SUPERVISORY CONTROL OF NETWORKED DISCRETE EVENT SYSTEMS WITH TIMING STRUCTURE

Marcos Vinícius Silva Alves

Tese de Doutorado 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 Doutor em Engenharia Elétrica.

Orientador: João Carlos dos Santos Basilio

Rio de Janeiro

Novembro de 2017

SUPERVISORY CONTROL OF NETWORKED DISCRETE EVENT SYSTEMS WITH TIMING STRUCTURE

Marcos Vinícius Silva Alves

TESE 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 DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. João Carlos dos Santos Basilio, Ph.D.

Prof. Lilian Kawakami Carvalho, D. Sc.

Prof. José Eduardo Ribeiro Cury, Docteur d'Etat

Prof. Antonio Eduardo Carrilho da Cunha, D. Eng.

Prof. Carlos Andrey Maia, D. Sc.

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2017

Alves, Marcos Vinícius Silva

Supervisory Control of Networked Discrete Event Systems with Timing Structure/Marcos Vinícius Silva Alves. – Rio de Janeiro: UFRJ/COPPE, 2017.

XIII, 124 p.: il.; 29, 7cm.

Orientador: João Carlos dos Santos Basilio

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 116 – 124.

1. discrete event systems. 2. supervisory control. 3. communication network. I. Basilio, João Carlos dos Santos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

He who desires but acts not,

breeds pestilence.

William Blake

Acknowledgments

First, I would like to thank God and my family for the protection and the support every time.

I thank my parents, Noêmia and Marcelo, for their love and affection for their sons, and their effort to give us pleasant lives. I have you to thank for my success.

I thank my brothers, Carlos, Charlles and Marcelo, which are true friends in all moments of my life.

I thank my girlfriend and future wife, Mayne, that motivated and helped me during the master and doctoral courses. Thank you for your love and endearments.

I deeply thank my advisor João Carlos Basilio for your patience and dedication to teach me.

I would like to thank Prof. Lilian Kawakami Carvalho for the support and the help during my postgraduate courses.

A special thank to my friend Carlos Eduardo for our nice discussions regarding our doctoral theses, photography, financial investment, etc.

I thank the friends of Laboratory of Control and Automation (LCA), specially, Prof. Marcos Vicente Moreira, Antonio Gonzalez, Gustavo Viana, Ingrid Antunes, Raphael Barcelos, Felipe Cabral, Alexandre Gomes, Juliano Freire, Públio Lima, Tiago França and Wesley Silveira.

I thank the National Council for Scientific and Technological Development (CNPq) for the financial support.

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

CONTROLE SUPERVISÓRIO DE SISTEMAS A EVENTOS DISCRETOS EM REDE COM ESTRUTURA TEMPORIZADA

Marcos Vinícius Silva Alves

Novembro/2017

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

Neste trabalho, considera-se o problema de controle supervisório de Sistemas a Eventos Discretos em rede com Estrutura Temporizada (SEDRET) sujeitos a atrasos limitados e perdas intermitentes de observação. Supõe-se que a comunicação entre a planta e o supervisor seja feita utilizando-se uma rede formada por vários canais de comunicação, de modo que atrasos de comunicação podem causar mudanças na ordem das observações. Será proposto um modelo não temporizado equivalente para SEDRET que caracteriza todas as consequências de atrasos e perdas de observação. Com esse objetivo, supõe-se o conhecimento prévio dos tempos mínimos de ativação das transições da planta e dos atrasos máximos na comunicação e considera-se, também, possíveis perdas de pacote. Um problema de controle supervisório em rede é formulado com base no modelo proposto e, em seguida, é apresentada uma condição necessária e suficiente para a existência de um supervisor em rede. Um método para projetar esses supervisores utilizando a propriedade da observabilidade relativa para aumentar a permissividade da linguagem obtida será apresentado. Adicionalmente, outro tópico de pesquisa abordado nesse trabalho é o conceito de observabilidade relativa. Neste contexto, algoritmos para a verificação da observabilidade relativa e para o cálculo de sublinguagens relativamente observáveis são propostos. Esses algoritmos são mais eficientes do que aqueles existentes na literatura.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SUPERVISORY CONTROL OF NETWORKED DISCRETE EVENT SYSTEMS WITH TIMING STRUCTURE

Marcos Vinícius Silva Alves

November/2017

Advisor: João Carlos dos Santos Basilio

Department: Electrical Engineering

In this work, we study the supervisory control problem of Networked Discrete Event Systems with Timing Structure (NDESWTS), that is subject to bounded communication delays and intermittent loss of observations. We assume that the communication between the plant and the supervisor is carried out through a network that can have several channels, so that, communication delays may change the order of the observations. We will propose an untimed equivalent model for NDESWTS that represents all possible implications of delays and loss of observations. For this matter, we assume *a priori* knowledge of the minimal transition activation time and the maximal communication delays, and also take into account possible packet losses. Based on this model, we formulate a networked supervisory control problem and present a necessary and sufficient condition for the existence of a networked supervisor. We also present a systematic way to design networked supervisors, where we use the property of relative observability in order to increase the achieved language permissiveness. In addition, another research topic addressed in this work is the concept of relative observability. In this concern, we propose new algorithms for the verification of relative observability and computation of relatively observable sublanguages, shown to be more efficient than the previous ones proposed in the literature.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades, the world has increasingly become integrated through communication networks. Industrial and domestic systems are being connected to communication networks for reasons, such as, the need for integration/cooperation of devices usually positioned far away from each other in a distributed system, and to improve the capacity of human operators remotely interact with devices in the system. In this context, new concepts, such as, Industry 4.0 [2–4] and Smart Cities [5, 6] have emerged in recent years, and are exploring, among other features, the interconnectivity between machines, devices, sensors, and people. In this sense, systems that integrate computing and communication capabilities to monitor and control physical processes, referred to as Cyber-Physical Systems (CPS), have been received considerable attention in the literature [7–11].

The quick growth in the use of communication networks has also increased the demand for new theoretical and practical approaches to deal with data transmission problems, such as, network delays, jitter, and losses [12, 13]. Furthermore, the growth of the system complexity creates the need for studying the behavior of a system in a higher level of abstraction with a view to understanding its whole behavior without introducing intractable growth of the method complexity. The behavior of a wide class of systems can be described, in a higher level of abstraction, by models in which the dynamic is represented as a function of the occurrence of asynchronous events. These event-driven dynamic systems, with discrete state spaces, are called Discrete Event Systems (DES).

In this work, we address the problem of supervisory control of DES, in which

the communication between the plant and the supervisor is carried out through a communication network that is subject to packet losses and non-negligible transmission delays. Such a supervisory control problem is referred in the literature to as supervisory control of networked discrete event systems [14].

Another research topic addressed in this work is the concept of relative observability. This new language property was proposed by CAI *et al.* [15] with a view to circumventing the deficiency of language observability regarding the non-existence of the supremal observable sublanguage. Although it has been introduced only recently, relative observability has already received considerable attention in the DES community [16–20]. As shown in [15], relative observability is closed under set union operation, as opposed to observability, which does not possess this property, and is also less conservative than normality [21]; although both the relative observability and normality share the set union closure property. As a consequence, the supremal relatively observable sublanguage always exists and is larger compared to the supremal normal sublanguage. The drawback of using relative observability property is the doubly exponential complexity of the algorithm proposed in [15] for the computation of the supremal relatively observable sublanguage, in comparison with the exponential complexity of the computation of the supremal normal sublanguage [22].

In the following section, we present an overview on the works that deal with supervisory control of networked discrete event systems. We also list some works that study other DES problems in the presence of communication delays and losses.

1.1 Discrete Event Systems Subject to Communication Delays and Losses

The supervisory control of DES was first proposed by RAMADGE and WONHAM [23, 24]. In this framework, the supervisor is used to modify the behavior of a DES in order to prevent the plant from executing some sequences of events that violate specifications we want to impose on the system behavior. The supervisor observes some, possibly all, of the events generated by the plant and, then, determines which events, in the current set of events that can be executed by the plant, are allowed to occur.

RAMADGE and WONHAM [23, 24] and most of the subsequent works in supervisory control of DES (*e.g.*, [25–29]) assume that sensors and actuators are not subject to failures and there is neither loss of data nor communication delay in the data transmission between the plant and the supervisor. That is, the supervisor observes the occurrence of an observable event immediately after it is executed by the plant, and a control action issued by the supervisor is instantaneously applied to the plant. However, these assumptions are reasonable only if the DES is tightly coupled with the supervisor, since, when the plant and supervisors are either far from each other or a more complex network is used to connect them, communication delays can cause delays in both the control action and the observation of events. In addition, sensor and low level controller/actuator malfunction can cause losses of event observation and inappropriate control action.

The problem of controlling input/output discrete event systems with communication delays has been studied in [30, 31]. In that framework, there are two communication channels: one from the plant to the supervisor, which is used to send the outputs of the plant (response events), and, another channel from the supervisor to the plant, which is used to control the inputs of the plant (command events). Both channels are considered subject to communication delays, and are modeled by first-in first-out queues (also referred to as FIFO queues), so that, the order of the transmitted sequence of events is not affected by delays. The set of command events (which are controllable) and the set of response events (which are uncontrollable) are disjoint, and the control specification is represented by a language over the set of response events. For a given specification language, BALEMI [31] provides necessary and sufficient conditions for the existence of a supervisor. However, an efficient manner to compute a supervisor is only provided in the restricted case of plants that generate the so-called memoryless languages.

PARK and CHO [32, 33] considered the supervisory control problem under full [32] and partial observation [33], in the presence of bounded delays in the transmission of observations and/or control actions. In this architecture, as in [30, 31], communication delays occur in the interaction between the plant and the supervisor, and it is assumed that communication delays do not modify either the order of the observations or the order in which the control actions are effectively applied

to the plant. In such approach, every controllable event is assumed to be disabled by default and it only occurs when is enabled by the supervisor, *i.e.*, its occurrence is not spontaneous. Then, due to the occurrence of delays in the transmission of observations or control actions, a finite number (bounded by some $D \in \mathbb{N}$) of uncontrollable events can be executed by the plant between the occurrence of a string in the plant and the application of the control action computed by the supervisor based on the observation of this string. In other words, each event executed by the plant is counted as one *step* and the maximum delay is equal to D *steps*. In addition, in order to prevent the accumulation of delay effects, PARK and CHO [32, 33] also assume that, when the supervisor sends a control action enabling controllable events, a new enabling control action can only be issued after observing either the occurrence of a string of uncontrollable events with the length equal to delay bound D or the occurrence of one of the enabled controllable events. Notice that, this additional assumption significantly restricts the behavior of the system and reduces the speed of the closed-loop system. Another restrictive assumption imposed in [33] is that each controllable event is also observable.

The approach proposed in [33] was extended in [34–36] to a decentralized version of the supervisory control problem, where the local supervisors do not communicate with each other. Similar to [33], it is assumed a bounded delay between the occurrence of a string in the plant and the effective application of the control action computed by each local supervisor based on this string observation. Additionally, the approach presented in [34] was extended in [37] to the decentralized supervisory control problem of Timed Discrete Event Systems (TDES) subject to communication delays, by considering the TDES framework proposed in [38].

The asynchronous implementation of a synchronous supervisor was studied in [39]. In this approach, the event occurrences are transmitted to the supervisor through a single communication channel and the control actions are sent to the plant through another communication channel. As in the Park and Cho's approach, it is assumed that both channels are subject to bounded transmission delays, which are measured by using the concept of *step*, and the channels are modeled by FIFO queues, which implies that there is no change in the order of event observations or in the order of applications of the control actions to the plant. A condition

for the existence of a supervisor is derived by assuming that the behavior achieved by the closed-loop system under delays must be equal to the behavior achieved by the closed-loop system without the occurrence of delays. However, this assumption is rather restrictive since it requires the same specification language be obtained, independently of the occurrence of communication delays.

More recently, LIN [14, 40] studied the supervisory control of Networked Discrete Event Systems, where the supervisor is connected to the plant by means of a communication network that can be shared with other devices in the systems, such as diagnosers and human interface devices. However, he assumes, as in the previous approaches, that all observations are transmitted to the supervisor through a single channel and also that the control actions are applied to the plant through a single channel. Both channels are modeled by FIFO queues subject to bounded communication delays and losses, and the concept of *step* is used to measure communication delays. Thus, as in the previous works, there is no change either in the order of event observations or in the order of applications of the control actions to the plant. A state-estimate-based supervisor is proposed to solve this problem and a necessary and sufficient condition for the existence of a state-estimate-based supervisor that is able to achieve a given specification language is provided. However, no result is presented for the cases when other supervisor may still exist despite state-estimate-based supervisors do not exist.

The supervisory control problem of networked DES proposed in [14, 40] is revisited in [41–47]. A decentralized supervisory control architecture without communication between the local supervisors is considered in [41] and a centralized supervisor is considered in [42–47]. In [41] and [42], the design of supervisors is carried out by assuming that there is neither losses of control actions nor losses of observations, and considering a lower and an upper bound on the language generated by the compensated system. The goal is to design a supervisor such that the behavior of the compensated system is both safe (*i.e.*, the considered language upper bound is contained within a given admissible language) and adequate (*i.e.*, the considered language lower bound contains a given required language). However, it can be checked that the language upper bound considered in [41, 42] may have some strings that cannot be executed by the compensated system even when communication delays

occur, and, consequently, the design approaches proposed there are too conservative and may produce less permissive supervisors. KOMENDA and LIN [43], WANG *et al.* [44], SHU and LIN [45] and SHU and LIN [46] address centralized versions of the networked supervisory control problem by assuming modular, robust, predictive and deterministic supervisors, respectively. However, an assumption, as strong as that made in [39], is imposed in [46] with a view to obtaining a deterministic closed-loop behavior, namely, to achieve the same language in both cases, with and without communication delays. In [47], the approach presented in [14] was extended to the supervisory control problem of TDES subject to communication delays and losses, by considering, as done in [37], the TDES framework proposed in [38].

Notice that all of the aforementioned approaches assume that there is no change in the order of observations by the supervisor, which is equivalent to assume the supervisory control architecture illustrated in Figure 1.1(a), where all observations are transmitted to the supervisor through a single communication channel modeled by a first-in first-out queue. However, a single communication channel is usually not enough, either because it may not have enough capacity to transmit all data, or if the system is distributed, sensors and supervisor are, in general, far away from each other [13, 48]. In this case, we need to consider a more general control architecture, as that illustrated in Figure 1.1(b), where the observations are transmitted to the supervisor through several communication channels.

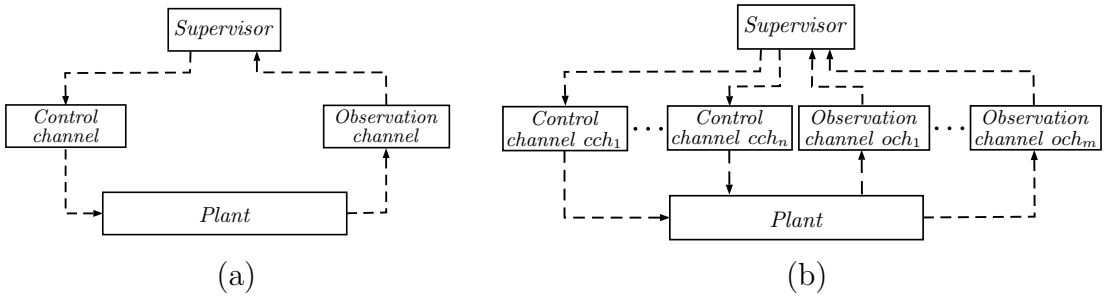


Figure 1.1: Networked supervisory control architectures with a single FIFO communication channel (a), and with several FIFO communication channels (b).

When multiple channels, subject to communication delays, are used to data transmission, it is necessary to take into account the possibility of the data received by the supervisor be in different order of its original order of occurrence [49]. With a view to softening this problem, we can insert time stamps in the communication

protocol. Thus, the protocol will be able to reorganize the events by using the time stamps. However, the supervisor must still deal with incomplete observations since, differently from the case of a single FIFO channel, the missing event observations may not correspond to the last event occurrence observation sent from the plant. In addition, in order to add a time information in the communication protocol, it is necessary to synchronize the clocks of the devices, which is not a simple task in distributed systems. Moreover, the synchronization process must be executed periodically on the whole plant, which increases the maintenance cost [50].

Another common feature of the approaches proposed in [14, 32–36, 39–45] and [46] is that the proposed models are based on the concept of *step* to measure the duration of communication delays; a communication delay bounded by n *steps* represents the case when n events can be generated by the plant while the data is being transmitted. Such an approach cannot represent plants with heterogeneous temporal behavior, *i.e.*, it cannot consider plants where different transitions are associated with different amount of time, so that, for example, when an event can precede two different sequences with the same number of events, a given communication delay may be such that is possible for an event to be observed after the occurrence/observation of one sequence but not after the occurrence/observation of the other sequence. Another way to approach this problem is by using the TDES model proposed in [38], as done in [37] and [47], where the time information is modeled by means of a new event called *tick* — each occurrence of the *tick* event represents that one time unit has elapsed. However, the use of the *tick* event can increase a great deal the state size of the corresponding untimed model when the plant has an heterogeneous temporal behavior.

Supervisory control problem of DES subject to loss of observations has been investigated by assuming permanent losses [51, 52] and intermittent losses [1, 14, 53]. ALVES *et al.* [53] presented necessary and sufficient conditions for the existence of robust supervisors that are able to cope with intermittent loss of observations under two different contexts: *(i)* the first context is associated with the concept of strong robust observability, and concerns the existence of a robust supervisor that is able to fully achieve a specification language, even if losses of observations occur, and; *(ii)* the second context is associated with the concept of weak robust observability,

and it concerns the existence of a robust supervisor that is able to fully achieve a specification language, when there is no loss of observation, and to restrict the behavior of the system to a sublanguage of the specification language in the presence of intermittent loss of observations.

The decentralized/distributed supervisory control problem of DES with delays in the communication between the local supervisors has been discussed in [54–58]. TRIPAKIS [54] considered a control architecture formed with a monolithic plant controlled by two local supervisors that communicate all of the observed event occurrences to each other. It is assumed in [54] that the communication between the supervisors is loss free and the channels are modeled by FIFO queues and subject to delays. As a consequence, only pairs of event occurrences that arrives to one of the supervisors, where one of them comes from the other supervisor, and the other event has been directly observed by the local supervisor, may have their observation order different from the original order of occurrence in the plant. Two types of delays are considered: *(i)* unbounded delay, and *(ii)* k-bounded delay (*i.e.* the delay upper bound is equal to k *steps*), where the same delay bound k is assumed for all communication channels of the system. TRIPAKIS [54] showed that, in the unbounded delay case, the verification of the existence of supervisors is an undecidable problem. HIRAISHI [55] proposed an automata formalism for communication with delay in decentralized control, and verifies that the decentralized supervisor problem is decidable in two special cases: *(i)* k-bounded delay communication, and *(ii)* when every cycle in the state transition diagram of the system contains an event that is observable by all controllers. SADID *et al.* [56] studied the problem of synthesizing robust synchronous communication protocols for decentralized supervisory control, assuming bounded communication delays that may change the order of event observations. As in [54] and [55], the same delay upper bound for all communication channels is assumed. Another important restriction imposed in [56] is that their approach restricts the problem to those systems whose automaton models have no loops of communication events (events that are subject to communication delays) in the original system.

