4.4, and the resulting automaton is shown in Figure 4.8. There are two unsafe transitions in the robotic arm behavior, transition  $ArmForward$  from state  $(Re, Re, 4)$ , and  $ArmBack$  from state  $(Re, Ex, 4)$ , according to Table 4.5. Since the unsafe behavior is an unsafe transition, it is necessary to apply the state splitting procedure, and the resulting automaton is depicted in Figure*

Table 4.5: Unsafe transitions of the plant.

Previous State Combinations (Figure,State)	Unsafe Transition	Physical Meaning	Consequence of Unsafe Transition
(4.3a, <i>Re</i> ) (4.3b, <i>Re</i> ) (4.4,4)	<i>ArmForward</i>	Robotic arm stopped in the 90° region, horizontal and vertical pneumatic actuators retracted.	Robotic arm collides with other components of the mechatronic system.
(4.3a, <i>Re</i> ) (4.3b, <i>Ex</i> ) (4.4,4)	<i>ArmBack</i>	Robotic arm stopped in the 90° region, horizontal pneumatic actuator extended and vertical pneumatic actuator retracted.	Robotic arm collides with other components of the mechatronic system.
(4.3a, <i>Re</i> ) (4.3b, <i>Ex</i> ) (4.3e, <i>Re</i> ) (4.4,7)	<i>PressOutside</i>	Robotic arm stopped in the 180° region, horizontal pneumatic actuator extended, and vertical pneumatic actuator retracted. Press horizontal pneumatic actuator retracted.	Press horizontal pneumatic actuator collides with robotic arm.
(4.3a, <i>Re</i> ) (4.3b, <i>Ex</i> ) (4.3e, <i>Ex</i> ) (4.4,7)	<i>PressInside</i>	Robotic arm stopped in the 180° region, horizontal pneumatic actuator extended, and vertical pneumatic actuator retracted. Press horizontal pneumatic actuator Extended.	Press horizontal pneumatic actuator collides with robotic arm.
(4.3a, <i>Re</i> ) (4.3b, <i>Re</i> ) (4.3e, <i>Ex</i> ) (4.4,7)	<i>ArmForward</i>	Robotic arm stopped in the 180° region, horizontal and vertical pneumatic actuators retracted. Press horizontal pneumatic actuator Extended.	Robotic arm horizontal pneumatic actuator collides with press.
(4.3a, <i>Re</i> ) (4.3b, <i>Ex</i> ) (4.3e, <i>Ex</i> ) (4.4,7)	<i>ArmBack</i>	Robotic arm stopped in the 180° region, horizontal pneumatic actuator extended and vertical pneumatic actuator retracted. Press horizontal pneumatic actuator Extended.	Robotic arm horizontal pneumatic actuator collides with press.



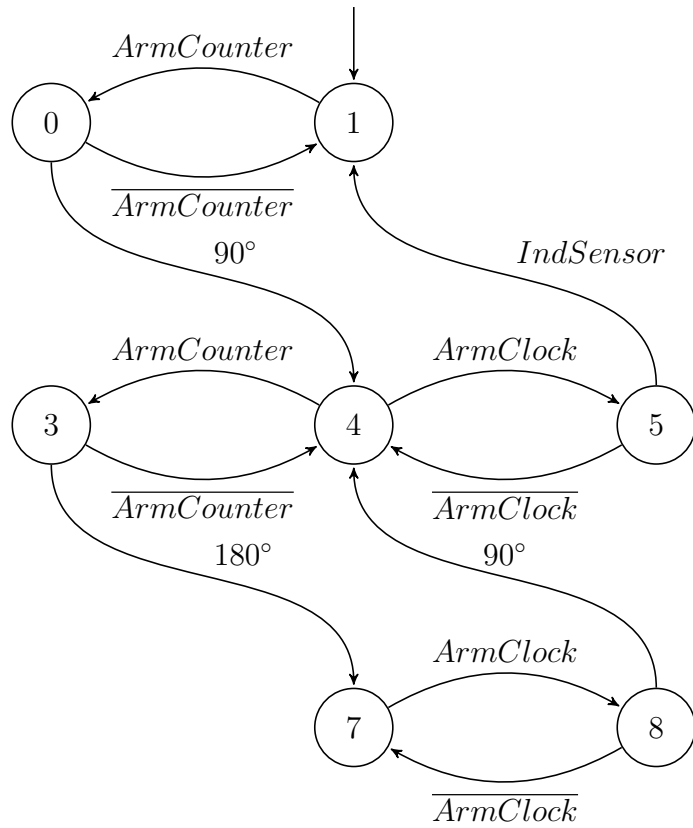


Figure 4.5: State transition diagram of safety device realization  $W$ , obtained by Algorithm 3.1 on the automaton of the robotic arm rotation subsystem, as described in Example 4.1.

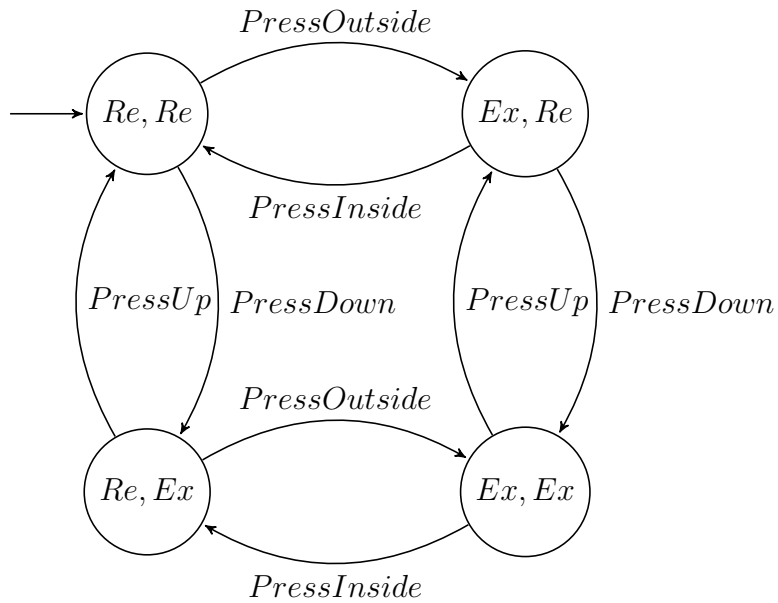


Figure 4.6: Parallel composition of automata depicted in Figures 4.3d and 4.3e, as described in Example 4.2.