More recently, ZHANG *et al.* [57] investigate a delay-robustness property in distributed supervisory control of DES with unknown unbounded communication

delays. To this end, they introduce 2-state automaton model for the inter-agent communication channels, which makes possible both the computation of the overall system behavior, and to verify if a distributed controller is delay-robust. Finally, ZHANG *et al.* [58] extend the approach proposed in [57] by adopting TDES, in which communication delays are measured by using the *tick* event; a procedure for verification of timed delay-robustness is presented, and, when the delay-robust property does not hold, a bounded delay-robustness is considered with a view to computing the maximal delay bound (measured by number of ticks) for all communication channels for which the system remains bounded delay-robust.

In the DES literature, communication delays have also been considered in failure diagnosis [49, 50, 59] and failure prognosis [60]. In [49], the problem of codiagnosability of networked DES subject to communication delays is addressed by assuming that the occurrences of events are communicated to each local diagnoser through a set of channels modeled by FIFO queues, and, consequently, communication delays can cause changes in the order of observations received by the local diagnosers. NUNES *et al.* [49] use the concept of step to measure the duration of communication delays, and assume that each communication channel is subject to delays with a different delay upper bound. In addition, losses of observations are taken into account in the problems of failure diagnosis in [61–64] and failure prognosis in [65].

Figure 1.2 shows the main differences between our approach and others previously presented in the literature regarding the location of the communication channels subject to delays, the number of communication channels/effect of communication delays, and the formalism used to measure communication delays.

1.2 Contributions of the Thesis

In this doctoral thesis, we formulate the supervisory control problem of networked discrete event systems, by assuming the supervisory control architecture depicted in Figure 1.1(b), where the communication between the plant and the supervisor is carried out through a network that can have several channels, so that, communication delays can cause changes in the order of the observations. We also consider intermittent loss of observations, but not delay and loss in the transmission of control

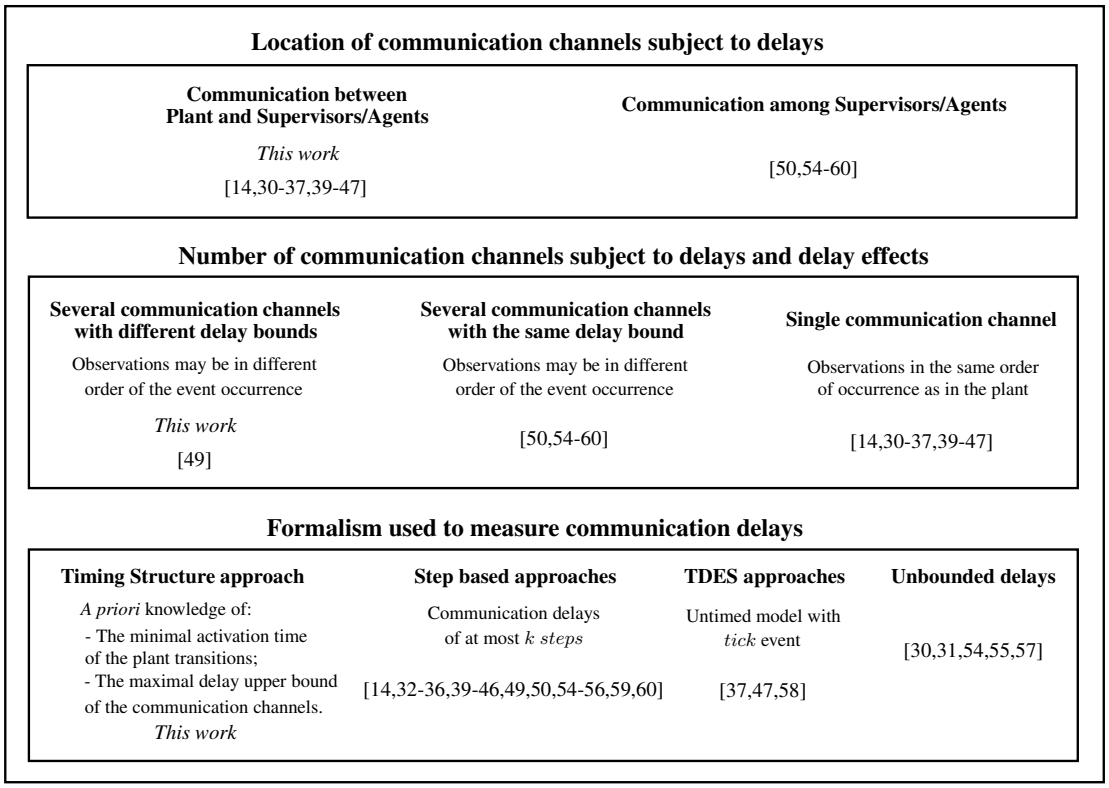


Figure 1.2: Comparison among different networked DES regarding the location of the communication channels subject to delays, the number of communication channels/effect of communication delays, and the formalism used to measure communication delays.

actions from the supervisor to the plant.

We propose a model, based on [49], for a class of networked DES, to be referred to as Networked Discrete Event Systems with Timing Structure (NDESWTS), for which we assume *a priori* knowledge of *(i)* the minimal activation times of the plant transitions, *(ii)* the maximal communication delay, possibly different, of each channel in the communication network, and *(iii)* the set of events that are subject to loss of observations. In order to avoid using the concept of *step* or the TDES model proposed in [38], we model the consequences of communication delays of the observations received by the supervisor by directly applying the time information. To this end, we define an extended language that represents all possible implications of communication delays and loss in the observations received by the supervisor. In addition, we present an algorithm for the construction of an untimed deterministic finite-state automaton that generates the extended language associated with a given NDESWTS. This model is general enough to allow its application to other research

topics in NDESWTS, such as, decentralized diagnosis of networked DES [66, 67].

Based on the model for NDESWTS proposed here, we formulate a networked supervisory control problem, and present a necessary and sufficient condition for the existence of a networked supervisor. We also present a systematic way to design networked supervisors, where we use the property of relative observability in order to increase the achievable language permissiveness.

In order to circumvent the drawbacks regarding the doubly exponential complexity of the algorithm proposed by [15], and, thus, to apply the relative observability to increase the language permissiveness of the supervisory control for NDESWTS, we also propose three new algorithms for the application of relative observability. The first algorithm, which has polynomial time complexity, can be used to verify if a regular language is relatively observable. The second algorithm computes the supremal relatively observable sublanguage of a regular language. This algorithm has exponential complexity, and is, therefore, considerably more efficient than that proposed in [15]. The key to the success of this algorithm is a new property on relative observability presented in this work, which ensures that for any ambient language, there exists an equivalent reduced ambient language that is a subset of the given language. It is worth remarking that the computational complexity of the algorithm proposed here for the computation of the supremal relatively observable sublanguage becomes polynomial when the automaton that marks the admissible language is state partition. The third algorithm, which is based on the second one, can be used to compute a controllable and observable sublanguage of a regular language by using the concept of relative observability.

1.3 Thesis Organization

The remainder of this doctoral thesis is structured as follows. We present some fundamentals on DES and supervisory control theories in Chapter 2. In Chapter 3, we introduce a new property of relative observability (Section 3.1) and propose three new algorithms: an algorithm for the verification of relative observability (Section 3.2), an algorithm for the computation of the supremal relatively observable sublanguage (Section 3.3), and an algorithm for the computation of a controllable

and observable sublanguage of a regular language by using the concept of relative observability (Section 3.4). In Chapter 4, we address the supervisory control problem of NDESWTS, and, to this end, we formally define NDESWTS and propose an equivalent model for this class of discrete event systems in Sections 4.1 and 4.2. In the sequence, we formulate and solve the supervisory control problem of NDESWTS (Section 4.3). Finally, in Chapter 5, we present the conclusions and outline future research directions.

Chapter 2

Fundamentals of Discrete Event Systems and Supervisory Control

In this chapter, we present a brief review on Discrete Event Systems (DES) theory and Supervisory Control of DES, with a view to introducing the main concepts on DES for novice readers. The theory presented here is based on CASSANDRAS and LAFORTUNE [68], except for Section 2.4 [62] and Subsection 2.5.3 [15].

The organization of this chapter is as follows. The definition of discrete event system is presented in Section 2.1. The languages associated with DES are formally defined together with some of their operations in Section 2.2. An overview on the automata theory is presented in Section 2.3. A model for DES subject to intermittent loss of observations is presented in Section 2.4. Some concepts of supervisory control theory are presented in Section 2.5; in Subsections 2.5.1 and 2.5.2 we address the problem of supervisory control under partial controllability and partial observability, respectively, and, in Subsection 2.5.3, we review the definition of relative observability.

2.1 Discrete Event Models

A system can be defined as a set of parts that act jointly to perform a task that cannot be performed by any of these parts separately. A system is usually formed by physical parts as, for example, the machines in a manufacturing system, or can also be associated with abstract phenomena such as financial transactions by using

digital currency or digital payment systems (*e.g.*, PayPal, Bitcoin, Ethereum).

When a system is studied in engineering, a model is usually created to characterize the behavior of the real system in a given level of abstraction. In order to construct a model, two sets of measurable variables are frequently chosen to describe the behavior of the system. The first set comprises the input variables, which can be directly manipulated. The second set comprises the output variables, which, although can be measured, they cannot be directly manipulated.

A system is said to be static if the values of its output variables, at every time instant $t = t_0$, are determined solely by the values of its input variables at $t = t_0$ and, consequently, they do not depend on the previous input values, *i.e.*, for $t < t_0$. On the other hand, in a dynamic system, the values of its output variables, at all time t_0, depend on the previous values of its input variables. In order to deal with this fact, the concept of state is introduced, being used to store the information about the previous behavior of the dynamic system that is needed to determine its current output values. Namely, the state of a dynamic system at a given time t_0 is the information required at t_0 such that the outputs of the system are uniquely determined, for all $t \geq t_0$, from the knowledge of the state at t_0 and the input values during time interval $[t_0, t]$. The state space of a dynamic system, usually denoted by X, is the set of all possible values that the state can assume.

In accordance with the features of the system and the task purposes, the model used to represent the system can have continuous or discrete state variables, where, in the last case, the state variable can only assume values belonging to a discrete set. In addition, the model can be continuous-time, *i.e.*, the state variables of the systems are continuous-time functions, or discrete-time, *i.e.*, the sate variables of the systems are discrete-time functions and, thus, functions defined only at discrete instants in time.

A distinct class of systems with discrete state space is that formed by systems in which the state transitions are associated with events that occur asynchronously at discrete points over time. An event can be seen as an instantaneous occurrence that causes a transition from one state value to another. They can be associated with a specific action, such as, somebody activates a machine, or spontaneous occurrences, *e.g.*, a shutdown of a computer due to an unknown agent, or, still, the result of sensor

measurement. Such systems, whose state transitions are driven by the occurrence of events, are called Discrete Event Systems, being formally defined as follows.

Definition 2.1 (discrete event system [68]) *A discrete event system (DES) is a dynamic system with discrete state space, where the state evolution is dictated by the occurrence of asynchronous discrete events over time.*

Because of the asynchronous nature of the state transitions of discrete event systems, their dynamics are, frequently, modeled as a function of sequences of events. For this reason, we review, in the next section, the concept of language.

2.2 Languages

The logical behavior of a DES can be described by the sequence of states visited by the system, and the sequence of events (strings) that induced these state transitions. The set of all strings that can be generated by a system characterizes its language, which is formally defined as follows.

Definition 2.2 (language) *A language defined over a finite event set Σ (alphabet) is a set of finite-length strings formed with events in Σ.*

The concatenation is the operation of linking events and/or strings together in a series, and is the basic operation for the creation of strings and, consequently, languages. For example, string abb formed from events in $\Sigma = \{a, b\}$, can be formed by the concatenation of string ab with event b, and string ab is obtained from the concatenation of events a and b. The empty string, denoted by ε, is the identity element of concatenation, *i.e.*, $s\varepsilon = \varepsilon s = s$, for every string s.

The Kleene closure of an event set Σ, denoted by Σ^*, is the set of all finite-length strings formed from events in Σ, including the empty string ε. All languages defined over Σ are, therefore, subsets of Σ^*. In particular, $\emptyset$, Σ and Σ^* are languages defined over Σ.

Example 2.1 *Consider the set of events $\Sigma = \{a, b\}$. Then, $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}$, and $L_1 = \emptyset$, $L_2 = \{\varepsilon, a, aa, aaa, \ldots\}$, $L_3 = \{aa, ab, ba, bb\}$ and $L_4 = \Sigma^*$ are languages defined over Σ.*

Before presenting the operations on languages, we need to define some terminology about strings. Let us consider a string s arbitrarily partitioned as $s = tuv$, where $t, u, v \in \Sigma^*$, being, therefore, substrings of s, in particular, substring t is a prefix of s, whereas substring v is a suffix of s. Notice that, ε and s are both substrings, prefixes and suffixes of s. Additionally, $\|s\|$ denotes the length of string s. The length of the empty string ε is equal to zero.

2.2.1 Operations on Languages

Since languages are sets whose elements are strings, the usual set operations, such as union, intersection, difference and complement with respect to Σ^*, can be also applied to languages, as we illustrate in the following example.

Example 2.2 *Consider event set* $\Sigma = \{a, b, c\}$ *and languages* $K_1 = \{\varepsilon, a, ab, abc\}$ *and* $K_2 = \{a, bc\}$ *defined over* Σ. *Then, the union, the intersection and the difference between* K_1 *and* K_2, *and the complement of* K_2 *with respect to* Σ^* *are, respectively,*

$$\begin{aligned} K_1 \cup K_2 &= \{\varepsilon, a, ab, bc, abc\}, \\ K_1 \cap K_2 &= \{a\}, \\ K_1 \backslash K_2 &= \{\varepsilon, ab, abc\}, \\ K_2^C &= \{\varepsilon, b, c, aa, ab, ac, ba, bb, ca, cb, cc, aaa, ...\}. \end{aligned}$$

In addition to set operations, the following operations on languages are frequently used in DES theory.

Concatenation

Let K_1 and K_2 denote languages defined over a set of events Σ. The concatenation of K_1 with K_2 is the language

$$K_1 K_2 := \{s \in \Sigma^* : (\exists (s_1, s_2) \in K_1 \times K_2)[s = s_1 s_2]\}.$$

That is, a string belongs to $K_1 K_2$ if it can be formed by the concatenation of a string in K_1 with a string in K_2.

Prefix-closure

Let $K \subseteq \Sigma^*$. The prefix-closure of K is the language

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

In words, $\overline{K}$ is the language formed by all prefixes of all strings in K. Notice that $K \subseteq \overline{K}$. Additionally, a language K is said to be prefix-closed if $K = \overline{K}$.

Kleene-closure

Let $K \subseteq \Sigma^*$. The Kleene-closure of K is the language

$$K^* = \{\varepsilon\} \cup K \cup KK \cup KKK \cup \cdots$$

That is, the Kleene-closure of K is the set of all finite-length strings created by the concatenation between strings in K and also includes the empty string ε.

Example 2.3 *Consider the event set Σ and the languages K_1 and K_2 presented in Example 2.2. Firstly, notice that $K_1 = \overline{K_1}$, and, thus, K_1 is prefix-closed. The concatenation of K_1 and K_2, the prefix-closure of K_2 and the Kleene-closure of K_2 are, respectively,*

$$\begin{aligned} K_1K_2 &= \{a, aa, bc, aba, abc, abca, abbc, abcbc\}, \\ \overline{K}_2 &= \{\varepsilon, a, b, bc\}, \\ K_2^* &= \{\varepsilon, a, aa, bc, aaa, abc, bca, aaaa, aabc, abca, bcaa, bcbc, \ldots\}. \end{aligned}$$

Let Σ_s and Σ_l denote two sets of events such that $\Sigma_s \subset \Sigma_l$. The following operations are used, respectively, to remove and to add some events from the strings that belong to a language.

Natural Projection [23]

The natural projection P of a string $s \in \Sigma_l^*$ is the mapping:

$$\begin{aligned} P : \Sigma_l^* &\longrightarrow \Sigma_s^* \\ s &\longmapsto P(s) \end{aligned}$$

defined as follows:

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

The natural projection operation is extended to a language $K \subseteq \Sigma_l^*$ by applying P to all string in K, that is,

$$P(K) := \{t \in \Sigma_s^* : (\exists s \in K)[P(s) = t]\}.$$

Throughout the text the terms projection and natural projection will be used with no distinction.

Inverse Projection

The inverse projection P^{-1} of a string $t \in \Sigma_s^*$ is the mapping:

$$\begin{aligned} P^{-1} : \Sigma_s^* &\longrightarrow 2^{\Sigma_l^*} \\ t &\longmapsto P^{-1}(t) \end{aligned}$$

defined as follows:

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

In addition, the inverse projection P^{-1} of a language $K_s \subseteq \Sigma_s^*$ is the union of the inverse projections of all strings in K_s, that is:

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

Example 2.4 *Consider event sets* $\Sigma_l = \{a, b, c\}$ *and* $\Sigma_s = \{a\}$ *and the projection* $P : \Sigma_l^* \to \Sigma_s^*$. *The projection of language* $K_l = \{a, b, c, aab, aba, aca, aaabc\}$, *defined over* Σ_l, *is* $P(K_l) = \{\varepsilon, a, aa, aaa\}$. *On the other hand, the inverse projection of language* $K_s = \{a\}\{a\}^*$, *defined over* Σ_s, *is* $P^{-1}(K_s) = \{b, c\}^*\{a\}\{a, b, c\}^*$.

We can use languages to describe the behavior of a DES and apply the operations

described above to manipulate the discrete event model. However, it could be hard to directly work with them. For example, language $K_s = \{a\}\{a\}^*$ is the set of all finite strings in $\Sigma_s^* = \{a\}^*$ except for the empty string ε. Notice that it is difficult to list all strings in K_s and, thus, the manipulation of this language is quite difficult. In order to circumvent this problem, we can use some formalisms to represent the languages of a DES in an appropriate manner, *e.g.*, automata [68, 69] and Petri nets [70, 71]. In the following section, we will present some fundamentals on automata theory, that we intend to use throughout this work.