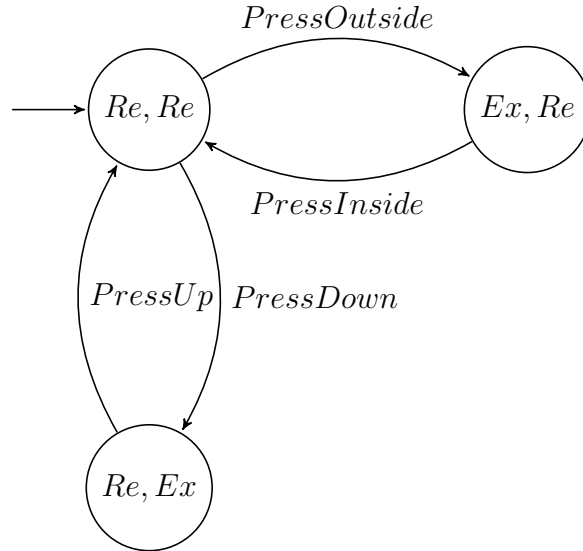


Figure 4.7: State transition diagram of safety device realization  $W$ , obtained by Algorithm 3.1 on the automaton of the press horizontal and vertical pneumatic actuators subsystem, as described in Example 4.2.

4.9. In addition, all events in the automaton plant are controllable and observable, which implies that the safety device is at safety level 1. Hence, safety device realization  $W$ , depicted in Figure 4.10, is obtained by removing all unsafe states of the automaton. Notice that, the implemented safety device needs to disable events  $ArmForward$  in state  $(Re, Re, 4)$ , and  $ArmBack$  in state  $(Re, Ex, 4)$ .

Examples 4.1, 4.2 and 4.3 cover all possible difficulties related to Algorithm 3.1, and, now it is possible to compute Algorithm 3.1 for the parallel composition of the automata depicted in Figures 4.3 and 4.4, and the unsafe states described in Tables 4.4 and 4.5.

### 4.3 Implementation and final comments

In order to implement the safety device in a programmable logic controller, it is first necessary to compute the observer of the system. In [52], a Petri net diagnoser for discrete event systems modeled by finite state automata is proposed. The Petri net diagnoser is computed from a binary Petri net, called Petri net state observer, whose marking, after the observation of a trace, corresponds to the state estimate of the automaton model of the system. The Petri net state observer proposed in [52] has polynomial growth with the size of the automaton model of the system, and can be used to implement the observer of the plant. Moreover, a conversion method of the Petri net diagnoser into Ladder logic for the implementation on a programmable logic controller is also proposed in [52].

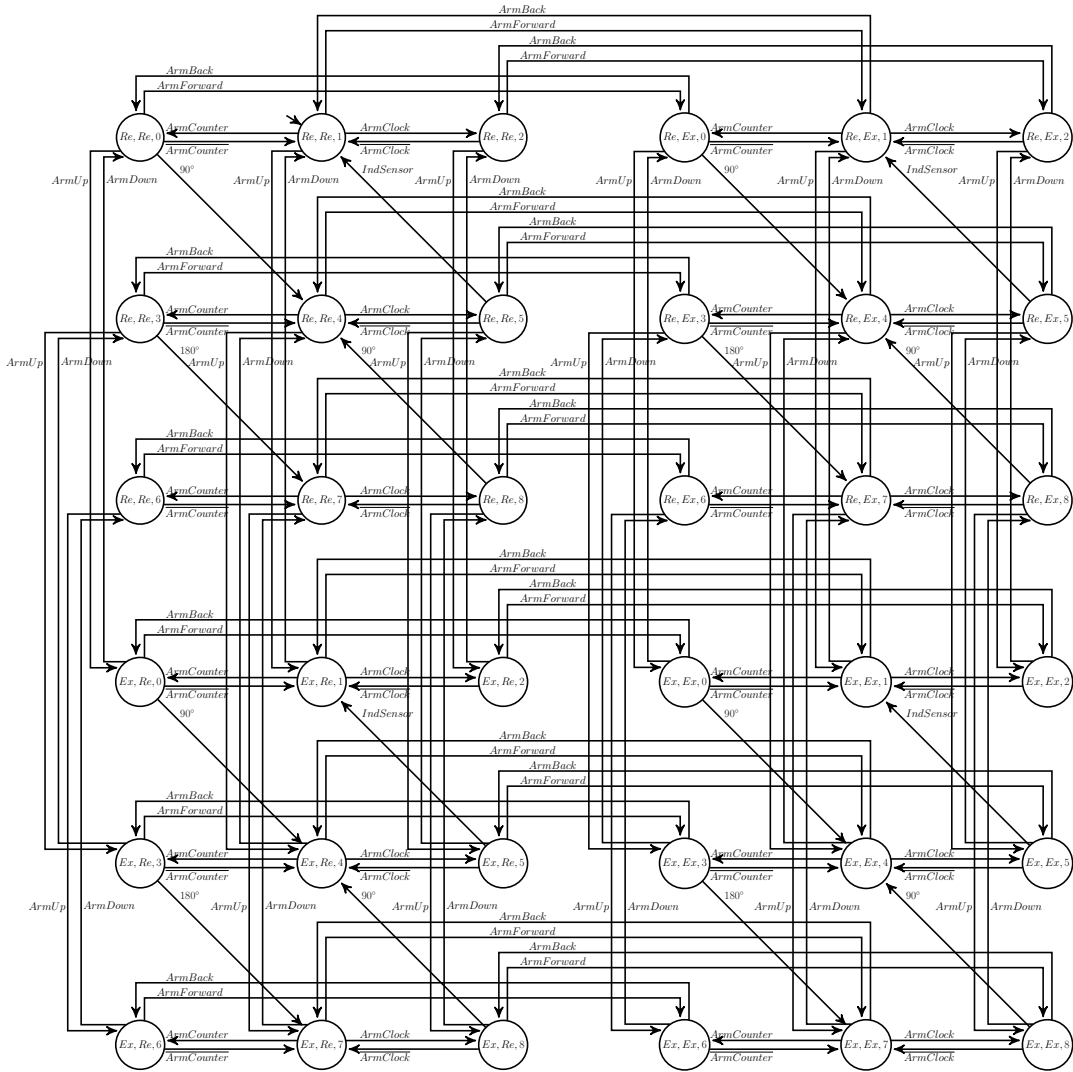


Figure 4.8: Parallel composition of automata depicted in Figures 4.3a, 4.3b, and 4.4, as described in Example 4.3.