2.3 Automata

Automata are devices that can be used to represent an important class of languages according to well-defined rules. This formalism represents languages by using a state transition structure, that is, by specifying which events can occur at each state of the system [68].

Definition 2.3 (deterministic automaton [69]) *A deterministic automaton G is a six-tuple*

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

where X is the set of states, Σ is the set of events associated with G, $f : X \times \Sigma \to X$ is the partial transition function, such that $f(x, \sigma) = y$ means that there is a transition labeled by event σ from state x to state y, $\Gamma : X \to 2^{\Sigma}$ is the set of active events [1]*, that is, for all $x \in X$, $\Gamma(x) = \{\sigma \in \Sigma : f(x, \sigma)!\}$, where ! means that $f(x, \sigma)$ is defined, i.e., $\exists y \in X : f(x, \sigma) = y$, x_0 is the initial state, and X_m is the set of marked states.*

The dynamic behavior of an automaton G occurs as follows: at the beginning, the automaton is in the initial state x_0; when some event $\sigma \in \Gamma(x_0)$ occurs, G goes from state x_0 to state $x = f(x_0, \sigma)$ and remains there until some event in $\Gamma(x)$ occurs, in which case, the automaton either moves to another state or can even remain in x (self-loop). After that, this process repeats according to the definition of transition function f.

[1] Strictly speaking, the use of Γ in the definition of G is redundant. However, we will keep it throughout the text because it sometimes makes the description of new automata easier.

Definition 2.3 do not impose that the set of states X must be finite. When X is finite, the automaton is said to be a finite-state automaton. Henceforth, the term finite-state deterministic automaton will be referred simply to as automaton.

A graph representation is a simple form of depicting an automaton. The state transition diagram of an automaton is a graph where each vertice is a state of the automaton, and the edges of the graph are associated with the transitions of the automaton. In addition, the initial state is identified by an arrow, and the marked states are highlighted by double circles. From the state transition diagram of an automaton, it is possible to infer some information about its elements, as shown in following example.

Example 2.5 *Consider the state transition diagram of automaton G_1 depicted in Figure 2.1. From this picture, it can be concluded that the set of states of G_1 is $X = \{0, 1\}$, the initial state is $x_0 = 0$, the set of marked states is $X_m = \{0\}$, the transition function is defined for the following pairs belonging to $X \times \Sigma$: $f(0, \alpha) = 0$, $f(0, \sigma) = f(0, \beta) = 1$, $f(1, \alpha) = 1$ and $f(1, \sigma) = 0$, and the sets of active events for each state of G_1 are $\Gamma(0) = \{\alpha, \beta, \sigma\}$ and $\Gamma(1) = \{\alpha, \sigma\}$. Moreover, from Figure 2.1, $\Sigma \supseteq \{\alpha, \beta, \sigma\}$. Notice that Σ can have other events that do not directly affect the dynamic of G_1, but they can affect the results obtained when G_1 is applied to the composition operations that we will present further in this chapter.*

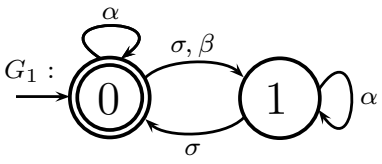


Figure 2.1: Example of state transition diagram of an automanton.

Each directed path that can be followed along the state transition diagram of an automaton can be associated with a string obtained by the concatenation of the event labels of the transitions composing this path. Frequently, two languages are associated with automaton behavior, as follows: *(i)* the language generated by automaton G, denoted as $L(G)$, that is the set of all strings associated with each directed path starting at the initial state, and *(ii)* the language marked by automaton G, denoted by $L_m(G)$, that is the subset of $L(G)$ composed by all strings

that reach a marked state. If a language can be marked by a finite-state automaton, it is said to be a regular language.

In order to formally define the languages generated and marked by an automaton, let us extend the transition function to the domain $X \times \Sigma^*$, as follows: *(i)* $f(x, \varepsilon) := x$; *(ii)* $f(x, s\sigma) := f(f(x, s), \sigma)$, for all $x \in X$, $s \in \Sigma^*$ and $\sigma \in \Sigma$ such that $f(x, s) = y$ and $f(y, \sigma)$ are both defined.

Definition 2.4 (generated and marked languages) *The language generated by automaton G is defined as $L(G) := \{s \in \Sigma^* : f(x_0, s)!\}$. The language marked by G is defined as $L_m(G) := \{s \in L(G) : f(x_0, s) \in X_m\}$.*

Notice that, in accordance with Definition 2.4, $\varepsilon \in L(G)$, for all G with a nonempty set of states. In addition, $L(G)$ is prefix-closed and $L_m(G) \subseteq L(G)$. Consequently, $\overline{L_m(G)} \subseteq L(G)$. Then, there are two possibilities: *(i)* $\overline{L_m(G)} = L(G)$, or *(ii)* $L_m(G)$ is a proper subset of $L(G)$, *i.e.*, $L_m(G) \subset L(G)$. When *(i)* $\overline{L_m(G)} = L(G)$, G is said to be nonblocking. On the other hand, when *(ii)* $L_m(G) \subset L(G)$, automaton G is said to be blocking.

2.3.1 Operations on Automata

In this subsection, we review some operations on automata. Initially, we present unary operations that can be used to modify the state transition diagram of an automaton. In the sequence, we show composition operations that can be applied to combine two or more automata.

Accessible Part

A state $x \in X$ of an automaton G is accessible if there exists a string $s \in \Sigma^*$ such that $f(x_0, s) = x$. Otherwise, x is a non-accessible state.

The Ac operation removes all non-accessible states of an automaton G, and is formally defined as follows:

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

where $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$, $f_{ac} = f|_{X_{ac} \times \Sigma \to X_{ac}}$, $\Gamma_{ac} = \Gamma|_{X_{ac} \to X_{ac}}$, and $X_{m,ac} = X_m \cap X_{ac}$. The notation $f|_{X_{ac} \times \Sigma \to X_{ac}}$ (resp. $\Gamma|_{X_{ac} \to X_{ac}}$) means that the

domain of transition function f (resp. Γ function) was reduced to pairs in $X_{ac} \times \Sigma$ (resp. states in X_{ac}). Notice that, Ac operation does not affect the languages generated and marked by the original automaton.

Coaccessible Part

A state $x \in X$ of an automaton G is coaccessible if there exists a string $s \in \Sigma^*$ such that $f(x, s) \in X_m$. Otherwise, x is a non-coaccessible.

The $CoAc$ operation excludes all non-coaccessible states of an automaton G, being formally defined as:

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

where $X_{coac} = \{x \in X : (\exists s \in \Sigma^*)[f(x, s) \in X_m]\}$, $f_{coac} = f|_{X_{coac} \times \Sigma \to X_{coac}}$, $\Gamma_{coac} = \Gamma|_{X_{coac} \to X_{coac}}$, and $x_{0,coac} = x_0$, if $x_0 \in X_{coac}$, or undefined, otherwise.

Notice that, $CoAc$ operation can affect the language generated by the original automaton, but it does not modify the marked language. In addition, when $G = CoAc(G)$, G is said to be coaccessible, and $L(G) = \overline{L_m(G)}$, which implies that coaccessibility is closely related to nonblocking.

Trim Operation

$Trim$ operation is equivalent to applying Ac and $CoAc$, consecutively, that is, for an automaton G, $Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)]$. When $G = Trim(G)$, G is said to be a trim automaton.

Complement Operation

The automaton obtained by applying the complement operation to a deterministic automaton G is defined as:

$$G^C = (X \cup \{x_d\}, \Sigma, f^c, \Gamma^c, x_0, X_m^c)$$

where

$$f^c(x, \sigma) = \begin{cases} f(x, \sigma), \text{ if } \sigma \in \Gamma(x), \\ x_d, \text{ otherwise,} \end{cases}$$

$\Gamma^c(x) = \Sigma$, for all $x \in X \cup \{x_d\}$, and $X_m^c = (X \cup \{x_d\}) \setminus X_m$.

Notice that the language generated by automaton G^C is equal to Σ^*, and the language marked by G^C is equal to the complement of $L_m(G)$ with respect to Σ^*.

Example 2.6 *Consider automaton G_2 depicted in Figure 2.2(a). It can be checked that state 3 of G_2 is not accessible, and states 1 and 4 are not coaccessible. Therefore, the state transition diagrams of $Ac(G_2)$ and $CoAc(G_2)$ are those presented in Figures 2.2(b) and 2.2(c), respectively. Moreover, from the definition of $Trim$ operation, we can conclude that states 1, 4 and 3 are not states of $trim(G_2)$, whose state transition diagram is depicted in Figure 2.2(d). Finally, automaton G_2^C, obtained from G_2 by applying the complement operation, is depicted in Figure 2.2(e). Notice that, $L_m(G_2^C) = \Sigma^* \setminus \{\beta\sigma\}\{\alpha, \beta, \gamma\}^*$, which is equal to the complement of $L_m(G_2) = \{\beta\sigma\}\{\alpha, \beta, \gamma\}^*$ with respect to $\Sigma^* = \{\alpha, \beta, \gamma, \sigma\}^*$.*

We will now present two composition operations on automata, product and parallel composition, that model different types of interconnection among automata that work concurrently. To do so, let us denote $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2}, X_{m_2})$. The difference between the product of G_1 and G_2, to be denoted by $G_1 \times G_2$, and the parallel composition of G_1 and G_2, to be denoted by $G_1 \| G_2$, concerns the private events of these automata. In both, product and parallel composition, a transition labeled by a common event (*i.e.*, those events in $\Sigma_1 \cap \Sigma_2$) can occur if, and only if, it occurs simultaneously in G_1 and G_2. On the other hand, product $G_1 \times G_2$ does not have transitions labeled by private events of G_1 or G_2, whereas these transitions can occur asynchronously in the parallel composition $G_1 \| G_2$. The product and the parallel composition of two automata are formally defined as follows.

Product

The product of automata G_1 and G_2 is the automaton

$$G_1 \times G_2 := Ac\left(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, \Gamma_{1\times 2}, (x_{0_1}, x_{0_2}), X_{m_1} \times X_{m_2}\right)$$

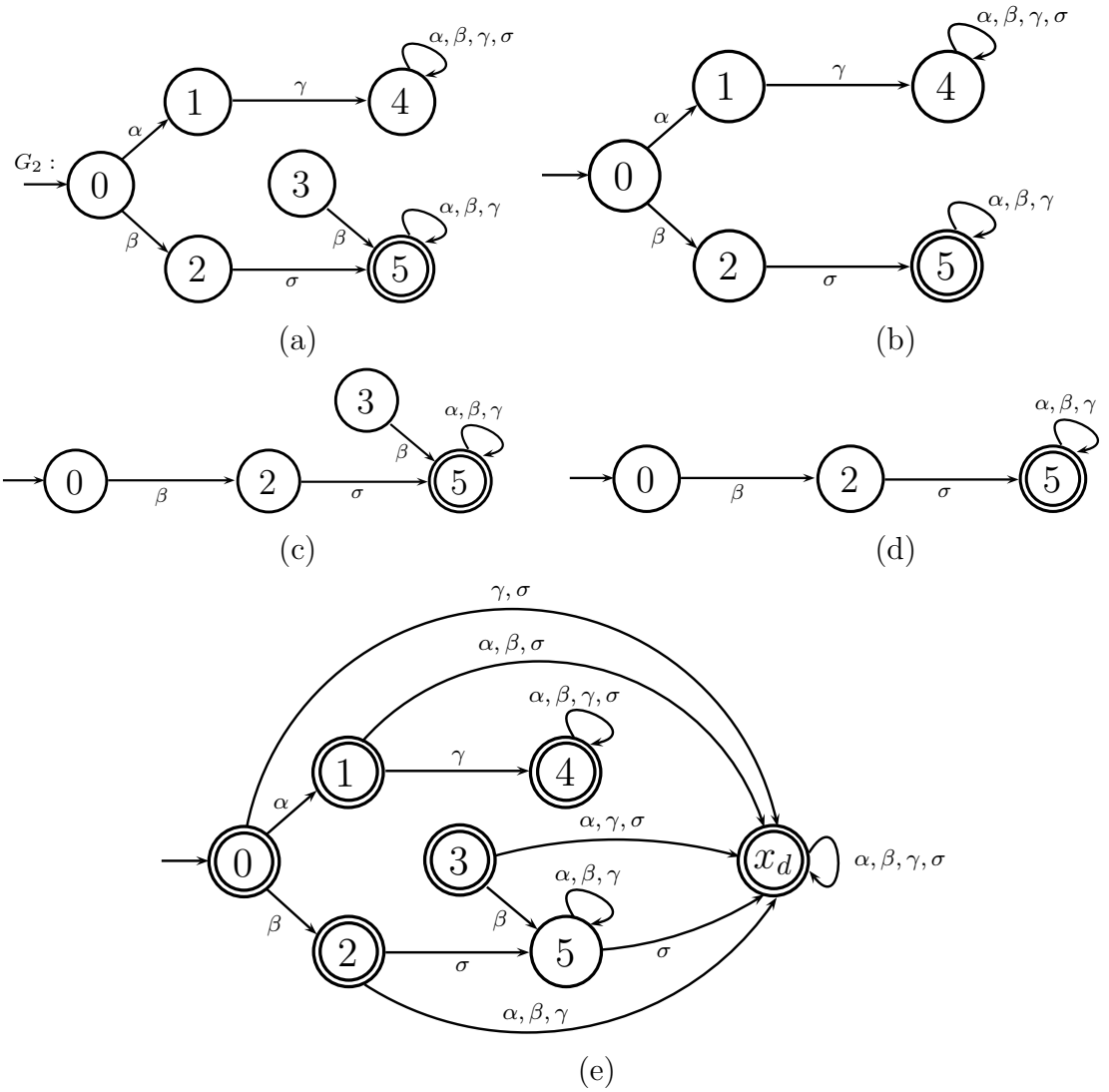


Figure 2.2: State transition diagrams of automata G_2 (a), $Ac(G_2)$ (b), $CoAc(G_2)$ (c), $trim(G_2)$ (d), and G_2^C (e).

where

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

and $\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$. It can be seen that the language generated and the language marked by $G_1 \times G_2$ are, respectively,

$$\begin{aligned} L(G_1 \times G_2) &= L(G_1) \cap L(G_2), \\ L_m(G_1 \times G_2) &= L_m(G_1) \cap L_m(G_2). \end{aligned}$$

Parallel Composition

The parallel composition of automata G_1 and G_2 is the automaton

$$G_1 \| G_2 := Ac\left(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, \Gamma_{1\|2}, (x_{0_1}, x_{0_2}), X_{m_1} \times X_{m_2}\right)$$

where

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

and $\Gamma_{1\|2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus \Sigma_2] \cup [\Gamma_2(x_2) \setminus \Sigma_1]$. The language generated and the language marked by $G_1 \| G_2$ are, respectively,

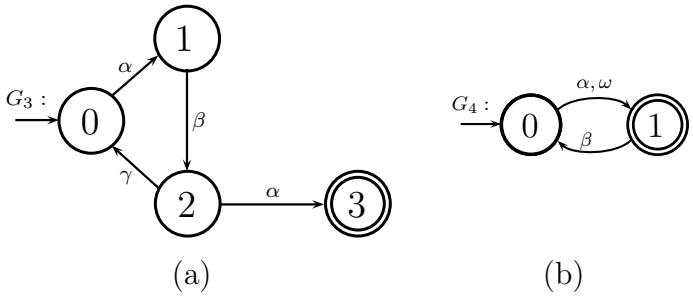
$$\begin{aligned} L(G_1 \| G_2) &= P_1^{-1}[L(G_1)] \cap P_2^{-1}[L(G_2)], \\ L_m(G_1 \| G_2) &= P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2)], \end{aligned}$$

where $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$, for $i = 1, 2$.

From the definitions of product and parallel composition, we can see that both operations are associative, and can be extend to three (or more) automata as follows:

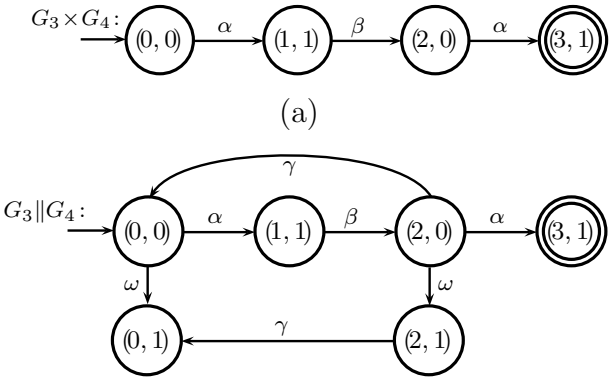
$$\begin{aligned} G_1 \times G_2 \times G_3 &:= (G_1 \times G_2) \times G_3 = G_1 \times (G_2 \times G_3); \\ G_1 \| G_2 \| G_3 &:= (G_1 \| G_2) \| G_3 = G_1 \| (G_2 \| G_3). \end{aligned}$$

Example 2.7 *Consider automata* $G_3 = (X_3, \Sigma_3, f_3, \Gamma_3, x_{0_3}, X_{m_3})$ *and* $G_4 = (X_4, \Sigma_4, f_4, \Gamma_4, x_{0_4}, X_{m_4})$*, depicted in Figures 2.3(a) and 2.3(b), respectively, and assume that* $\Sigma_3 = \{\alpha, \beta, \gamma\}$ *and* $\Sigma_4 = \{\alpha, \beta, \omega\}$*. The automata obtained by the product and parallel composition of* G_3 *and* G_4 *are presented in Figures 2.4(a) and 2.4(b), respectively. Notice that, although the private event* γ *(resp.* ω*) is active in state* 2 *of* G_3 *(resp.* 0 *of* G_4*), this event is not active in state* $(2, 0)$ *(resp. states* $(0, 0)$ *and* $(2, 0)$*) of automaton* $G_3 \times G_4$*, however, event* γ *(resp.* ω*) is active in states* $(2, 0)$ *and* $(2, 1)$ *(resp.* $(0, 0)$ *and* $(2, 0)$*) of* $G_3 \| G_4$*.*



(a) (b)

Figure 2.3: State transition diagrams of automata G_3 (a) and G_4 (b).



(a)

(b)

Figure 2.4: State transition diagrams of automata $G_4 \times G_5$ (a) and $G_4 \| G_5$ (b).

2.3.2 Subautomata

Let $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2}, X_{m_2})$ denote automata. We say that G_1 is a subautomaton of G_2 if the state transition diagram of G_1 is a subgraph of the state transition diagram of G_2. Formally, G_1 is a subautomaton of G_2, denoted by $G_1 \sqsubseteq G_2$, if

$$(\forall s \in L(G_1),\ f_1(x_{0_1}, s) = f_2(x_{0_2}, s)) \wedge (X_{m_1} = X_{m_2} \cap X_1).$$

Notice that this condition implies that $X_1 \subseteq X_2$, $x_{0_1} = x_{0_2}$, $L(G_1) \subseteq L(G_2)$ and $L_m(G_1) \subseteq L_m(G_2)$.

2.3.3 Language Projections and Observer Automaton

The observer automaton is used to model the behavior of an automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ over a set $\Sigma_o \subseteq \Sigma$ of observable events. Namely, the observer of G with respect to Σ_o, denoted by $Obs(G, \Sigma_o)$, is a deterministic automaton whose generated and marked languages are, respectively, $P_o[L(G)]$ and $P_o[L_m(G)]$, where

P_o denotes the projection $P_o : \Sigma^* \to \Sigma_o^*$.

In order to formally define $Obs(G, \Sigma_o)$, let us first define the unobservable reach of a state $x \in X$ as:

$$UR(x, \Sigma_o) := \{y \in X : (\exists t \in (\Sigma \setminus \Sigma_o)^*)[f(x, t) = y]\}$$

and, similarly, for all $B \in 2^X$,

$$UR(B, \Sigma_o) := \bigcup_{x \in B} UR(x, \Sigma_o).$$

We can now define the observer of G with respect to Σ_o as:

$$Obs(G, \Sigma_o) := Ac(2^X, \Sigma_o, f_{obs}, \Gamma_{obs}, UR(x_0, \Sigma_o), X_{m_{obs}}),$$

where

$$\begin{aligned} f_{obs}(B, \sigma_o) &= UR(\{x \in X : (\exists x' \in B)[f(x', \sigma_o) = x]\}, \Sigma_o), \forall (B, \sigma_o) \in 2^X \times \Sigma_o, \\ \Gamma_{obs}(B) &= \{\sigma_o \in \Sigma_o : f_{obs}(B, \sigma_o)!\}, \forall B \in 2^X, \\ X_{m_{obs}} &= \{B \in 2^X : B \cap X_m \neq \emptyset\}. \end{aligned}$$

The observer automaton can be constructed by using the following algorithm.

Algorithm 2.1 (Construction of Observer Automaton [72])

Inputs:

- $G = (X, \Sigma, f, \Gamma, x_0, X_m)$*;*
- Σ_o*: set of observable events.*

Output:

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

Step 1: Define $x_{0,obs} = UR(x_0, \Sigma_o)$*,* $X_{obs} = \{x_{0,obs}\}$ *and* $X_{obs}^{new} = X_{obs}$.

Step 2: Set $X_{obs}^{temp} = X_{obs}^{new}$ *and* $X_{obs}^{new} = \emptyset$.

Step 3: For each $B \in X_{obs}^{temp}$*:*

- *3.1:* $\Gamma_{obs}(B) = \left(\bigcup_{x \in B} \Gamma(x)\right) \cap \Sigma_o$.
- *3.2: For each* $\sigma \in \Gamma_{obs}(B)$*:*

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

(b) If $x_{obs} \notin X_{obs}$, *then* $X_{obs}^{new} \leftarrow X_{obs}^{new} \cup \{x_{obs}\}$.

Step 4: If $X_{obs}^{new} \neq \emptyset$, *then* $X_{obs} \leftarrow X_{obs} \cup X_{obs}^{new}$ *and return to Step 2.*

Step 5: $X_{m,obs} = \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$.

The following example illustrates the use of Algorithm 2.1.

Example 2.8 *Consider automaton* G_3 *initially shown in Figure 2.3(a), and also depicted in Figure 2.5(a), and assume* $\Sigma_o = \{\beta, \gamma\}$. *Automaton* $Obs(G_3, \Sigma_o)$, *depicted in Figure 2.5(b), was obtained by using Algorithm 2.1 as follows:*

Step 1: $x_{0,obs} = \{0, 1\}$, $X_{obs} = \{\{0, 1\}\}$ *and* $X_{obs}^{new} = X_{obs}$;

Step 2: $X_{obs}^{temp} = \{\{0, 1\}\}$ *and* $X_{obs}^{new} = \emptyset$;

Step 3: $\Gamma_{obs}(\{0, 1\}) = \{\beta\}$, $f_{obs}(\{0, 1\}, \beta) = \{2, 3\}$ *and* $X_{obs}^{new} = \{\{2, 3\}\}$;

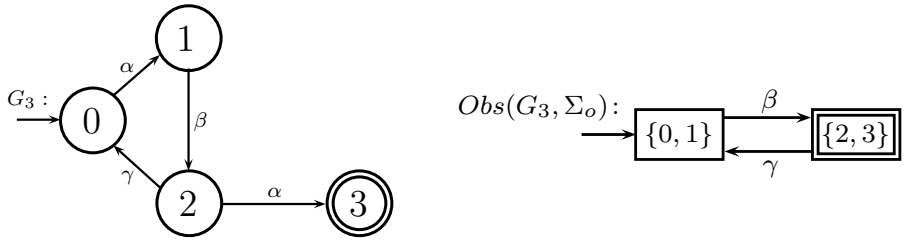
Step 4: Since $X_{obs}^{new} \neq \emptyset$, $X_{obs} \leftarrow \{\{0, 1\}, \{2, 3\}\}$ *and we return to Steps 2;*

Step 2: $X_{obs}^{temp} = \{\{2, 3\}\}$ *and* $X_{obs}^{new} = \emptyset$;

Step 3: $\Gamma_{obs}(\{2, 3\}) = \{\gamma\}$, $f_{obs}(\{2, 3\}, \gamma) = \{0, 1\}$ *and* $X_{obs}^{new} = \emptyset$;

Step 4: Since $X_{obs}^{new} = \emptyset$, *then the accessible part of* $Obs(G_3, \Sigma_o)$ *is complete;*

Step 5: $X_{m,obs} = \{\{2, 3\}\}$.



(a) (b)

Figure 2.5: State transition diagrams of automata G_3 (a) and $Obs(G_3, \Sigma_o)$ (b).

State Partition Automaton

Let $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ and $Obs(G, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0_{obs}}, X_{m_{obs}})$ be an automaton and its observer with respect Σ_o, respectively. Automaton G is said to be state partition if, for all $B, B' \in X_{obs}$, $B \neq B'$ implies that $B \cap B' = \emptyset$ [22, 73–75]. When automaton G is not state partition, an equivalent state partition automaton

can always be obtained by computing [76]

$$G_{sp} = G \| Obs(G, \Sigma_o),$$

such that $L(G_{sp}) = L(G)$ and $L_m(G_{sp}) = L_m(G)$. The state partition property is useful in the computation of the supremal normal sublanguage [74], and, as we will present in Chapter 3, we use this concept to compute the supremal relatively observable sublanguage of a given language.

Inverse Projection

We sometimes need to compute an automaton that represents the inverse projections of the generated and marked languages of an automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ with respect to a projection $P_l : \Sigma_l^* \to \Sigma^*$. To do so, we can add self-loops at each state of G labeled by the events in $(\Sigma_l \setminus \Sigma)$. The generated and marked languages of the automaton obtained through this procedure are $P_l^{-1}[L(G)]$ and $P_l^{-1}[L_m(G)]$, respectively.

2.4 Discrete Event Systems Subject to Loss of Observations

In this section, we present the model for DES subject to intermittent loss of observations proposed by CARVALHO *et al.* [62], which can be used to handle loss of observations caused by either sensor malfunction or communication problems. CARVALHO *et al.* [62] consider the case when the set of events of a DES is partitioned as follows:

$$\Sigma = \Sigma_{uo} \dot{\cup} \Sigma_{lo} \dot{\cup} \Sigma_{nlo},$$

where Σ_{uo} is the set of unobservable events, Σ_{lo} is the set of observable events subject to loss of observations, and Σ_{nlo} is the set of observable events that are not subject to loss of observations. In order to represent loss of observations, the renaming function ℓ is defined in domain Σ_{lo}, where, for all $\sigma \in \Sigma_{lo}$, $\ell(\sigma) = \sigma_l$. In addition, the set of events $\Sigma_{lo}^l = \{\ell(\sigma) : \sigma \in \Sigma_{lo}\}$ is defined, and, thus, the set of events becomes $\Sigma_{dil} = \Sigma \cup \Sigma_{lo}^l$. The operation that determines the language over Σ_{dil} that

is generated by the DES in the presence of loss of observations is formally defined as follows.

Definition 2.5 (Dilation [62]) *Dilation is the mapping* $D : \Sigma^* \rightarrow 2^{\Sigma^*_{dil}}$*, recursively defined as:*

$$\begin{aligned} D(\varepsilon) &:= \{\varepsilon\}; \\ D(\sigma) &:= \begin{cases} \{\sigma\}, \text{ if } \sigma \in \Sigma \setminus \Sigma_{lo}; \\ \{\sigma, \sigma_l\}, \text{ if } \sigma \in \Sigma_{lo}; \end{cases} \\ D(s\sigma) &:= D(s)D(\sigma), \text{ for all } s \in \Sigma^* \text{ and } \sigma \in \Sigma. \end{aligned}$$

The extension of D *to domain* 2^{Σ^*}*, i.e., to languages, is defined as* $D(L) = \bigcup_{s \in L} D(s)$.

The idea behind the definition of dilation is to represent the loss of observation of an event σ by replacing it with $\sigma_l = \ell(\sigma)$. For example, by assuming $\Sigma = \{\alpha, \beta\}$ and $\Sigma_{lo} = \{\beta\}$, the dilation of string $s = \beta\alpha\beta$ is $D(s) = \{\beta\alpha\beta, \beta_l\alpha\beta, \beta\alpha\beta_l, \beta_l\alpha\beta_l\}$, where: *(i)* string $\beta\alpha\beta$ represents the case when no loss of observation occurs, *(ii)* string $\beta_l\alpha\beta$ (resp. $\beta\alpha\beta_l$) represents the case when only the observation of the first (resp. the second) occurrence of event β is lost, and *(iii)* string $\beta_l\alpha\beta_l$ represents the case when both observations of event β are lost.

Consider a DES modeled by an automaton G and let $\Sigma_o = \Sigma_{lo} \dot{\cup} \Sigma_{nlo}$ denote the set of observable events and $P_{dil,o} : \Sigma^*_{dil} \rightarrow \Sigma^*_o$ the natural projection from Σ^*_{dil} over Σ^*_o. Then, for a string $s \in L(G)$, $P_{dil,o}(D(s))$ is the set of possible strings that can be observed when the plant executes s in the presence of loss of observations.

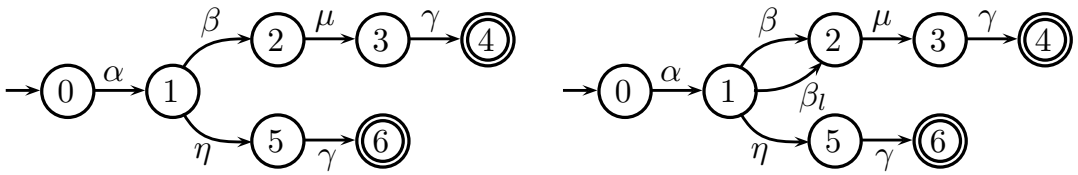
An automaton model G_{dil} that takes into account loss of observations was proposed in [62], being formed by adding to the transitions labeled with event $\sigma \in \Sigma_{lo}$, parallel transitions labeled with the corresponding event $\sigma_l \in \Sigma^l_{lo}$. G_{dil} is formally defined as:

$$G_{dil} = (X, \Sigma_{dil}, f_{dil}, \Gamma_{dil}, x_0, X_m), \tag{2.1}$$

where $\Sigma_{dil} = \Sigma \cup \Sigma^l_{lo}$, $\Gamma_{dil}(x) = D(\Gamma(x))$, and f_{dil} is defined as follows: $\forall x \in X$, $f_{dil}(x, \sigma) = f(x, \sigma)$ for $\sigma \in \Sigma$, and $f_{dil}(x, \sigma_l) = f(x, \sigma)$ for $\sigma_l \in \Sigma^l_{lo}$. As proved in [62], $L(G_{dil}) = D(L(G))$ and $L_m(G_{dil}) = D(L_m(G))$.

The following example illustrates the use of dilation and the construction of automaton G_{dil}.

Example 2.9 *Let us consider the generated and marked languages of automaton G_5 depicted in Figure 2.6(a), where $\Sigma = \{\alpha, \beta, \sigma, \mu, \eta\}$ and $\Sigma_o = \{\alpha, \beta, \gamma\}$. If we assume that $\Sigma_{lo} = \{\beta\}$, then $D(L(G)) = \overline{\{\alpha\beta\mu\gamma, \alpha\beta_l\mu\gamma, \alpha\eta\gamma\}}$, and $D(L_m(G)) = \{\alpha\beta\mu\gamma, \alpha\beta_l\mu\gamma, \alpha\eta\gamma\}$. Notice that, for string $s_1 = \alpha\eta\gamma$, $D(s_1) = \{\alpha\eta\gamma\}$, which implies that the unique string that can be observed when the plant executes s is $\alpha\gamma$. On the other hand, when the plant executes string $s_2 = \alpha\beta\mu\gamma$, either string $\alpha\beta\gamma$ or string $\alpha\gamma$ is observed since $P_{dil,o}(D(s_2)) = P_{dil,o}(\{\alpha\beta\mu\gamma, \alpha\beta_l\mu\gamma\})$. Finally, automaton $G_{5_{dil}}$, whose generated and marked languages are, respectively, $D(L(G))$ and $D(L_m(G))$, is depicted in Figure 2.6(b).*



(a) (b)

Figure 2.6: Automata G_5 (a) and $G_{5_{dil}}$ (b).

Let $s, s_1, s_2 \in \Sigma^*$, and K, K_1 and K_2 languages defined over Σ. Let K_{dil} be a subset of Σ^*_{dil} not necessarily obtained by dilating some $K \in \Sigma^*$. Table 2.1 presents some properties of dilation proved in [1].

2.5 Supervisory Control

In the classical control theory, feedback control systems are frequently used to modify the behavior of a system. In the same way, in discrete event systems theory, we can use a feedback control to modify the behavior of the DES, when the language generated by the DES contains strings that violate specifications that we want to impose on the behavior of the system. This feedback control is called Supervisory control.

Let the behavior of the open-loop system be modeled by an automaton G. Then, the control specifications are defined with a view to preventing unacceptable strings

Table 2.1: Properties of Dilation [1].

P1	$K \subseteq D(K)$
P2	$s_1 \neq s_2 \Leftrightarrow D(s_1) \cap D(s_2) = \emptyset$
P3	$D(K_1K_2) = D(K_1)D(K_2)$
P4	$D(K_1) \cup D(K_2) = D(K_1 \cup K_2)$
P5	$D(K_1) \cap D(K_2) = D(K_1 \cap K_2)$
P6	$\overline{D(K)} = D(\overline{K})$
P7	$D(K_1 \setminus K_2) = D(K_1) \setminus D(K_2)$
P8	$K_1 \subseteq K_2 \Leftrightarrow D(K_1) \subseteq D(K_2)$.
P9	$K = K_{dil} \cap \Sigma^*$, for all $K \subseteq K_{dil} \subseteq D(K)$
P10	$D(s) \cap D(K) \neq \emptyset \Leftrightarrow D(s) \subseteq D(K)$

of $L(G)$ from happening; for example, when the order of certain events is not correct, or strings that reach deadlocks, livelocks, or unacceptable states, *e.g.*, a state that represents a buffer overflowed or the collision between machines. In accordance with the control specifications, we define the sublanguage L_a of $L(G)$, usually referred to as admissible language, that represents the legal behavior of the compensated system. When the compensated system behavior remains inside L_a, it is said to be safe. In the supervisory control problem, we want to design a supervisor that is able to make the compensated system be safe. We also have the following additional requirements: *(i)* the compensated behavior must be as permissible as possible, and *(ii)* blocking cannot occur.

In accordance with the supervisory control paradigm proposed by RAMADGE and WONHAM [23, 24], the supervisor can observe some, possibly all, of the events generated by G and, then, determines those events in the current set of active events of G that are allowed to occur. In other words, the supervisor can disable some, but not necessarily all, of the active events of G. Notice that, the supervisors can be restricted to observe a subset of the events generated by G (observable events), and they can be limited to disabling only a subset of the feasible events of G (controllable events).

In the following subsection, we define the supervisory control problem under full observation, *i.e.*, we assume that the supervisor is able to observe all of the events

generated by the DES.

2.5.1 Supervisory Control Problem

Consider a DES modeled by a language L defined over a set of events Σ, where $L = \overline{L}$ is the set of all strings that can be generated by the system. Without loss of generality, assume that L is the language generated by an automaton G, *i.e.*, $L = L(G)$. The supervisory control problem consists in designing a supervisor S that is able to interact with G in a feedback manner, as depicted in Figure 2.7, and make the compensated system S/G (read as "S controlling G") be safe, *i.e.*, $L(S/G) = L_a \subseteq L(G)$.

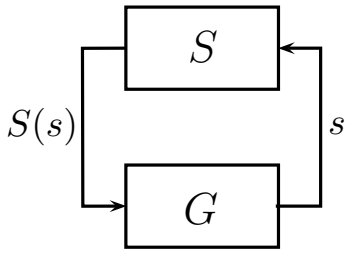


Figure 2.7: The feedback loop of supervisory control.

Formally, a supervisor is a function $S : L(G) \longrightarrow 2^{\Sigma}$, from the language generated by G to the power set of Σ, such that the new set of active events $\Gamma_N[f(x_0, s)]$, *i.e.*, the events that G can execute at state $f(x_0, s)$ under the control of S, is equal to $\Gamma[f(x_0, s)] \cap S(s)$. In other words, G is not able to execute an event σ at state $f(x_0, s)$ if σ does not belong to $S(s)$.

The compensated system S/G is a DES and its generated language is recursively defined as follows:

(i) $\varepsilon \in L(S/G)$;

(ii) $\forall s \in \Sigma^*$ and $\forall \sigma \in \Sigma$, $s\sigma \in L(S/G) \Leftrightarrow s \in L(S/G) \wedge s\sigma \in L(G) \wedge \sigma \in S(s)$.

The language marked by S/G is defined as:

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

Supervisory Control Under Partial Controllability

Assume, now, that the set of events Σ is partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, where Σ_c is the set of controllable events, that is, the events that can be disabled by a supervisor S, and Σ_{uc} is the set of uncontrollable events that cannot be disabled by S. In this regard, a supervisor is said to be admissible if $\Sigma_{uc} \cap \Gamma[f(x_0, s)] \subseteq S(s)$, for all $s \in L(G)$. Hereafter, we will consider admissible supervisors only.