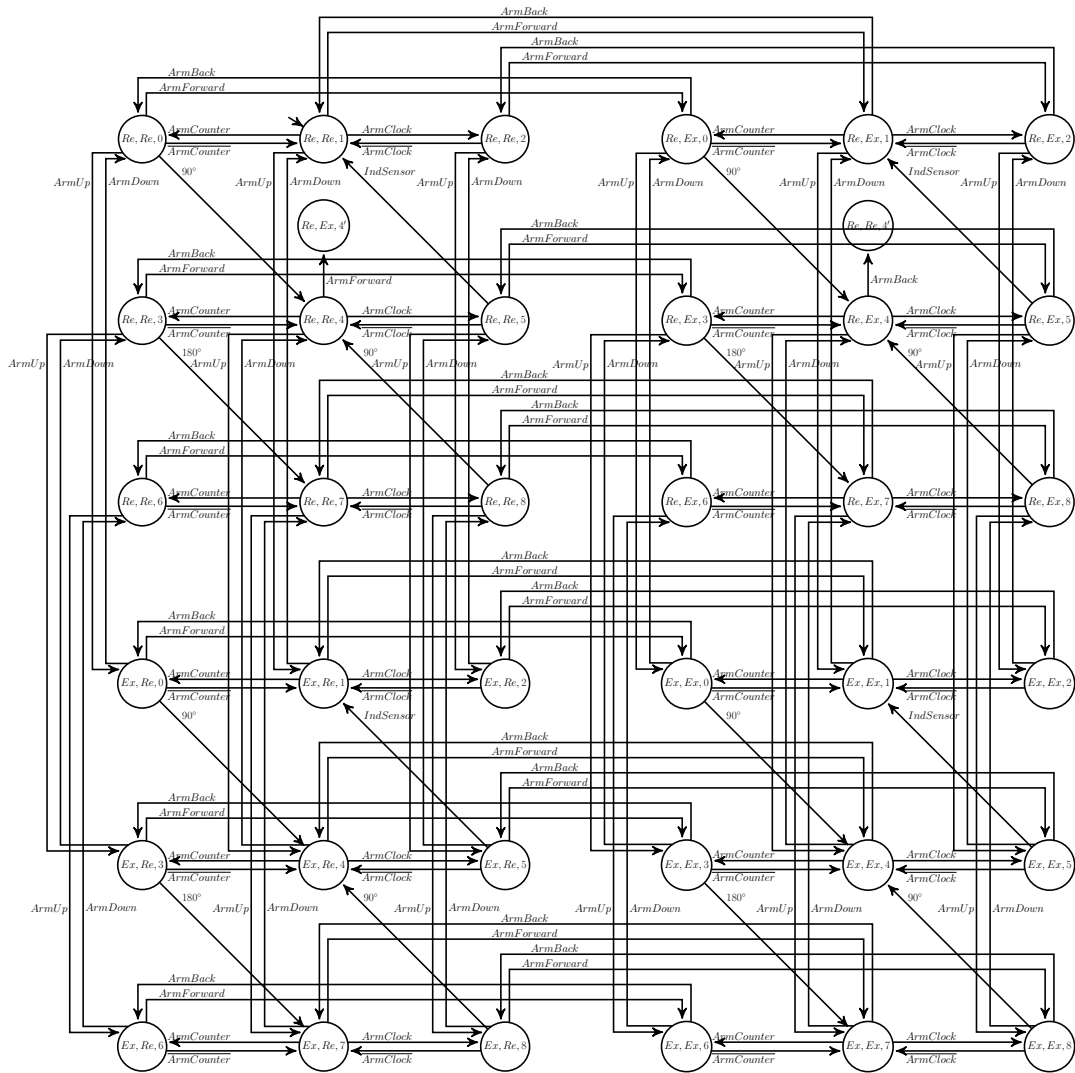


Figure 4.9: Automaton of a state splitting procedure on the automaton depicted in Figure 4.8, and the unsafe transitions *ArmForward* from state  $(Re, Re, 4)$ , and *ArmBack* from state  $(Re, Ex, 4)$ , as described in Example 4.3.