Definition 2.6 (controllability) *Let K and L denote languages defined over a set of events Σ, such that $K \subseteq L$ and $L = \overline{L}$, and assume that Σ is partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Then, K is controllable with respect to L and Σ_{uc} if $\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}$.*

Definition 2.6 can be interpreted as follows. If K is controllable, then for all string $s \in \overline{K}$ and for all uncontrollable event $\sigma_{uc} \in \Sigma_{uc}$, if string $s\sigma_{uc}$ belongs to L, it also belongs to $\overline{K}$. Notice that, controllability is a property of the prefix-closure of a language, *i.e.*, a language K is controllable if, and only if, $\overline{K}$ is controllable.

Controllability becomes an important property because, as stated by the following theorem, it is associated with the existence of a supervisor that is able to make the language generated by the compensated system equal to the prefix of a given admissible language.

Theorem 2.1 (controllability theorem) *Consider a DES modeled by an automaton G, where $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Let $K \subseteq L(G)$, where $K \neq \emptyset$. Then, there exists a supervisor S such that $L(S/G) = \overline{K}$ if, and only if, K is controllable with respect to $L(G)$ and Σ_{uc}.*

In order to satisfy the language permissiveness objective when an admissible language K is not controllable, we must restrict to a sublanguage (*i.e.*, a subset) of K that is controllable, and search for the "largest" sublanguage of K that is controllable, where "largest" here concerns set inclusion. This language is called the supremal controllable sublanguage of K and is denoted by $K^{\uparrow C}$.

It can be checked that controllability is closed under set union, that is, if languages K_i, $i = 1, 2, \ldots, n$, are controllable with respect to $L(G)$ and Σ_{uc}, then so is language $K_1 \cup K_2 \cup \ldots \cup K_n$. As a consequence, it can be concluded that the

supremal controllable sublanguage of a given language K always exists. In the worst case, $K^{\uparrow C} = \emptyset$. On the other hand, if K is controllable, then $K^{\uparrow C} = K$.

Another important language associated with K is the infimal prefix-closed and controllable superlanguage of K, denoted by $K^{\downarrow C}$. This language is such that all prefix-closed and controllable languages that are supersets of K, are also supersets of $K^{\downarrow C}$. The existence of $K^{\downarrow C}$ is proved by using the following property of controllability: if the prefix-closed languages $\overline{K_i}$, $i = 1, 2, \ldots, n$, are controllable with respect to $L(G)$ and Σ_{uc}, then so is the prefix-closed language $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_n}$. Moreover, in the worst case, $K^{\downarrow C} = L(G)$. On the other hand, $K^{\downarrow C} = \overline{K}$ if K is controllable.

Let K and L be regular languages, and consider automata $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$, where $L(H) = \overline{K}$ and $L_m(H) = K$, and $G = (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_{m_g})$, whose generated language is L. The computational complexity of checking if K is controllable with respect to L and Σ_{uc} is $O(|X_h| \cdot |X_g| \cdot |\Sigma|)$, where notation $|\ |$ is used to denote the cardinality of a set. In addition, the computation of $K^{\uparrow C}$ is $O(|X_h|^2 \cdot |X_g|^2 \cdot |\Sigma|)$, and becomes $O(|X_h| \cdot |X_g| \cdot |\Sigma|)$ when K is prefix-closed. The computation of $K^{\downarrow C}$ is also $O(|X_h| \cdot |X_g| \cdot |\Sigma|)$ [68, ch.3].

2.5.2 Supervisory Control Under Partial Observation

In the supervisory control problem described in Subsection 2.5.1, we assume that the supervisor is able to observe all of the events generated by the system. We now assume that the set of events Σ is partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o is the set formed with the observable events, *i.e.*, those events whose occurrences can be observed by the supervisor, and Σ_{uo} is the set formed with the unobservable events, *i.e.*, those events whose occurrences cannot be directly observed by the supervisor. The main reason why an event is unobservable is the lack of sensors that are able to record its occurrences.

It is worth remarking that, when $\Sigma_{uo} \neq \emptyset$, the supervisor decides which events will be disabled based on the projection over Σ_o^* of the string generated by the plant. This is equivalent to saying that the supervisor makes its decision based on $P_o(s)$, where $P_o : \Sigma^* \to \Sigma_o^*$, and not on s, as illustrated in Figure 2.8. As a consequence of this partial observation, two different strings s_1 and s_2 with the same projection lead to the same control action.

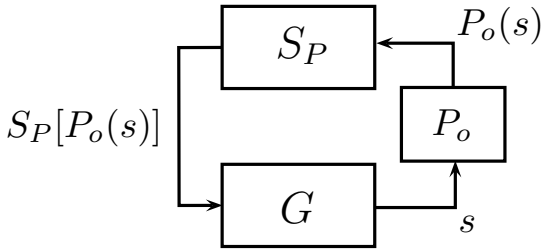


Figure 2.8: The feedback loop of supervisory control under partial observation.

Formally, a partial observation supervisor, or simply, a P-supervisor, is a mapping

$$\begin{aligned} S_P \;:\; P_o(L) &\rightarrow 2^{\Sigma} \\ P_o(s) &\mapsto S_P[P_o(s)] \end{aligned}$$

where $S_P[P_o(s)]$ is such that $\Gamma_N[f(x_0, s)] = \Gamma[f(x_0, s)] \cap S_P[P_o(s)]$.

The existence of unobservable events makes necessary an admissibility condition for P-supervisors different from that presented in the previous subsection. To this end, let $t = t'\sigma \in P_o(L(G))$ and $\sigma \in \Sigma_o$ and define language $L_t = P_o^{-1}(t')\{\sigma\}(S_P(t) \cap \Sigma_{uo})^* \cap L(G)$. Then, supervisor S_P is admissible if, for all $t \in P_o(L(G))$,

$$\Sigma_{uc} \cap \left[\bigcup_{s \in L_t} \Gamma[f(x_0, s)] \right] \subseteq S_P(t).$$

The compensated system S_P/G is a DES and its generated and marked languages are defined as follows.

Definition 2.7 *The language generated by S_P/G is recursively defined as:*

(i) $\varepsilon \in L(S_P/G)$;

(ii) $\forall s \in \Sigma^*$ *and* $\forall \sigma \in \Sigma$, $s\sigma \in L(S_P/G) \Leftrightarrow (s \in L(S_P/G) \wedge (s\sigma \in L(G)) \wedge (\sigma \in S_P[P_o(s)])$.

The language marked by S_P/G is defined as:

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

Under partial observation, the existence of a supervisor that is able to achieve a given specification is not only associated with controllability, but also must satisfy the following property.

Definition 2.8 (observability[77]) *Let K and L denote languages defined over a set of events Σ, such that $K \subseteq L$ and $L = \overline{L}$, and let $P_o : \Sigma^* \to \Sigma_o^*$ with $\Sigma_o \subseteq \Sigma$. Then, K is observable with respect to L and P_o if, for all $s' \in \overline{K}$ and for all $\sigma \in \Sigma$,*

$$(s'\sigma \in L \setminus \overline{K}) \Rightarrow (\nexists s \in \overline{K})[(P_o(s') = P_o(s)) \wedge (s\sigma \in \overline{K})]$$

or, equivalently,

$$(s'\sigma \in L \setminus \overline{K}) \Rightarrow P_o^{-1}[P_o(s')]\{\sigma\} \cap \overline{K} = \emptyset.$$

Let K and L be regular languages, and consider automata $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$ and $G = (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_{m_g})$ whose generated languages are $\overline{K}$ and L, respectively. We can check if K is observable with respect to L and P_o in polynomial time, by using the algorithm proposed in [78], that has worst-case complexity equal to $O(|X_h|^2 \cdot |X_g| \cdot |\Sigma|)$.

The following theorem concerns the language generated by the compensated system S_P/G.

Theorem 2.2 (controllability and observability theorem) *Consider a DES modeled by an automaton G and assume that $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$. Let P_o denote the projection $P_o : \Sigma^* \to \Sigma_o^*$. Then, for a sublanguage K of $L(G)$, where $K \neq \emptyset$, there exists a P-supervisor S_P such that $L(S_P/G) = \overline{K}$ if, and only if, K is controllable with respect to $L(G)$ and Σ_{uc} and observable with respect to $L(G)$ and P_o.*

In contrast to controllability, observability is not closed under set union, *i.e.*, if two languages K_1 and K_2 are observable, then $K_1 \cup K_2$ need not be observable. As a consequence, the supremal observable sublanguage does not exist in general. On the other hand, it can be checked, from Definition 2.8, that language observability is closed under the intersection of prefix-closed languages, *i.e.*, if the prefix-closed languages $\overline{K_i}$, $i = 1, 2, \ldots, n$, are observable with respect to $L(G)$ and P_o, then so is the prefix-closed language $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_n}$. As a consequence, there always exists the infimal prefix-closed controllable and observable superlanguage of a language K, denoted here by $K^{\downarrow CO}$, that satisfies the following conditions: *(i)* $K^{\downarrow CO} \supseteq K$ is prefix-closed, controllable and observable; *(ii)* if there exists a prefix-closed language

$K'' \supseteq K$, controllable and observable, then $K^{\downarrow CO} \subseteq K''$. RUDIE and WONHAM [79] and KUMAR and SHAYMAN [80] have proposed algorithms that compute $K^{\downarrow CO}$ in exponential time, for a given regular language K. More recently, MASOPUST [81] has shown that the state complexity of the infimal prefix-closed observable language $K^{\downarrow O}$ is exponential in the worst case, *i.e.*, the number of states of the minimal deterministic finite automaton required to represent $K^{\downarrow O}$ is exponential in the worst case. Based on these observations, we may conjecture that there may not exist an algorithm for the computation of a deterministic finite automaton that generates $K^{\downarrow CO}$, which has, in the worst case, polynomial time complexity, since all known methods to compute $K^{\downarrow CO}$ requires the calculation of $K^{\downarrow O}$.

The Property of Normality

Let us assume that the admissible language K is not observable. In this case, since the supremal observable sublanguage of a given language does not exist in general, we cannot deal with observability directly in order to remain within the admissible behavior. One way to circumvent this drawback is by replacing the property of observability with normality [21] in order to design P-supervisors that achieve a controllable and observable sublanguage of K.

Definition 2.9 (Normality) *Let K and L denote languages defined over a set of events Σ, such that $K \subseteq L$ and $L = \overline{L}$. Let Σ be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ and $P_o : \Sigma^* \to \Sigma_o^*$. Then, K is normal with respect to L and P_o if*

$$P_o^{-1}(P_o(\overline{K})) \cap L = \overline{K}.$$

Definition 2.9 can be interpreted as follows: a language K is normal with respect to L and P_o if, and only if, K can be recovered from its projection $P_o(\overline{K})$ and language L. Notice that, like the properties of controllability and observability, normality is a property of the prefix-closure of a language, *i.e.*, a language K is normal if, and only if, $\overline{K}$ is normal. Furthermore, it can be checked that the property of normality is closed under set union, and, consequently, for every language $K \subseteq L$, the supremal normal sublanguage of K, denoted by $K^{\uparrow N}$, and the supremal controllable and normal sublanguage of K, denoted by $K^{\uparrow CN}$, always exist.

It can be proved that, if K is normal with respect to L and P_o, then K is observable with respect to L and P_o, but the reverse is not true in general. Thus, according to Theorem 2.2, there exists a P-supervisor such that $L(S_P/G) = \overline{K}$ if K is controllable with respect to $L(G)$ and Σ_{uc} and normal with respect to $L(G)$ and P_o. Finally, when $\Sigma_c \subseteq \Sigma_o$ (*i.e.*, all controllable events are also observable), it can be shown that the properties of normality and observability become equivalent.

Remark 2.1 *Let us consider two languages K and L, defined over a set of events $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, such that $K \subset L$. Notice that, if there exist a string $s \in K$ and an event $\sigma_{uo} \in \Sigma_{uo}$ such that $s\sigma_{uo} \in L \setminus K$, then $P_o^{-1}(P_o(K)) \cap L \not\subset K$ since $s\sigma_{uo} \in P_o^{-1}(P_o(K)) \cap L$, which implies that K is not normal with respect to L and P_o. Therefore, we can conclude that the property of normality cannot be used to design supervisors that must disable unobservable events.*

Realization of P-supervisors

After constructing the automaton that models a controllable and observable admissible language K, where $K \subseteq L$, we need to represent the P-supervisor in a convenient manner, since listing the control action $S_P(s_o)$, for each $s_o \in P_o(\overline{K})$, may be somewhat hard.

When K and L are regular languages, supervisor S_P can be represented by a finite-state automaton. Moreover, the parallel composition, shown in Subsection 2.3.1, can be used to determine the behavior of the compensated system S_P/G.

The automaton that models P-supervisor S_P, called realization of S_P and denoted by R_s, can be constructed by using the following algorithm.

Algorithm 2.2 (Realization of a P-supervisor)

Inputs:

- *G: deterministic automaton whose generated language is L;*
- *H: deterministic automaton whose marked language is K (controllable and observable);*
- *Σ_o: set of observable events.*

Output:

- *$R_s = (X_s, \Sigma, f_s, \Gamma_s, x_{0_s}, X_{m_s})$: realization of the P-supervisor.*

Step 1: Compute automaton $R := Trim(H) = (X_r, \Sigma, f_r, \Gamma_r, x_{0_r}, X_{m_r})$ *whose generated language is* $\overline{K}$.

Step 2: Construct observer $R_{obs} = Obs(R, \Sigma_o)$ *by using Algorithm 2.1.*

Step 3: Construct automaton R_s *from* R_{obs} *by adding, for each state* x_{obs} *of* R_{obs}*, self-loops labeled by the unobservable events that belong to* $\bigcup_{x \in x_{obs}} \Gamma_r(x)$*. In addition, mark all states of* R_s.

Automaton R_S obtained with Algorithm 2.2 is such that

$$S_p(s_o) = \Gamma_s(f_s(x_0, s_o)), \text{ for all } s_o \in P_o(\overline{K}),$$

and, consequently, $L(R_S \| G) = L(S_P/G) = \overline{K}$ and $L_m(R_S \| G) = L_m(S_P/G) = L_m(G) \cap \overline{K}$.

2.5.3 Relative Observability

Another way to circumvent the deficiency of language observability regarding the non-existence of a supremal observable sublanguage, a new definition of observability, called relative observability, has been recently proposed [15], being formally defined as follows.

Definition 2.10 (relative observability) *Given a language* $C \subseteq L_m(G)$*, we say that a language* $K \subseteq C$ *is relatively observable*[2] *with respect to* $\overline{C}$*,* G *and* P_o*, or simply* $\overline{C}$*-observable, if* $\forall s, s' \in \Sigma^*$ *such that* $P_o(s) = P_o(s')$*, and* $\forall \sigma \in \Sigma$*,*

$$(s\sigma \in \overline{K}) \wedge (s' \in \overline{C}) \wedge (s'\sigma \in L(G)) \Rightarrow s'\sigma \in \overline{K}. \tag{2.2}$$

Language $\overline{C}$ is referred to as ambient language.

Conditional statement (2.2) can be better explained with the help of the diagrams depicted in Figures 2.9(a) and 2.9(b). It can be seen, with the help of Figure 2.9(a), that observability (Definition 2.8) requires that, for every pair (s, s') such that $s, s' \in \overline{K}$ and $P_o(s) = P_o(s')$, the one-step continuations of s in $\overline{K}$ be consistent with those continuations of s'. On the other hand, according to Figure 2.9(b), relative

[2]In [15], there is an additional condition for relative observability as follows: $\forall s, s' \in \Sigma^*$ if $P_o(s) = P_o(s')$, then $(s \in K) \wedge (s' \in \overline{C} \cap L_m(G)) \Rightarrow s' \in K$. However, this second condition is necessary only if *marking nonblocking supervisors* are applied.

observability enlarges the range of verification for every pair (s, s') such that $s \in \overline{K}$, $s' \in \overline{C}$ and $P_o(s) = P_o(s')$. It can also verified from Figures 2.9(a) and 2.9(b),

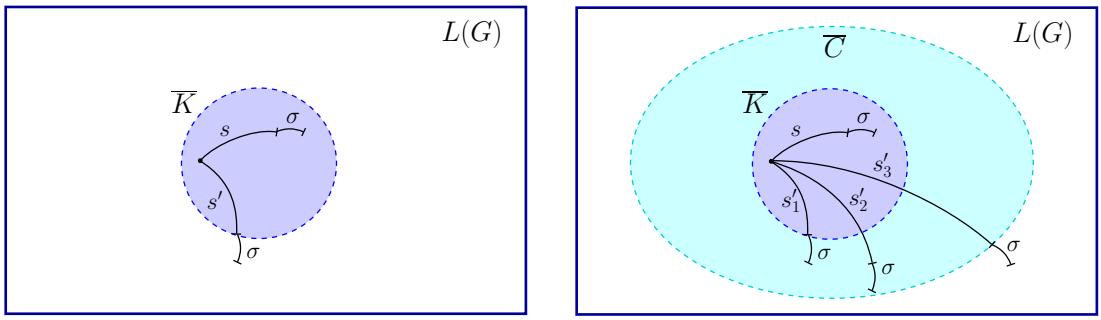


Figure 2.9: Illustrative diagrams of observability (Definition 2.8) (a), and relative observability (b). If $P_o(s) = P_o(s')$, then K is not observable. On the other hand, if $P_o(s) = P_o(s'_i)$, for $i = 1, 2$ or 3, then K is not $\overline{C}$-observable.

that the relative observability implies observability. In addition, when $C = K$, relative observability is equivalent to observability. Finally, notice that the larger the ambient language, the stronger (*i.e.*, more restrictive) the relative observability property will be.

CAI *et al.* [15] proved that relative observability is closed under set union operation, as opposed to observability, that does not possess this property; thus, the supremal relatively observable sublanguage always exists. Moreover, it is also proved in [15] that, the relative observability is weaker than normality [21], which implies that the supremal relatively observable sublanguage is larger compared to the supremal normal sublanguage, in general.

An algorithm for the computation of the supremal relatively observable sublanguage of a regular language K with respect to $\overline{K}$, G and P_o, *i.e.*, the ambient language is set as $\overline{K}$, has been proposed in [15]. This algorithm has doubly exponential complexity, namely, its complexity is $O(2^{[(2^{|X_h|}\cdot|X_h|+1)|X_g|+|X_h|]} \cdot |X_h| \cdot |\Sigma|)$, where $|X_g|$ (resp. $|X_h|$) is the number of states of G (resp. the automaton that recognizes K), and $|\Sigma|$ is the cardinality of the set of events of G. More recently, CAI *et al.* [20] proposed a new algorithm for the computation of the supremal relatively observable sublanguage of a regular language K with respect to $\overline{K}$, G and P_o whose computational complexity is $O\left(2^{3|X_g|\cdot|X_h|} \cdot |X_g|^6 \cdot |X_h|^7 \cdot |\Sigma|\right)$. In the next chapter, we will propose algorithms for the verification of relative observability of regular languages, that has polynomial time computational complexity, and for the computation of

the supremal relatively observable sublanguage, that has computational complexity lower than those of the algorithms proposed in [15] and [20].

The following example illustrates the use of relative observability.

Example 2.10 *Let us consider automaton G, depicted in Figure 2.10(a), where $\Sigma = \{\beta, \gamma, \delta, \mu, \nu, \eta\}$ is the set of events, and $\Sigma_o = \{\beta, \gamma, \delta\}$ and $\Sigma_{uo} = \{\mu, \nu, \eta\}$ are the sets of observable and unobservable events, respectively. In addition, consider languages C and K marked by the automata depicted in Figures 2.10(b) and 2.10(c), respectively. We can verify that K is not $\overline{C}$-observable with respect to G and P_o since, for example,*

(i) For strings $s_1 = \varepsilon \in \overline{K}$ and $s_1' = \eta \in \overline{C}$ and event β, $P_o(s_1) = P_o(s_1') = \varepsilon$, and $s_1\beta \in \overline{K}$ but $s_1'\beta \in L(G) \setminus \overline{K}$;

(ii) For strings $s_2 = \beta \in \overline{K}$ and $s_2' = \beta\nu \in \overline{C}$ and event δ, $P_o(s_2) = P_o(s_2') = \beta$, and $s_2\delta \in \overline{K}$ but $s_2'\delta \in L(G) \setminus \overline{K}$.

Notice that, case (i) also violates the observability condition (with respect to $L(G)$ and P_o), whereas case (ii) solely violates the $\overline{C}$-observability condition. The supremal $\overline{C}$-observable (with respect to $L(G)$ and P_o) sublanguage of K is the language marked by the automaton depicted in Figure 2.10(d). Finally, we can check that the supremal normal (with respect to $L(G)$ and P_o) sublanguage of K is empty, since, for all nonempty language $K' \subseteq K$, $s = \varepsilon \in \overline{K'}$, $s' = \mu\nu \in L(G) \setminus \overline{K'}$ and $s' \in P_o^{-1}(P_o(s)) \cap L(G)$.

Computation of Controllable and Observable Sublanguages by Using Relative Observability

Consider a system modeled by an automaton G with set of events $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$, and let K and C be languages such that $K \subseteq C \subseteq L_m(G)$. The supremal controllable (with respect to $L(G)$ and Σ_{uc}) and $\overline{C}$-observable (with respect to G and P_o) sublanguage of K, to be denoted here by $supCRO(K, \overline{C})$ can be computed by using the following iterative procedure [15]:

1. Set $i = 1$ and $K_0 = K$;

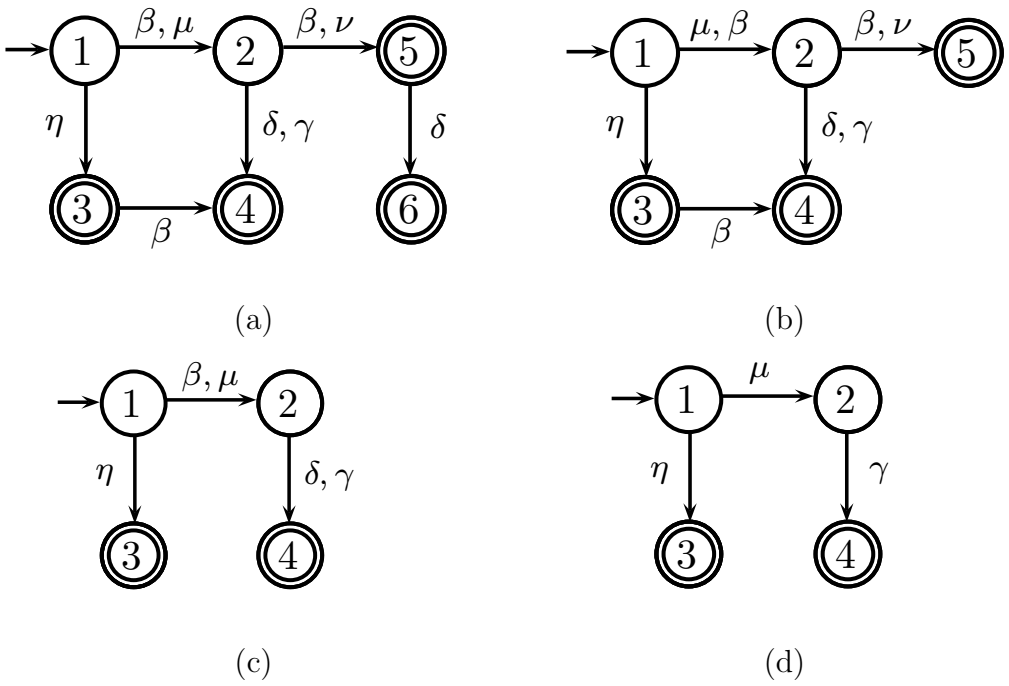


Figure 2.10: System automaton G (a), and automata whose marked languages are C (b), K (c) and $K^{\uparrow RO}$ (d), respectively.

2. Compute the supremal controllable (with respect to $L(G)$ and Σ_{uc}) sublanguage of K_{i-1}, denoted here by $K_{i-1}^{\uparrow C}$;

3. Compute the supremal $\overline{C}$-observable (with respect to G and P_o) sublanguage of $K_{i-1}^{\uparrow C}$, referred here to as K_i;

4. If $K_i = K_{i-1}$, then $supCRO(K, \overline{C}) = K_i$. Otherwise, set $i = i + 1$ and return to Step 2.

Notice that the language obtained at the end of this procedure is relatively observable with respect to an ambient language $\overline{C}$ previously determined. In order to obtain a more permissive controllable and observable sublanguage, CAI *et al.* [15] shrink the ambient language at each iteration, by making it be equal to K, in the first step, and equal to $K_{i-1}^{\uparrow C}$ in the next steps. The idea behind this choice of ambient languages comes from the intuition that at each iteration i, the language computed in Step 3 is a sublanguage of $K_{i-1}^{\uparrow C}$, and, thus, by setting this language as the ambient language, we may obtain a more permissive sublanguage since the smaller the ambient language, the weaker the relative observability property will be. Furthermore, the fact that the language computed after this step is relatively observable guarantees (regardless of the ambient language) that it is also observ-

able, besides being controllable. In order to distinguish language $supCRO(K, \overline{C})$ from that computed as proposed by CAI *et al.* [15] (*i.e.*, by shrinking the ambient language), the latter will be denoted by $K^{\uparrow Cro}$.

In Chapter 3, we will propose an algorithm, for the computation of a controllable and relatively observable sublanguage, that has the same steps of the procedure presented above. In this algorithm, besides shrinking the ambient language at the beginning of each execution of Step 3 as proposed by CAI *et al.* [15], we additionally shrink the ambient language during the execution of this step, *i.e.*, at each iteration of the algorithm used to compute the supremal relatively observable sublanguage of $K_{i-1}^{\uparrow C}$. As a consequence, the computed sublanguage may be more permissive than $supCRO(K, \overline{C})$ and $K^{\uparrow Cro}$. In order to distinguish the controllable and relatively observable sublanguage to be obtained by the algorithm to be proposed in Chapter 3 from the aforementioned sublanguages, it will be denoted by $K^{\uparrow CRO}$.

Chapter 3

New Algorithms for Relative Observability

In this chapter, we propose three new algorithms on relative observability. The first algorithm verifies if a regular language K is relatively observable with respect to a given ambient language $\overline{C}$ a plant G and a projection P_o. This algorithm has polynomial complexity, and, since the relative observability is equivalent to observability when $\overline{K}$ is chosen as ambient language, it can be also applied to verify if a language K is observable with respect to $L(G)$ and P_o.

The second algorithm computes the supremal relatively observable (with respect to $\overline{C}$, G and P_o) sublanguage of a regular language K. This algorithm has exponential complexity, and is more efficient than those proposed in [15] and [20]. The key to the success of this algorithm is a new property on relative observability which ensures that for any ambient language $\overline{C}$, there exists an equivalent reduced ambient language that is a subset of $\overline{C}$. It is worth remarking that the computational complexity of the algorithm proposed here for the computation of the supremal relatively observable sublanguage becomes polynomial when the automaton that marks K is state partition [22, 73–76].

The third algorithm, which is based on the second one, computes a controllable and observable sublanguage of a regular language K by using the relative observability property and shirking the ambient language at each algorithm iteration with a view to obtaining more permissive languages.

Preliminary versions of the results presented in this chapter were published in

[82, 83], and the current version was published in [84].

This chapter is organized as follows. We present and prove a new property of relative observability in Section 3.1. We propose an algorithm for the verification of relative observability in Section 3.2, and an algorithm for the computation of the supremal relatively observable sublanguage of regular languages in Section 3.3. We propose an algorithm for the computation of a controllable and relatively observable sublanguage of a regular language in Section 3.4. In Section 3.5, we analyze the complexities of the algorithms proposed in Sections 3.2, 3.3 and 3.4. Finally, we present some conclusions in Section 3.6.

3.1 An Equivalent Reduced Ambient Language

We will consider in this section the problem of finding a language $C_s \subseteq C$ for which if K is relatively observable with respect to $\overline{C}$, G and P_o, it is also relatively observable with respect to $\overline{C_s}$, G and P_o, and conversely. This result will play a key role in the algorithms proposed later on this chapter for the verification of relative observability and for the computation of the supremal relatively observable sublanguage. For simplicity, we only consider the case when the set of unobservable events of the plant is nonempty, *i.e.*, $\Sigma_{uo} \neq \emptyset$, since observability and relative observability conditions are automatically satisfied in the case when $\Sigma_{uo} = \emptyset$.

Lemma 3.1 *Let $K \subseteq C \subseteq L_m(G)$. Then, K is relatively observable with respect to $\overline{C}$, G and P_o if, and only if, K is relatively observable with respect to $\overline{C_s} = (\overline{K}\Sigma_{uo}^* \cap \overline{C})$, G and P_o.*

Proof: ($\Rightarrow$) It is straightforward and comes from the fact that $\overline{C_s} \subseteq \overline{C}$.

($\Leftarrow$) Assume, now, that K is not relatively observable with respect to $\overline{C}$, G and P_o. Then, there exist $s \in \overline{K}$, $s' \in \overline{C}$ and $\sigma \in \Sigma$ such that $s\sigma \in \overline{K}$, $s'\sigma \in L(G) \setminus \overline{K}$ and $P_o(s) = P_o(s')$. Without loss of generality, write $s' = s'_p s'_s$, where s'_p is the longest prefix of s' in $\overline{K}$ and $s'_s \in \Sigma^*$. Notice that, s'_s must satisfy one of following conditions: *(i)* $s'_s \in \Sigma_{uo}^*$ or *(ii)* $s'_s \in \Sigma^* \setminus \Sigma_{uo}^*$, *i.e.* s'_s has, at least, one observable event. Let us now consider each one of these possibilities:

(i) $s'_s \in \Sigma_{uo}^*$. In this case, $s' = s'_p s'_s \in \overline{K}\Sigma_{uo}^* \cap \overline{C}$. Therefore, K is not relatively observable with respect to $\overline{C_s}$, G and P_o.

(ii) $s'_s \in \Sigma^* \setminus \Sigma^*_{uo}$. Without loss of generality, write $s'_s = s'_{sp}\alpha s'_{ss}$, where $s'_{sp} \in \Sigma^*_{uo}$, $\alpha \in \Sigma_o$ and $s'_{ss} \in \Sigma^*$. Thus, $P_o(s') = P_o(s'_p s'_{sp})\alpha P_o(s'_{ss})$. Since $P_o(s) = P_o(s')$, we can write s as $s = s_p \alpha s_s$, where $P_o(s_p) = P_o(s'_p s'_{sp})$ and $P_o(s_s) = P_o(s'_{ss})$. Defining, now, $t = s_p$ and $t' = s'_p s'_{sp}$, it can be seen that $t\alpha \in \overline{K}$, $t' \in (\overline{K}\Sigma^*_{uo} \cap \overline{C})$, $t'\alpha \in L(G) \setminus \overline{K}$ and $P_o(t) = P_o(t')$, which implies that K is not relatively observable with respect to $\overline{C_s}$, G and P_o. ■

Therefore, in accordance with Lemma 3.1 instead of considering the ambient language $\overline{C}$, we can equivalently consider the reduced ambient language $\overline{C_s} = (\overline{K}\Sigma^*_{uo} \cap \overline{C})$ in all computations regarding relative observability.

3.2 Verification of Relative Observability

An equivalent way to state the relative observability definition given in (2.2) is as follows: a language K is $\overline{C}$-observable with respect to $L(G)$ and P_o if,

$$\begin{aligned}(\forall(s,\sigma) \in \overline{K} \times \Sigma),& \\ s\sigma \in \overline{K} \;\; \Rightarrow \;\; (\nexists s' \in \overline{C_s}) &\left[(s'\sigma \in L(G) \setminus \overline{K}) \wedge (P_o(s) = P_o(s'))\right].\end{aligned} \tag{3.1}$$

Notice that, in Expression 3.1 C has been replaced with C_s, as guaranteed by Lemma 3.1.

According to conditional statement (3.1), $\overline{C}$-observability is violated when there exist $s\sigma \in \overline{K}$ and $s'\sigma \in \overline{C_s}\Sigma \cap \overline{K}^C \cap L(G)$ (where $\overline{K}^C$ denotes the complement of $\overline{K}$ with respect to Σ^*), such that $P_o(s) = P_o(s')$. This observation suggests an algorithm for the verification of relative observability based on the comparison between the projections of languages $\overline{K}$ and $\overline{C_s}\Sigma \cap \overline{K}^C \cap L(G)$, like the algorithm proposed in [85] for the verification of codiagnosability.

Let $\Sigma_R = \{\sigma_R : \sigma \in \Sigma_{uo}\} \cup \Sigma_o$. Define the renaming function R, recursively, as follows: $R : \Sigma^* \longrightarrow \Sigma^*_R$, where: *(i)* $R(\varepsilon) := \varepsilon$, *(ii)* $R(\sigma) := \sigma$, if $\sigma \in \Sigma_o$, *(iii)* $R(\sigma) := \sigma_R$, if $\sigma \notin \Sigma_o$, and *(iv)* $R(s\sigma) = R(s)R(\sigma)$ for $s \in \Sigma^*$ and $\sigma \in \Sigma$. The inverse renaming function is the mapping $R^{-1} : \Sigma^*_R \longrightarrow \Sigma^*$, where $R^{-1}(s_R) = s$, such that $R(s) = s_R$. Both the renaming and the inverse renaming functions can be extended to languages by applying $R(s)$ and $R^{-1}(s)$ to all strings s in the language. The following result provides the basis for the algorithm we propose here.

Lemma 3.2 *Consider automata* $G_1 = (X_1, \Sigma, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ *and* $G_2 = (X_2, \Sigma, f_2, \Gamma_2, x_{0_2}, X_{m_2})$*, whose event sets are partitioned as* $\Sigma = \Sigma_{uo} \dot{\cup} \Sigma_o$*. Let* $G_m^R := (X_1, R(\Sigma), f_R, \Gamma_R, x_{0_1}, X_1)$ *where,* $\forall x \in X_1$, $\Gamma_R(x) = R(\Gamma_1(x))$ *and* $f_R(x, R(\sigma)) = f_1(x, \sigma)$, $\forall \sigma \in \Gamma_1(x)$*, and define* $V = G_m^R \| G_2 = (X_v, \Sigma \cup R(\Sigma), f_v, \Gamma_v, x_{0_v}, X_{m_v})$*. Then, for any event* $\sigma \in \Sigma$ *and a pair of strings* $(s_v\sigma, s_v R(\sigma)) \in L_m(V) \times L(V)$ *for which* $f_v(x_{0_v}, s_v) = (x_1, x_2)$*, there exists a pair of strings* $(s_1\sigma, s_2\sigma) \in L(G_1) \times L_m(G_2)$ *such that* $P_o(s_1) = P_o(s_2)$, $f_1(x_{0_1}, s_1) = x_1$ *and* $f_2(x_{0_2}, s_2) = x_2$*, and conversely.*

Proof: Define projections $P_R : [\Sigma \cup R(\Sigma)]^* \to R(\Sigma)^*$ and $P_\Sigma : [\Sigma \cup R(\Sigma)]^* \to \Sigma^*$.

($\Rightarrow$) Assume that there exists a pair of strings $(s_v\sigma, s_v R(\sigma)) \in L_m(V) \times L(V)$ such that $f_v(x_{0_v}, s_v) = (x_1, x_2)$. Notice that $L(V) = P_R^{-1}[L(G_m^R)] \cap P_\Sigma^{-1}[L(G_2)]$ and $L_m(V) = P_R^{-1}[L_m(G_m^R)] \cap P_\Sigma^{-1}[L_m(G_2)]$, since $V = G_m^R \| G_2$. Define $s_{1R} = P_R(s_v)$ and $s_2 = P_\Sigma(s_v)$, then: *(i)* $s_v R(\sigma) \in P_R^{-1}[L(G_m^R)] \Rightarrow P_R[s_v R(\sigma)] \in L(G_m^R) \Rightarrow s_{1R} R(\sigma) \in L(G_m^R)$, and; *(ii)* $s_v\sigma \in P_\Sigma^{-1}[L_m(G_2)] \Rightarrow P_\Sigma(s_v\sigma) \in L_m(G_2) \Rightarrow s_2\sigma \in L_m(G_2)$.

Since G_m^R is obtained from G_1 by applying the renaming function R and marking all of its states, it is easy to check that $L(G_m^R) = L_m(G_m^R) = R[L(G_1)]$. Therefore, by defining $s_1 = R^{-1}(s_{1R})$, and since $s_{1R} R(\sigma) \in L(G_m^R)$, we have that $s_1\sigma \in L(G_1)$. Notice that, $P_o(s_1) = P_\Sigma(s_{1R})$ and $P_o(s_2) = P_R(s_2)$, and since $s_{1R} = P_R(s_v)$ and $s_2 = P_\Sigma(s_v)$, we can conclude that $P_o(s_1) = P_\Sigma[P_R(s_v)]$ and $P_o(s_2) = P_R[P_\Sigma(s_v)]$. Finally, as $P_\Sigma[P_R(s_v)] = P_R[P_\Sigma(s_v)]$, then $P_o(s_1) = P_o(s_2)$.