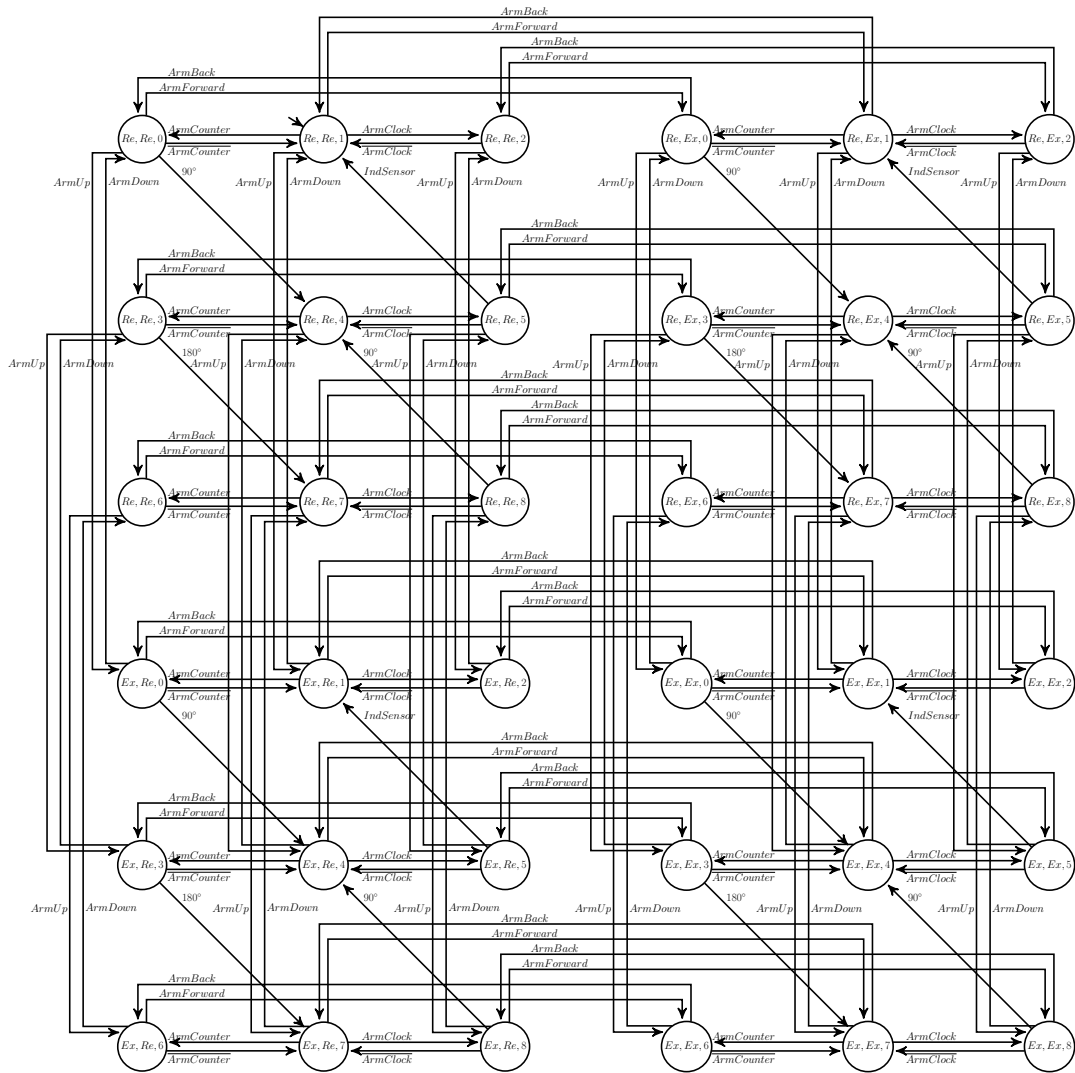


Figure 4.10: State transition diagram of safety device realization  $W$ , obtained by Algorithm 3.1 on the automaton of the robotic arm horizontal and vertical pneumatic actuators, and robotic arm rotation subsystem, as described in Example 4.3.

It is important to remark that the system is modular system, where the global system model is obtained from the parallel composition of several subsystems. Thus, in order to implement the observer of the plant, it is necessary to implement the observer of each subsystem [53], and then, the safety device. In this case, all events in this system are controllable and observable, and there is no event in common between two automata that model the subsystems. This condition allows the safety device to recognize the exact state of the system, as proved in [54].

Then, for this implementation, all automata in Figures 4.3 and 4.4 were separately converted to Ladder logic, and the safety device was implemented based on the combination of states of the automata. In addition, the disabling logic for the safety device was implemented by energizing a coil for each controllable event of the plant, *i.e.*, for each actuator that might be disabled. This coil opens a normally closed contact that must precede each output coil on the controller that represents this actuator. Moreover, the safety device coil is energized when the plant model reaches a state previous to an unsafe state or an unsafe transition. The coil to be energized in this state is the coil related to the disabled event. In the sequel, we present an example of conversion from automaton to Ladder logic, based on the subsystem of Example 4.1.

**Example 4.4** *In order to implement the subsystem described in Example 4.1 and its respective safety device, the automaton depicted in Figure 4.4 was converted to Ladder logic. The Ladder logic implemented is divided in 4 modules (Appendix A), initialization module (Figure A.1), events module (Figure A.2), conditions module (Figures A.3 and A.4), and dynamics module (Figures A.5, A.6, and A.7). Then, two output coils, referring to actuators ArmCounter and ArmClock, were implemented to be disabled after the opening of theirs respective contacts. Each of this contacts is commanded to open by reaching states P7 and P1 in the automaton of the plant, respectively. Thus, as shown in Figure A.8, when the system reaches state 1, the normally opened contact related to this state was closed, and the coil of the safety device related to event ArmClock is energized. Analogously, when the system reaches state 7, its respectively normally opened contact is closed, and the coil of the safety device related to event ArmCounter is energized. These energized coils of the safety device open two normally closed contacts that precede their respective output coils, as depicted in Figure A.9.*

The safety device was successfully implemented in the mechatronic system module of cube processing found in the Control and Automation Laboratory at the Federal University of Rio de Janeiro. In order to implement the safety device, we converted the plant model into Ladder logic and used its states to determine in

which states combination the device must disable the events. Then, in the next chapter, we present the conclusions and future work.

# Chapter 5

## Conclusions and future works

In this work, we address the problem of cyber-attacks that change the logic of the discrete event control, and the problem of incorrect implementation or design of the control logic. In addition, the modified, or incorrectly implemented, control is considered to be unknown before and after the attack.

Thus, we propose a safety device that prevents the system from reaching unsafe states, with a minimum level of interaction with any possible controller implemented in the system. We also present a formal method for obtaining this safety device from the automaton model of the plant. Then, we present a verification algorithm for the safety level of the plant. These safety levels identify the blocked behaviors of the system.

Finally, a safety device is implemented in a mechatronic plant located at the Control and Automation Laboratory (LCA) at the Federal University Rio de Janeiro (UFRJ).

For future works, we propose the analysis of the effects of modular implementation of the safety device on the safety level, which we expect the safety device to block behaviors that the non-modular implementation does not block. In addition, we intend to work on the expansion of the language of the safety device with a better decision-making on which events the device disables, based on the knowledge of the specification language. Moreover, the design of the safety device should consider a reduced knowledge of the specification language and the effects on the decision-making. Finally, we expect to search for better alternatives for devices in safety levels 3 and 4.



# Bibliography

- [1] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata theory, languages, and computation*. Pearson, 2014.
- [2] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. New York, Springer, 2008.
- [3] HOLLOWAY, L. E., KROGH, B. H., GIUA, A. “A Survey of Petri Net Methods for Controlled Discrete Event Systems”, *Discrete Event Dynamic Systems*, v. 7, n. 2, pp. 151–190, 1997.
- [4] DAVID, R. “Grafcet: a powerful tool for specification of logic controllers”, *IEEE Transactions on Control Systems Technology*, v. 3, n. 3, pp. 253–268, 1995.
- [5] ZHOU, M. C., TWISS, E. “Design of industrial automated systems via relay ladder logic programming and Petri nets”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 28, n. 1, pp. 137–150, 1998.
- [6] SUZUKI, I., MURATA, T. “A method for stepwise refinement and abstraction of Petri nets”, *Journal of Computer and System Sciences*, v. 27, n. 1, pp. 51 – 76, 1983.
- [7] ZHOU, M., DICESARE, F., DESROCHERS, A. A. “A hybrid methodology for synthesis of Petri net models for manufacturing systems”, *IEEE Transactions on Robotics and Automation*, v. 8, n. 3, pp. 350–361, 1992.
- [8] JENG, M. D., DICESARE, F. “A review of synthesis techniques for Petri nets with applications to automated manufacturing systems”, *IEEE Transactions on Systems, Man, and Cybernetics*, v. 23, n. 1, pp. 301–312, 1993.
- [9] RAMADGE, P. J., WONHAM, W. M. “Supervisory Control of a Class of Discrete Event Processes”, *SIAM Journal on Control and Optimization*, v. 25, n. 1, pp. 206–230, 1987.
- [10] IORDACHE, M., ANTSAKLIS, P. *Supervisory control of concurrent systems: a Petri net structural approach*. Springer Science & Business Media, 2007.

- [11] HU, H., ZHOU, M., LI, Z., et al. “An Optimization Approach to Improved Petri Net Controller Design for Automated Manufacturing Systems”, *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 3, pp. 772–782, 2013.
- [12] MOREIRA, M. V., BASILIO, J. C. “Bridging the Gap Between Design and Implementation of Discrete-Event Controllers”, *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 48–65, 2014.
- [13] SABOORI, A., HADJICOSTIS, C. N. “Notions of security and opacity in discrete event systems”. In: *2007 46th IEEE Conference on Decision and Control*, pp. 5056–5061, New Orleans, Louisiana, USA, 2007.
- [14] MAZARÉ, L. “Using unification for opacity properties”, *Proceedings of the 4th IFIP WG1*, v. 7, pp. 165–176, 2004.
- [15] TAKAI, S., OKA, Y. “A Formula for the Supremal Controllable and Opaque Sublanguage Arising in Supervisory Control”, *SICE Journal of Control, Measurement, and System Integration*, v. 1, n. 4, pp. 307–311, 2008.
- [16] DUBREIL, J., DARONDEAU, P., MARCHAND, H. “Opacity enforcing control synthesis”. In: *2008 9th International Workshop on Discrete Event Systems*, pp. 28–35, Goteborg, Sweden, 2008.
- [17] SABOORI, A., HADJICOSTIS, C. N. “Verification of initial-state opacity in security applications of DES”. In: *2008 9th International Workshop on Discrete Event Systems*, pp. 328–333, Goteborg, Sweden, 2008.
- [18] BEN-KALEFA, M., LIN, F. “Opaque superlanguages and sublanguages in discrete event systems”. In: *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 199–204, Shanghai, China, 2009.
- [19] CASSEZ, F., DUBREIL, J., MARCHAND, H. “Dynamic Observers for the Synthesis of Opaque Systems”. In: Liu, Z., Ravn, A. P. (Eds.), *Automated Technology for Verification and Analysis: 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, pp. 352–367, Berlin, Heidelberg, Springer Berlin Heidelberg, 2009.
- [20] BEN-KALEFA, M., LIN, F. “Supervisory control for opacity of discrete event systems”. In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1113–1119, Monticello, Illinois, USA, 2011.

- [21] CASSEZ, F., DUBREIL, J., MARCHAND, H. “Synthesis of opaque systems with static and dynamic masks”, *Formal Methods in System Design*, v. 40, n. 1, pp. 88–115, 2012.
- [22] WU, Y.-C., LAFORTUNE, S. “Comparative analysis of related notions of opacity in centralized and coordinated architectures”, *Discrete Event Dynamic Systems*, v. 23, n. 3, pp. 307–339, 2013.
- [23] JACOB, R., LESAGE, J.-J., FAURE, J.-M. “Overview of discrete event systems opacity: Models, validation, and quantification”, *Annual Reviews in Control*, v. 41, pp. 135 – 146, 2016.
- [24] 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, Xi’an, China, 2016.
- [25] LIMA, P. M. *SECURITY AGAINST NETWORK ATTACKS IN SUPERVISORY CONTROL SYSTEMS*. Tese de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 2017.
- [26] LIMA, P. M., ALVES, M. V. S., DE BRITO MOREIRA, M. V. “Security Against Network Attacks in Supervisory Control Systems”. In: *The 20th World Congress of the International Federation of Automatic Control (IFAC)*, Toulouse, France, 2017. accepted for publication.
- [27] CHEN, Y.-L., LIN, F. “Safety control of discrete event systems using finite state machines with parameters”. In: *American Control Conference, 2001. Proceedings of the 2001*, v. 2, pp. 975–980, Arlington, Virginia, USA, 2001. IEEE.
- [28] SHU, S., LIN, F. “Fault-tolerant control for safety of discrete-event systems”, *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 78–89, 2014.
- [29] 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, San Diego, California, USA, 2006.
- [30] 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.

- [31] FAWZI, H., TABUADA, P., DIGGAVI, S. “Secure state-estimation for dynamical systems under active adversaries”. In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 337–344, Monticello, Illinois, USA, 2011.
- [32] SHI, J., WAN, J., YAN, H., et al. “A survey of Cyber-Physical Systems”. In: *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, Nanjing, China, 2011.
- [33] BAHETI, R., GILL, H. “Cyber-physical systems”, *The impact of control technology*, v. 12, pp. 161–166, 2011.
- [34] 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.
- [35] 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.
- [36] 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, 2014.
- [37] 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. 19th IFAC World Congress.
- [38] LIN, F., WONHAM, W. “Decentralized supervisory control of discrete-event systems”, *Information Sciences*, v. 44, n. 3, pp. 199 – 224, 1988.
- [39] RAMADGE, P. J. G., WONHAM, W. M. “The control of discrete event systems”, *Proceedings of the IEEE*, v. 77, n. 1, pp. 81–98, 1989.
- [40] CHUNG, S. L., LAFORTUNE, S., LIN, F. “Limited lookahead policies in supervisory control of discrete event systems”, *IEEE Transactions on Automatic Control*, v. 37, n. 12, pp. 1921–1935, 1992.
- [41] BRANDIN, B. A., WONHAM, W. M. “Supervisory control of timed discrete-event systems”, *IEEE Transactions on Automatic Control*, v. 39, n. 2, pp. 329–342, 1994.
- [42] CURY, J. E. R., KROGH, B. H., NIINOMI, T. “Synthesis of supervisory controllers for hybrid systems based on approximating automata”, *IEEE Transactions on Automatic Control*, v. 43, n. 4, pp. 564–568, 1998.

- [43] DA CUNHA, A. E., CURY, J. E. “HIERARCHICALLY CONSISTENT CONTROLLED DISCRETE EVENT SYSTEMS”, *IFAC Proceedings Volumes*, v. 35, n. 1, pp. 253 – 258, 2002. 15th IFAC World Congress.
- [44] CURY, J. E., TORRICO, C. R., DA CUNHA, A. E. “Supervisory Control of Discrete Event Systems with Flexible Marking”, *European Journal of Control*, v. 10, n. 1, pp. 47 – 60, 2004.
- [45] 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.
- [46] YOO, T.-S., LAFORTUNE, S. “A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems”, *Discrete Event Dynamic Systems*, v. 12, n. 3, pp. 335–377, 2002.
- [47] USHIO, T., TAKAI, S. “Supervisory control of discrete event systems modeled by Mealy automata with nondeterministic output functions”. In: *2009 American Control Conference*, pp. 4260–4265, St. Louis, Missouri, USA, 2009.
- [48] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to Algorithms*. 2 ed. Cambridge, Massachusetts London, England, McGraw-Hill Book Company, 1990.
- [49] 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.
- [50] UZAM, M., JONES, A. H. “Discrete event control system design using automation Petri nets and their ladder diagram implementation”, *The International Journal of Advanced Manufacturing Technology*, v. 14, n. 10, pp. 716–728, 1998.
- [51] “Christiani Technical Institute for Vocation Training”. 2017. Disponível em: <<https://www.christiani.eu/>>.
- [52] 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.
- [53] CABRAL, F. G., MOREIRA, M. V., DIENE, O. “Online fault diagnosis of modular discrete-event systems”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 4450–4455, Osaka, Japan, 2015.