Since G_m^R is obtained from G_1 by renaming its unobservable events, then the renamed unobservable events, $R(\Sigma_{uo})$, and the unobservable events, Σ_{uo}, become private events of G_1^R and G_2, respectively, in the parallel composition $V = G_1^R \| G_2$. Then, by the construction of V, it can be seen that, if a transition is labeled by an unobservable (resp. a renamed unobservable) event occurs, the first (resp. second) component of state of V does not modify. Therefore, we may conclude that $x_1 = f_R(x_{0_1}, s_{1R}) = f_1(x_{0_1}, s_1)$ and $x_2 = f_2(x_{0_2}, s_2)$.

($\Leftarrow$) Take now a pair of strings $(s_1\sigma, s_2\sigma) \in L(G_1) \times L_m(G_2)$ such that $P_o(s_1) = P_o(s_2)$, $f_1(x_{0_1}, s_1) = x_1$ and $f_2(x_{0_2}, s_2) = x_2$. Since $P_o(s_1) = P_o(s_2)$, we have that s_1 and s_2 are different only in the unobservable events. Therefore, $P_R^{-1}[R(s_1)] \cap P_\Sigma^{-1}(s_2) \neq \emptyset$, which implies that there exists $s_v \in P_R^{-1}[R(s_1)] \cap P_\Sigma^{-1}(s_2)$. Notice that, as $L(G_m^R) = L_m(G_m^R) = R[L(G_1)]$, then $L(V) = P_R^{-1}\{R[L(G_1)]\} \cap P_\Sigma^{-1}[L(G_2)]$ and $L_m(V) = P_R^{-1}\{R[L(G_1)]\} \cap P_\Sigma^{-1}[L_m(G_2)]$. Initially, consider the case when

$\sigma \in \Sigma_o$. In this case, it can be seen that $s_v\sigma = s_vR(\sigma) \in P_R^{-1}[R(s_1)R(\sigma)] \cap P_\Sigma^{-1}(s_2\sigma)$. Therefore, as $s_1\sigma \in L(G_1)$, $s_2\sigma \in L_m(G_2)$, then $s_v\sigma = s_vR(\sigma) \in L_m(V)$. Consider, now, the case when $\sigma \in \Sigma_{uo}$. In this case, it is not difficult to see that $s_v\sigma \in P_R^{-1}[R(s_1)] \cap P_\Sigma^{-1}(s_2\sigma)$ and $s_vR(\sigma) \in P_R^{-1}[R(s_1)R(\sigma)] \cap P_\Sigma^{-1}(s_2)$. Therefore, as $s_1 \in L(G_1)$ and $s_2\sigma \in L_m(G_2)$, then $s_v\sigma \in L_m(V)$, and, as $s_1\sigma \in L(G_1)$ and $s_2 \in L(G_2)$, then $s_vR(\sigma) \in L(V)$.

Finally, since the first (resp. second) component of the states of V does not modify if a transition labeled by an event in Σ_{uo} (resp. $R(\Sigma_{uo})$) occurs and $P_o(s_1) = P_o(s_2)$, then $f_v(s_{0_v}, s_v) = (x_1, x_2)$. ■

Algorithm 3.1 (Verification of relative observability)

Inputs:

- $G = (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_{m_g})$*: automaton whose marked language is* $L_m(G)$*;*
- $A = (X_a, \Sigma, f_a, \Gamma_a, x_{0_a}, X_{m_a})$*: nonblocking automaton whose marked language is* C*;*
- $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$*: nonblocking automaton whose marked language is* K*.*

Output:

- K *is relatively observable wrt* $\overline{C}$*,* G *and* P_o*: true/false.*

Step 1: Compute automaton G_m *by marking all states of* G*, i.e.* $G_m := (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_g)$.

Step 2: From automaton A*, construct automaton* $M := (X_a \cup \{x_d\}, \Sigma, f_m, \Gamma_m, x_{0_a}, X_a \cup \{x_d\})$*, where (i)* $\Gamma_m(x) = \Sigma, \forall x \in X_a$*, and* $\Gamma_m(x_d) = \emptyset$*; and (ii)* $\forall (x, \sigma) \in X_a \times \Sigma$*,* $f_m(x, \sigma) = f_a(x, \sigma)$*, if* $\sigma \in \Gamma_a(x)$*, and* $f_m(x, \sigma) = x_d$*, otherwise.*

Step 3: Compute automaton $M_g := M \times G_m$.

Step 4: From automaton H*, construct automaton* $N := (X_h \cup \{x_{d1}, x_{d2}\}, \Sigma, f_n, \Gamma_n, x_{0_h}, \{x_{d1}, x_{d2}\})$*, where (i)* $\Gamma_n(x) = \Sigma$*,* $\forall x \in X_h \cup \{x_{d1}\}$*, and* $\Gamma_n(x_{d2}) = \emptyset$*; (ii)* $\forall (x, \sigma) \in X_h \times \Sigma$*:* $f_n(x, \sigma) = f_h(x, \sigma)$*, if* $\sigma \in \Gamma_h(x)$*,* $f_n(x, \sigma) = x_{d1}$*, if* $\sigma \in (\Sigma_{uo} \setminus \Gamma_h(x))$*, and* $f_n(x, \sigma) = x_{d2}$*, if* $\sigma \in (\Sigma_o \setminus \Gamma_h(x))$*; and (iii)* $f_n(x_{d1}, \sigma) = x_{d1}$*, if* $\sigma \in \Sigma_{uo}$ *and* $f_n(x_{d1}, \sigma) = x_{d2}$*, if* $\sigma \in \Sigma_o$.

Step 5: Compute automaton $H_c := CoAc(M_g \times N)$.

Step 6: Construct automaton $H_m^R := (X_h, R(\Sigma), f_R, \Gamma_R, x_{0_h}, X_h)$*, where* $\Gamma_R(x) = \Gamma_h(x)$ *and* $f_R(x, R(\sigma)) = f_h(x, \sigma)$*,* $\forall x \in X_h$ *and* $\forall \sigma \in \Sigma$.

Step 7: Compute the verifier automaton $V := H_m^R \| H_c = (X_v, \Sigma \cup \Sigma_R, f_v, \Gamma_v, x_{0_v}, X_{m_v})$.

Step 8: For all $(x, \sigma) \in X_v \times \Sigma$ *such that* $f_v(x, \sigma) \in X_{m_v}$*, verify if the following conditions hold true*

(a) $\sigma \in \Sigma_o$*;*

(b) $(\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))$.

If there exists a transition $f_v(x, \sigma)$ *such that either condition (a) or (b) holds, then* K *is not relatively observable with respect to* $\overline{C}$*,* G *and* P_o*. Otherwise,* K *is relatively observable with respect to* $\overline{C}$*,* G *and* P_o*.*

In Step 1 of Algorithm 3.1, we obtain automaton G_m from G that marks the language generated by G, *i.e.*, $L_m(G_m) = L(G)$. In Step 2, we construct automaton M from the nonblocking automaton A, whose marked language is $L_m(M) = \overline{C}\Sigma \cup \{\varepsilon\}$. In Step 3, we build automaton $M_g = M \times G_m$ that marks language $(\overline{C}\Sigma \cup \{\varepsilon\}) \cap L(G)$. In Step 4, we construct automaton N from the nonblocking automaton H for which $L_m(N) = \overline{K}\Sigma_{uo}^*\Sigma \cap \overline{K}^C$. In Step 5, we obtain automaton $H_c = M_g \times N$, which has the following property.

Lemma 3.3 $L_m(H_c) = (\overline{K}\Sigma_{uo}^* \cap \overline{C})\Sigma \cap \overline{K}^C \cap L(G)$.

Proof: The proof is done in three steps as follows:

1st Step: $L_m(M) = \{\varepsilon\} \cup \overline{C}\Sigma$. Notice that, automaton M is constructed from automaton A by creating new transitions labeled by the non active events connecting each state of A to the dump state, x_d, and by marking all states of M. Then, $\forall s \in \Sigma^*$, $s \in \overline{C} \Leftrightarrow f_m(x_{0_a}, s) \in X_a$. It is then straightforward to see from the construction of M, that every string $s \in \{\varepsilon\} \cup \overline{C}\Sigma$ also belongs to $L_m(M)$. Take, now, a string $s \in L_m(M)$. If $s = \varepsilon$, then $s \in \{\varepsilon\} \cup \overline{C}\Sigma$. If $s \neq \varepsilon$, then it is possible to write s as $s = s_p\sigma$ where $s_p \in \Sigma^*$ and $\sigma \in \Sigma$. From the construction of M, $f_m(x_d, \sigma)$ is undefined, and thus, $x = f_m(x_{0_a}, s_p) \in X_a$, which ultimately implies that $s = s_p\sigma \in \overline{C}\Sigma$.

2nd Step: $L(N) = \overline{K}\Sigma_{uo}^*\Sigma \cap \overline{K}^C$. Notice that we construct automaton N from the nonblocking automaton H, by creating two new states, x_{d1} and x_{d2}, making the set of marked states equal to $\{x_{d1}, x_{d2}\}$, and adding new transitions labeled by the non active unobservable events connecting each original state of H to state x_{d1}, and

transitions labeled by the non active observable events to state x_{d2}, and create self-loops in state x_{d_1} labeled with all unobservable events, and transitions from state x_{d_1} to state x_{d2}, labeled with all observable events. Since $L(H) = \overline{K}$, then, $\forall s \in \Sigma^*$, $s \in \overline{K} \Leftrightarrow f_n(x_{0_h}, s) \in X_h$, and, consequently, every string $s \in L_m(N)$ also belongs to $\overline{K}^C$. In addition, from the construction of N, we can conclude that: *(i)* if s reaches state x_{d1}, it can be written as $s = s_p s_{uo}$, where s_p is the largest prefix of s in $\overline{K}$ and $s_{uo} \in \Sigma_{uo}^* \setminus \{\varepsilon\}$; *(ii)* if s reaches state x_{d2}, it can be written as $s = s_p s_{uo} \sigma_o$, where s_p is the largest prefix of s in $\overline{K}$, $s_{uo} \in \Sigma_{uo}^*$ and $\sigma_o \in \Sigma_o$. In both situations described above, we have that $s \in \overline{K}\Sigma_{uo}^*\Sigma$. Let, now, $s \in \overline{K}\Sigma_{uo}^*\Sigma \cap \overline{K}^C$. Then s can be written as $s = s_p s_u \sigma$, where $s_p \in \overline{K}$, $s_u \in \Sigma_{uo}^*$ and $\sigma \in \Sigma$. Write s_u as $s_u = s_{u1} s_{u2}$ where s_{u1} is the largest string in Σ_{uo}^* such that $s_p s_{u1} \in \overline{K}$ and $s_{u2} \in \Sigma_{uo}^*$. Let $x' = f_n(x_{0_h}, s_p s_{u1})$. Thus $x' \in X_h$. In addition, from the construction of N, $f_n(x', s_{u2}) = x_{d1}$, and, either $f_n(x', s_{u2}\sigma) = x_{d1}$, if $\sigma \in \Sigma_{uo}$, or $f_n(x', s_{u2}\sigma) = x_{d2}$, if $\sigma \in \Sigma_o$, which implies that $s \in L_m(N)$.

3rd Step: Since $H_c = CoAc(M_g \times N) = CoAc((M \times G_m) \times N)$, then $L_m(H_c) = L_m(M) \cap L(G) \cap L_m(N)$. Therefore, $L_m(H_c) = (\overline{C}\Sigma \cup \{\varepsilon\}) \cap L(G) \cap \overline{K}\Sigma_{uo}^*\Sigma \cap \overline{K}^C = \overline{C}\Sigma \cap L(G) \cap \overline{K}\Sigma_{uo}^*\Sigma \cap \overline{K}^C = (\overline{K}\Sigma_{uo}^* \cap \overline{C})\Sigma \cap \overline{K}^C \cap L(G)$, since for two languages L_1 and L_2 and an event set Σ, $L_1\Sigma \cap L_2\Sigma = (L_1 \cap L_2)\Sigma$. ■

In Step 6, we compute automaton H_m^R by applying the renaming function to the events of H and marking all of its states. In Steps 7 and 8 we construct the verifier automaton $V = H_m^R \| H_c$, and check if K is $\overline{C}$-observable. The following theorem demonstrates the correctness of Algorithm 3.1.

Theorem 3.1 *Let H, A and G denote the automata whose marked languages are, respectively, K, C and $L_m(G)$ such that $K \subseteq C \subseteq L_m(G)$, and consider the verifier automaton $V = (X_v, \Sigma \cup \Sigma_R, f_v, \Gamma_v, x_{0_v}, X_{m_v})$ computed using Algorithm 3.1 with the inputs G, A and H. Then, K is not relatively observable with respect to $\overline{C}$, G and P_o if and only if there exists $(x, \sigma) \in X_v \times \Sigma$ that satisfies $f_v(x, \sigma) \in X_{m_v}$ with either $(\sigma \in \Sigma_o)$ or $[(\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))]$.*

Proof: ($\Rightarrow$) Assume that K is not $\overline{C}$-observable wrt G and P_o. Then, according to Lemma 3.1, K is not $(\overline{K}\Sigma_{uo}^* \cap \overline{C})$-observable wrt G and P_o. Therefore, there exist $s \in \overline{K}$, $s' \in (\overline{K}\Sigma_{uo}^* \cap \overline{C})$, and $\sigma \in \Sigma$, such that $P_o(s) = P_o(s')$, $s\sigma \in \overline{K}$ and $s'\sigma \in L(G) \setminus \overline{K}$.

Notice that $s\sigma \in L(H)$, since $L(H) = \overline{K}$. On the other hand, since $s' \in (\overline{K}\Sigma_{uo}^* \cap \overline{C})$ and $s'\sigma \in L(G) \setminus \overline{K}$, it can be concluded that $s'\sigma \in (\overline{C} \cap \overline{K}\Sigma_{uo}^*)\Sigma \cap \overline{K}^C \cap L(G) = L_m(H_c)$. Therefore, from the construction of verifier automaton V and according to Lemma 3.2, there exists a pair of strings $(s_v\sigma, s_vR(\sigma)) \in L_m(V) \times L(V)$. Finally, defining $x = f_v(x_{0_v}, s_v)$, then $f_v(x, \sigma) \in X_{m_v}$. Moreover, since $s_vR(\sigma) \in L(V)$ and $R(\sigma) \in \Gamma_v(x)$, it is possible to conclude that either $(\sigma \in \Sigma_o)$ or $[(\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))]$.

($\Leftarrow$) Assume, now, that there exists $(x, \sigma) \in X_v \times \Sigma$ such that $f_v(x, \sigma) \in X_{m_v}$ and $(\sigma \in \Sigma_o) \vee [(\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))]$. Then, from the construction of automaton V, there exists $s_v \in L(V)$ such that $f_v(x_{0_v}, s_v) = x$ and $s_v\sigma \in L_m(V)$. Moreover, since $R(\sigma) = \sigma$, if $\sigma \in \Sigma_o$, and $R(\sigma) \in \Gamma_v(x)$, if $\sigma \notin \Sigma_o$, then $s_vR(\sigma) \in L(V)$. Consequently, according to Lemma 3.2, there exists $(s\sigma, s'\sigma) \in L(H) \times L_m(H_c)$ such that $P_o(s) = P_o(s')$. Notice that, $s\sigma \in \overline{K}$, $s' \in (\overline{K}\Sigma_{uo}^* \cap \overline{C})$, and $s'\sigma \in L(G) \setminus \overline{K}$, which implies that K is not $(\overline{K}\Sigma_{uo}^* \cap \overline{C})$-observable with respect to G and P_o. ■

Remark 3.1 *(Verification of language observability) Algorithm 3.1 can be also applied to verify if a language K, marked by a nonblocking automaton H, is observable with respect to G and P_o, just by making $A = H$.*

Remark 3.2 *As it will be shown in Section 3.3, the use of the reduced ambient language is crucial for the application of Algorithm 3.1 in the computation of the supremal $\overline{C}$-observable sublanguage of a language K. However, when we want only to verify if a language K is $\overline{C}$-observable (or observable, by making $C = K$), we can use ambient language $\overline{C}$ instead of $\overline{C_s}$, and thus, Step 4 of Algorithm 3.1 should be modified so as to compute automaton N such that $L_m(N) = \overline{K}^C$. However, this change does not improve the computational complexity of the algorithm, since the original and the modified versions of automaton N have $|X_h|+2$ and $|X_h|+1$ states, respectively, but with the same number of transition $(|X_h| + 1)|\Sigma|$.*

The following example illustrates Algorithm 3.1.

Example 3.1 *Consider automata G, A and H depicted in Figure 3.1, where $\Sigma = \{\alpha, \beta, \sigma, \mu\}$ and $\Sigma_o = \{\alpha, \beta, \sigma\}$. Automaton M, constructed in Step 2, and automaton N, constructed in Step 4, are shown in Figures 3.2(a) and 3.2(b), respectively. Automata H_c and H_m^R obtained in Steps 5 and 6, respectively, are depicted*

in Figures 3.2(c,d). Finally, the verifier automaton V computed in Step 7 is depicted in Figure 3.2(e), from where, it can be checked that transition $((1,0),\alpha,(2,1))$ satisfies condition (a) of Step 8. Therefore, K is not $\overline{C}$-observable with respect to G and P_o. Notice that, since $(2,1) \in X_{m_v}$, the following strings can be obtained from V: $s = \mu \in \overline{K}$, $s' = \varepsilon \in \overline{C}$, $s\alpha = \mu\alpha \in \overline{K}$ and $s'\alpha = \alpha \in L(G) \setminus \overline{K}$, and $P_o(s) = P_o(s') = \varepsilon$.

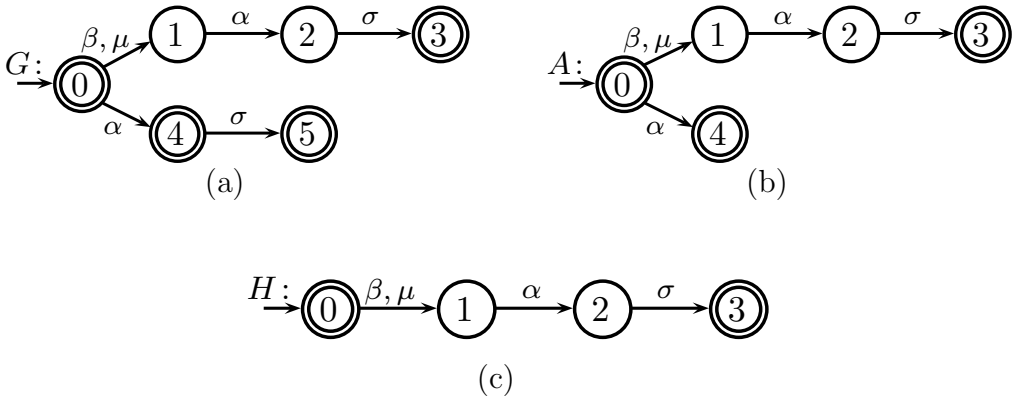


Figure 3.1: System automaton G (a), automaton A that marks ambient language C (b), and automaton H whose marked language is K (c).