- [54] CABRAL, F. G., MOREIRA, M. V. “Online failure diagnosis of modular discrete-event systems”, *IEEE Transactions on Automatic Control*, 2017. Submitted for publication.

# Appendix A

## Ladder logic implementation example

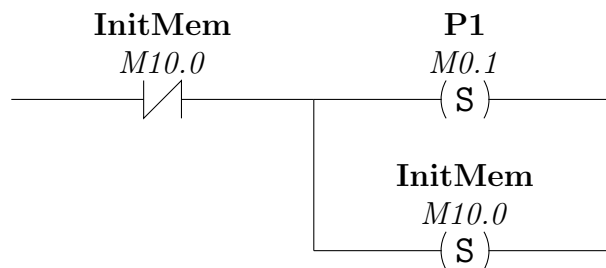


Figure A.1: Initialization module of Ladder logic of Example 4.4.

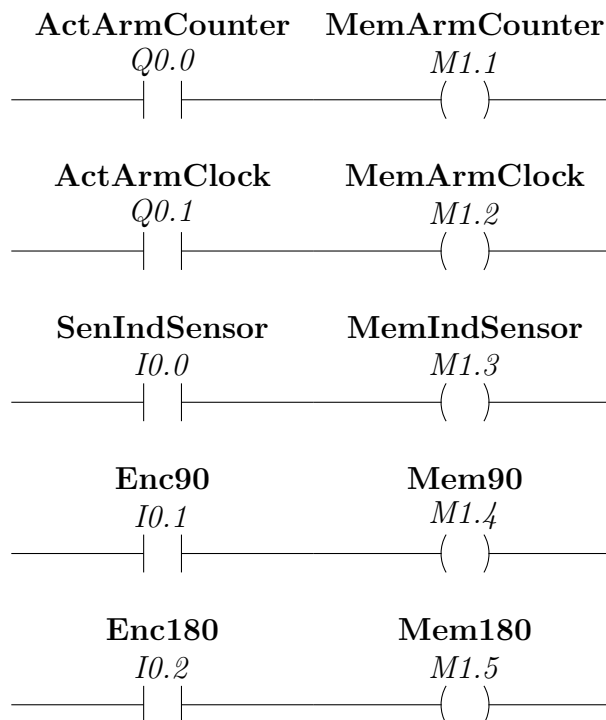


Figure A.2: Events module of Ladder logic of Example 4.4.

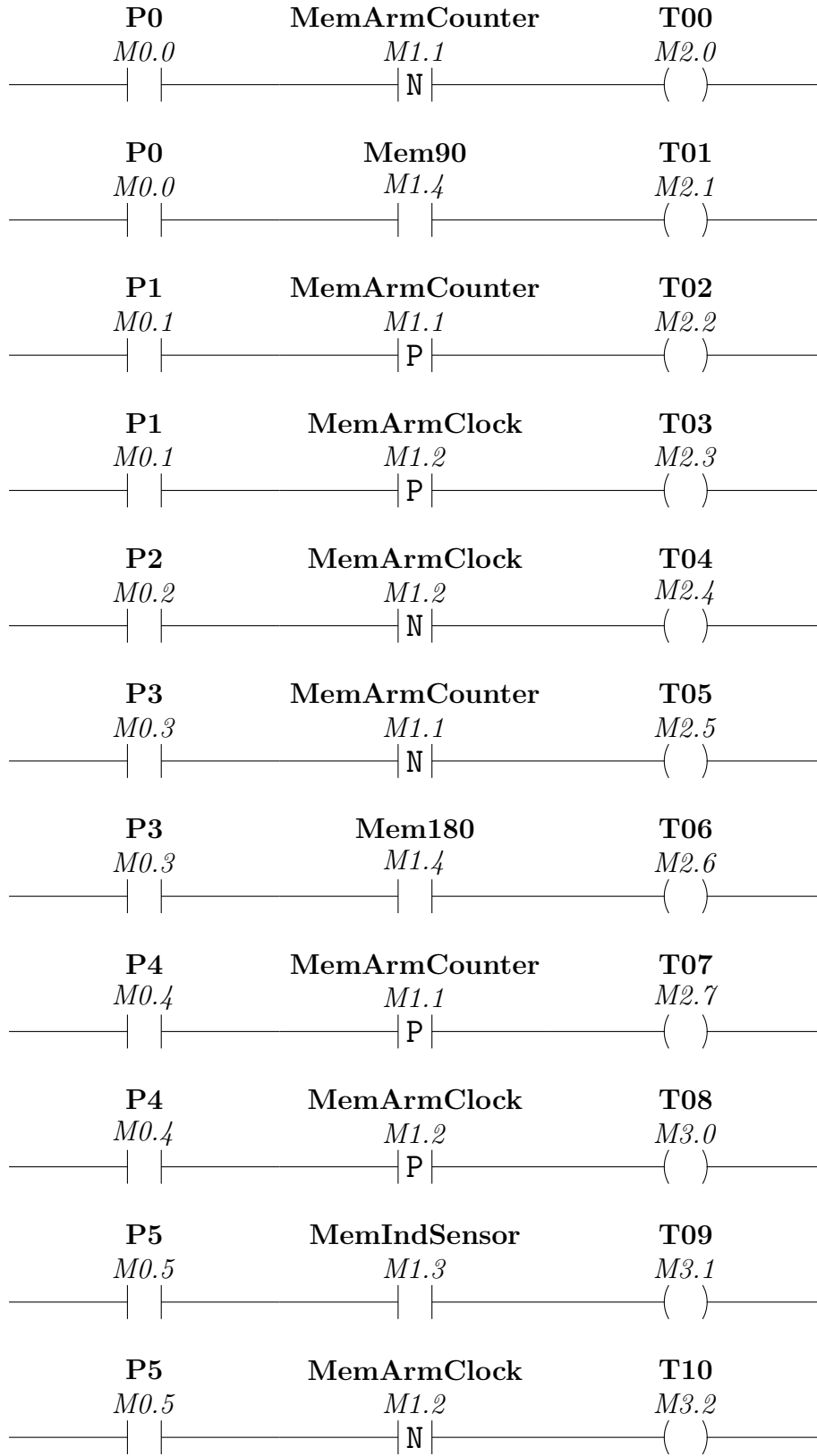


Figure A.3: Condition module of Ladder logic of Example 4.4.



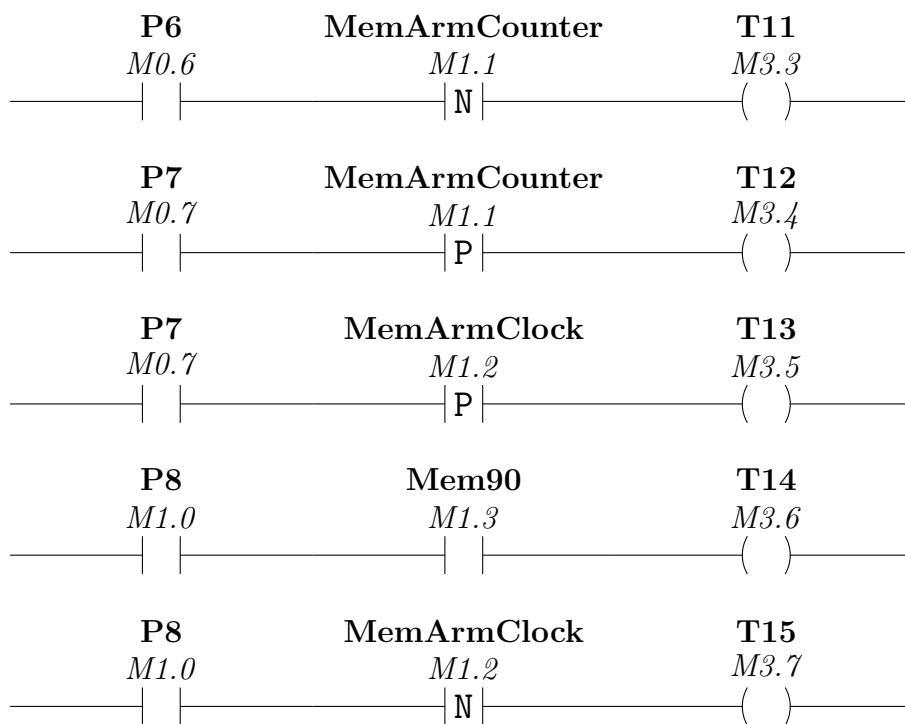


Figure A.4: Condition module of Ladder logic of Example 4.4.

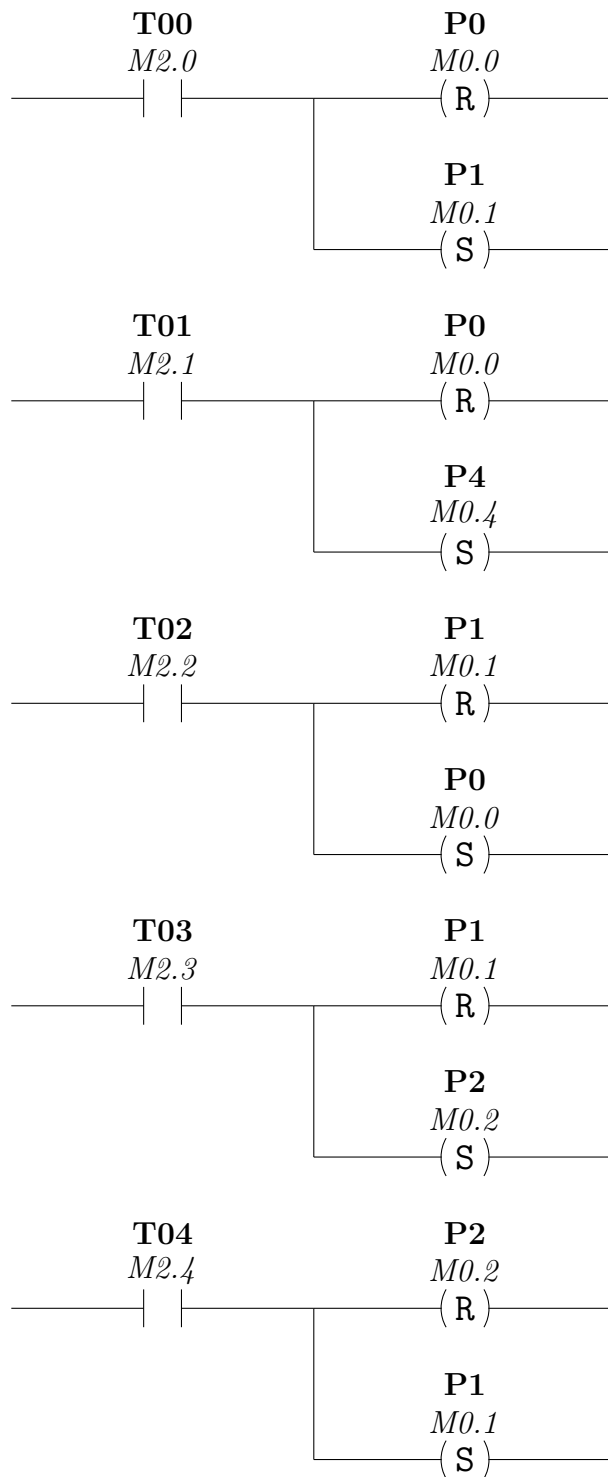


Figure A.5: Dynamics module of Ladder logic of Example 4.4.

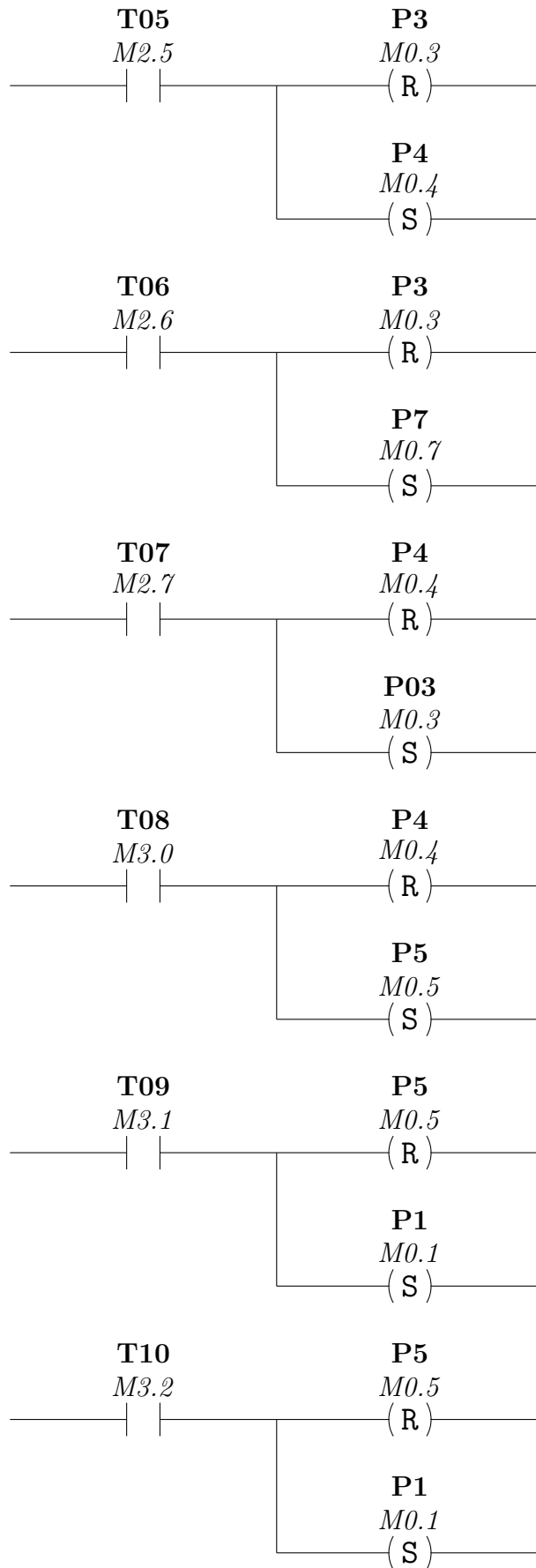


Figure A.6: Dynamics module of Ladder logic of Example 4.4.

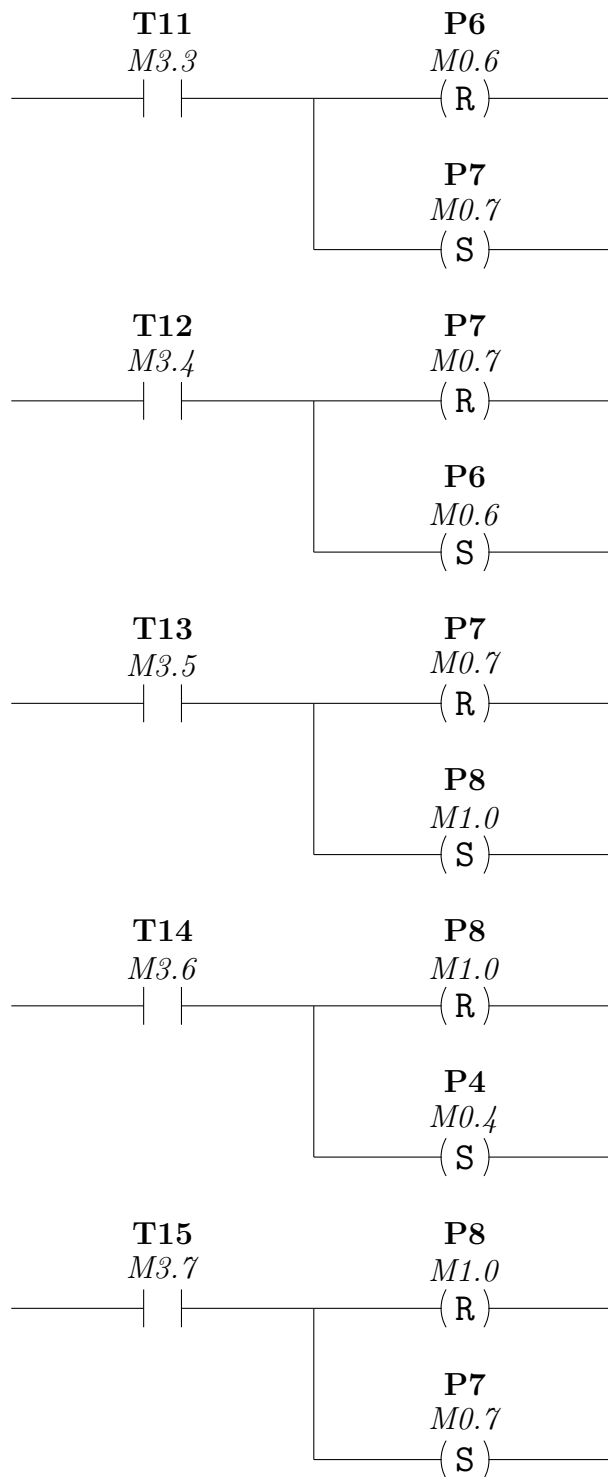


Figure A.7: Dynamics module of Ladder logic of Example 4.4.

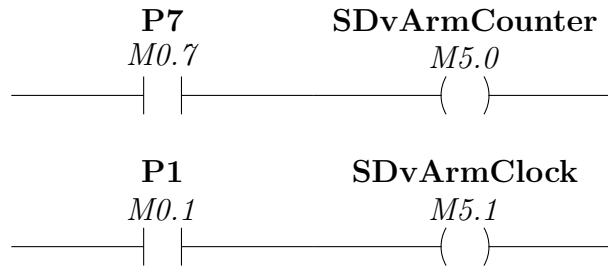


Figure A.8: Safety device implemented in Ladder logic, for Example 4.4.

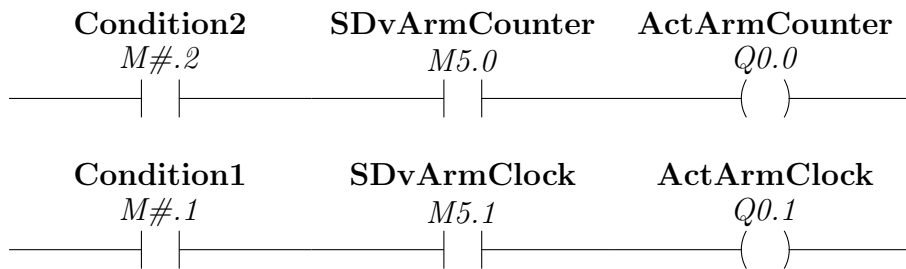


Figure A.9: Action module of Ladder logic with the safety device restrictions of Example 4.4.