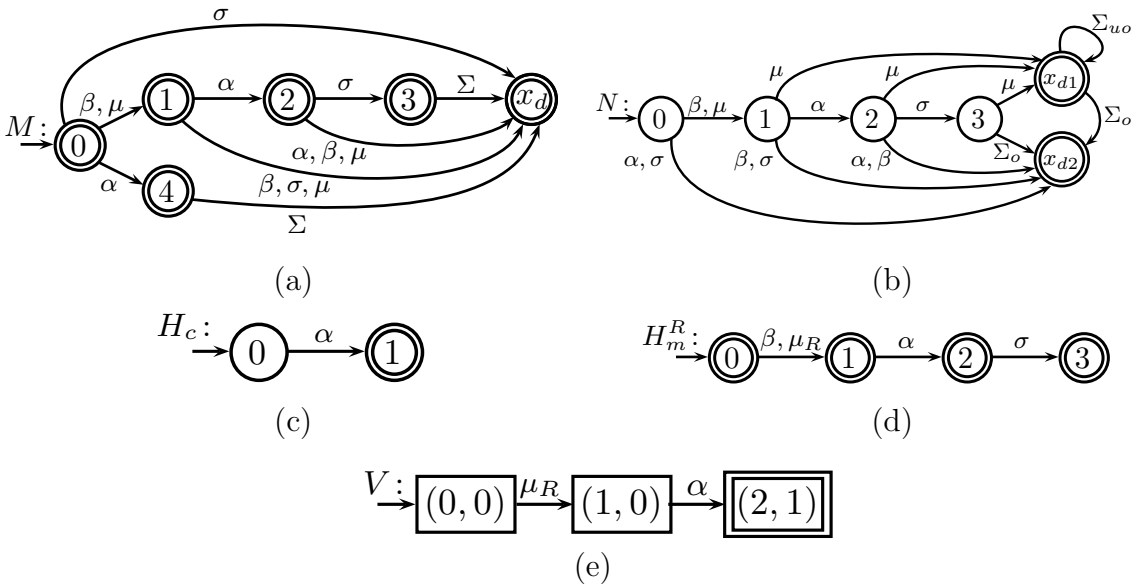


Figure 3.2: Automata obtained in Example 3.1 by Algorithm 3.1: M (a), N (b), H_c (c), H_m^R (d) and V (e).

3.3 Computation of the Supremal Relatively Observable Sublanguage

Consider nonblocking automata A and H such that $L_m(A) = C$ and $L_m(H) = K$, and let f_g, f_a and f_h (resp. x_{0_g}, x_{0_a} and x_{0_h}) denote the transition functions (resp. initial states) of automata G, A and H. We make the following assumptions.

A1. For all $s, s' \in L(A)$ such that $f_a(x_{0_a}, s) = f_a(x_{0_a}, s')$, then $f_g(x_{0_g}, s) = f_g(x_{0_g}, s')$;

A2. For all $s, s' \in L(H)$ such that $f_h(x_{0_h}, s) = f_h(x_{0_h}, s')$, then $f_a(x_{0_a}, s) = f_a(x_{0_a}, s')$.

Notice that, if A (resp. H) does not satisfy Assumption **A1** (resp. **A2**), then another automaton A (resp. H) that satisfies Assumption **A1** (resp. **A2**) can be obtained by computing the completely synchronous composition $A \times G$ (resp. $H \times A$). It is worth remarking that Assumptions **A1** and **A2** are less restrictive than assuming that H and A are subautomata of G. Finally, when $K = C$, Assumptions **A1** and **A2** reduces to a single one, equivalent to that made in [15].

We will propose an algorithm to obtain a deterministic automaton that marks the supremal $\overline{C}$-observable sublanguage of K with respect to G and P_o, whose main idea is to remove, from an automaton that marks K, all transitions that correspond to transitions in verifier automaton V that violate the relative observability condition according to Theorem 3.1 and Algorithm 3.1. It is worth remarking that not every automaton H has the correct structure to prevent that other strings, besides those which actually violates the relative observability condition, are removed from K. This fact is illustrated by the following example.

Example 3.2 *Let us consider languages $L(G)$, C and K, generated, respectively, by automata G, A and H of Example 3.1, shown in Figures 3.1(a,b,c). In verifier automaton V computed in Example 3.1, and depicted in Figure 3.2(e), transition $((1,0), \alpha, (2,1))$ satisfies condition (a) of Step 8 of Algorithm 3.1. This is due to string $s = \mu \in \overline{K}$ that violates the relative observability since $s' = \varepsilon \in (\overline{K}\Sigma_{uo}^* \cap \overline{C})$, $P_o(s) = P(s')$ but $s\alpha \in \overline{K}$ and $s'\alpha \in L(G) \setminus \overline{K}$. However, if we remove transition $(1, \alpha, 2)$ of H (Figure 3.1(c)), we not only remove string $\mu\alpha$, but also exclude string*

$\beta\alpha$ which must not be removed. In such case, the computed supremal relatively observable sublanguage will be ε, which is incorrect. However, if we use the state partition automaton $H_{sp} = H\|Obs(H, \Sigma_o)$, depicted in Figure 3.3, we successfully eliminate string $\mu\alpha$ and preserve string $\beta\alpha$ when we exclude transition $(1', \alpha, 2)$; therefore leading to the correct supremal relatively observable sublanguage.

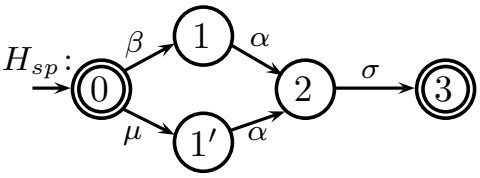


Figure 3.3: Automaton $H_{sp} = H\|Obs(H, \Sigma_o)$ obtained from the automaton H depicted in Figure 3.1(c).

Let us define the state partition automaton $H_{sp} = H\|obs(H, \Sigma_o)$. It is not difficult to see that $L(H_{sp}) = L(H) = \overline{K}$, $L_m(H_{sp}) = L_m(H) = K$, and that H_{sp} also satisfies Assumption **A2**. Let us define the *uncertainty set* [15] of an automaton H after the occurrence of a string $s \in L(H)$ as: $U_h(s) := \{x \in X_h : (\exists s' \in L(H))[(x = f_h(x_{0,h}, s')) \wedge (P_o(s') = P_o(s))]\}$. The following results can be stated.

Lemma 3.4 *Let $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$ and $H_{sp} = H\|Obs(H, \Sigma_o) = (X_{sp}, \Sigma, f_{sp}, \Gamma_{sp}, x_{0_{sp}}, X_{m_{sp}})$. Then, $f_{sp}(x_{0_{sp}}, s) = (f_h(x_{0_h}, s), U_h(s))$, $\forall s \in L(H_{sp})$.*

Proof: In accordance with the construction of H_{sp}, $f_{sp}(x_{0_{sp}}, s) = (f_h(x_{0_h}, s), f_{obs}(x_{0_{obs}}, P_o(s)))$, for all $s \in L(H_{sp})$, where $x_{0_{obs}}$ and f_{obs} denote the initial state and the transition function of $Obs(H, \Sigma_o)$, respectively. Thus, using the definition of the estimate of possible states of H after string s proposed in [86], which is equivalent to the definition of uncertainty set $U_h(s)$, we conclude that $f_{obs}(x_{0_{obs}}, P_o(s)) = U_h(s)$. ■

Lemma 3.5 *Let $H = (X_h, \Sigma, f_h, x_{0_h}, \Gamma_h, X_{m_h})$, $H_{sp} = H\|Obs(H, \Sigma_o) = (X_{sp}, \Sigma, f_{sp}, \Gamma_{sp}, x_{0_{sp}}, X_{m_{sp}})$ and $H_s = (X_s, \Sigma, f_s, \Gamma_s, x_{0_s}, X_{m_s})$ such that $H_s \sqsubseteq H_{sp}$ and $L_m(H_s) = K_s \subseteq K$. Assume that there exist $s, s' \in \overline{K_s}$ and $\sigma \in \Sigma$ such that $s\sigma, s'\sigma \in \overline{K_s}$ and $f_s(x_{0_s}, s) = f_s(x_{0_s}, s')$. If $\exists s'' \in (\overline{K_s}\Sigma_{uo}^* \cap \overline{C})$ such that $P_o(s'') = P_o(s')$ and $s''\sigma \in L(G) \setminus \overline{K_s}$, then $\exists s_c \in \overline{C}$ such that $P_o(s_c) = P_o(s)$ and $s_c\sigma \in L(G) \setminus \overline{K_s}$.*

Proof: Assume that there exist $s, s' \in \overline{K_s}$, $\sigma \in \Sigma$ and $s'' \in (\overline{K_s}\Sigma_{uo}^* \cap \overline{C})$ such that $s\sigma, s'\sigma \in \overline{K_s}$, $f_s(x_{0_s}, s) = f_s(x_{0_s}, s')$, $P_o(s'') = P_o(s')$ and $s''\sigma \in L(G) \setminus \overline{K_s}$.

Without loss of generality, write s'' as $s'' = s_p''s_s''$, where s_p'' is the longest prefix of s'' in $\overline{K_s}$ and $s_s'' \in \Sigma_{uo}^*$. Since $f_s(x_{0_s}, s) = f_s(x_{0_s}, s')$, according to Lemma 3.4, $U_h(s) = U_h(s') = U_h(s_p'')$, where the last equality is a consequence of the fact that $P_o(s') = P_o(s'') = P_o(s_p'')$. According to the definition of uncertainty set, $f_h(x_{0_h}, s_p'') \in U_h(s_p'')$, and, since $U_h(s_p'') = U_h(s)$, there exists $s_{cp} \in \overline{K}$ such that $P_o(s_{cp}) = P_o(s)$ and $f_h(x_{0_h}, s_{cp}) = f_h(x_{0_h}, s_p'')$. Thus, using Lemma 3.4, $f_{sp}(x_{0_{sp}}, s_{cp}) = (f_h(x_{0_h}, s_p''), U_h(s_p''))$, *i.e.*, strings s_p'' and s_{cp} reach the same state of H_{sp} and also H_s, which implies that these strings are continued in H_s with the same strings. Therefore, $s_{cp}s_s''\sigma \notin \overline{K_s}$, since s_p'' is not continued with $s_s''\sigma$ in $\overline{K_s}$. Moreover, according to Assumptions **A2** and **A1**, since s_p'' and s_{cp} reach the same state of H, they also reach the same states of A and G, which, together with the fact that $s_p''s_s'' \in \overline{C}$ and $s_p''s_s''\sigma \in L(G)$, imply, respectively, that $s_{cp}s_s'' \in \overline{C}$ and $s_{cp}s_s''\sigma \in L(G)$. Finally, defining $s_c = s_{cp}s_s''$, we have that $s_c \in \overline{C}$, $P_o(s) = P_o(s_c)$ and $s_c\sigma \in L(G) \setminus \overline{K_s}$. ■

Lemma 3.5 shows that, when H, A and G satisfy Assumptions **A1** and **A2**, then for an automaton H_s ($H_s \sqsubseteq H_{sp} = H \| Obs(H, \Sigma_o)$) where $L_m(H_s) = K_s$, if a string $s'\sigma \in \overline{K_s}$ violates $(\overline{K_s}\Sigma_{uo}^* \cap \overline{C})$-observability, then all strings $s \in \overline{K_s}$ that reach the state of H_s reached by s' are such that $s\sigma$ also violates $\overline{C}$-observability. As a consequence, if we remove transitions of H_s associated with the strings of $\overline{K_s}$ that violate $(\overline{K_s}\Sigma_{uo}^* \cap \overline{C})$-observability, we only eliminate from $\overline{K_s}$ those strings that violate $\overline{C}$-observability. This fact suggests the following algorithm for computing the supremal $\overline{C}$-observable sublanguage of a language K.

Algorithm 3.2 (Computation of the supremal relatively observable sublanguage)

Inputs:

- $G = (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_{m_g})$*: automaton whose marked language is* $L_m(G)$*;*
- $A = (X_a, \Sigma, f_a, \Gamma_a, x_{0_a}, X_{m_a})$*: nonblocking automaton whose marked language is* C*;*

• $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$*: nonblocking automaton whose marked language is* K.

Output:

• H_{sup}*: nonblocking automaton whose marked language is the supremal* $\overline{C}$*-observable sublanguage of* K *with respect to* G *and* P_o.

Step 1: Compute $H_{sp} := H \| Obs(H, \Sigma_o) = (X_{sp}, \Sigma, f_{sp}, \Gamma_{sp}, x_{0_{sp}}, X_{m_{sp}})$;

Step 2: Set $H_s = H_{sp}$;

Step 3: Compute verifier automaton $V = (X_v, \Sigma \cup \Sigma_R, f_v, \Gamma_v, x_{0_v}, X_{m_v})$, *by using Algorithm 3.1 with inputs* G, A *and* H_s;

Step 4: If V *is not an empty automaton, then form the following set:*

$$\begin{aligned} X\Sigma \;\; = \;\; & \{(x_v, \sigma) \in X_v \times \Sigma : (f_v(x_v, \sigma) \in X_{m_v}) \;\wedge \\ & ((\sigma \in \Sigma_o) \vee ((\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))))\}. \end{aligned}$$

Step 5: If $X\Sigma \neq \emptyset$, *then:*

- *5.1: For all* $(x_v, \sigma) \in X\Sigma$, *exclude from* H_s *transition* $(x, \sigma, f_s(x, \sigma))$, *where* x *is the state of* H_s *equal to the first component of* x_v;
- *5.2:* $H_s \leftarrow Trim(H_s)$;
- *5.3: Return to Step 3;*

Step 6: $H_{sup} \leftarrow H_s$.

Notice that in Algorithm 3.2, after the computation of the state partition automaton H_{sp}, we execute Steps 3 to 5 iteratively. For each iteration, we compute the verifier automaton V in Step 3 by using Algorithm 3.1 with the inputs G, A and H_s. In Step 4, we form set $X\Sigma$, which represents all pairs (x_v, σ), $x_v \in X_v$ and $\sigma \in \Sigma$, responsible for the loss of relative observability according to Theorem 3.1. Notice that, when $X\Sigma = \emptyset$, language $L_m(H_s)$ is $\overline{C}$-observable with respect to G and P_o. If $X\Sigma \neq \emptyset$, then, according to Lemma 3.2, for each $(x_v, \sigma) \in X\Sigma$, there exists a string s that reaches state x, equal to the first component of x_v, and is continued by σ such that $s\sigma \in L(H_s)$ violates $\overline{C}$-observability; in Step 5.1, we remove transition $(x, \sigma, f_s(x, \sigma))$ of H_s and, in order to remove possible non-accessible and/or non-coaccessible states, we apply $Trim()$ operator in Step 5.2. When we remove

transitions in Step 5.1, it is necessary to verify if the language marked by the new H_s is $\overline{C}$-observable, therefore, after carrying out Step 5.2 we return to Step 3. Notice also that, at each iteration of Algorithm 3.2, we remove at least one transition from automaton $H_s \sqsubseteq H_{sp}$, which implies that the number of iterations is at most equal to the number of transitions of automaton H_{sp}. Therefore, Algorithm 3.2 terminates in finite steps.

Theorem 3.2 *Consider automaton G and nonblocking automata A and H such that $L_m(A) = C$, $L_m(H) = K$ and $K \subseteq C \subseteq L_m(G)$, and assume that automata G, A and H satisfy Assumptions **A1** and **A2**. Then, automaton H_{sup} obtained by Algorithm 3.2 with inputs G, A and H marks the supremal $\overline{C}$-observable sublanguage of K with respect to G and P_o.*

Proof: Let K'_{sup} denote the supremal $\overline{C}$-observable sublanguage of K with respect to G and P_o. Algorithm 3.2 finishes when $X\Sigma = \emptyset$. Therefore, according to Theorem 3.1, $L_m(H_{sup})$ is $\overline{C}$-observable with respect to G and P_o, which implies that $L_m(H_{sup}) \subseteq K'_{sup}$. Then, we only need to prove that $K'_{sup} \subseteq L_m(H_{sup})$. The proof will be done using mathematical induction over the (finite) set of iterations in Algorithm 3.2.

Basis step. At the beginning of the first iteration, $H_s = H_{sp}$. Therefore, $K'_{sup} \subseteq L_m(H_s) = K$;

Induction hypothesis. Suppose that $K'_{sup} \subseteq L_m(H_s)$, up to the beginning of the i-th iteration;

Inductive step. Let us now consider the (i+1)-st iteration. Notice that in the i-th iteration, we have removed transitions from H_s in Steps 5.1 and 5.2. Since *Trim* operator applied in Step 5.2 does not modify the marked language of an automaton, strings are only removed from $L_m(H_s)$ in Step 5.1. Thus we need only to analyze Step 5.1.

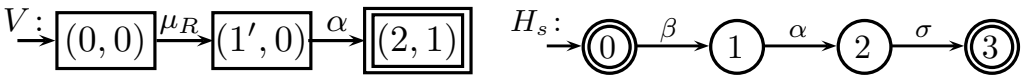
By using Lemma 3.5, it can be concluded that for each string $s_m \in L_m(H_s)$ that is removed from $L_m(H_s)$ in Step 5.1, there exist $s \in \overline{\{s_m\}}$, $s' \in \overline{C}$ and $\sigma \in \Sigma$ such that $s\sigma \in \overline{\{s_m\}}$, $s'\sigma \in L(G) \setminus L(H_s)$ and $P_o(s) = P_o(s')$, *i.e.*, $s\sigma$ violates $\overline{C}$-observability. In accordance with the induction hypothesis, $K'_{sup} \subseteq L_m(H_s)$ at the beginning of the i-th iteration, and thus, at the beginning of the i-th, $(L(G) \setminus L(H_s)) \subseteq (L(G) \setminus \overline{K'_{sup}})$, which implies that $s'\sigma \in L(G) \setminus \overline{K'_{sup}}$, and thus, since K'_{sup} is $\overline{C}$-observable, $s\sigma \notin$

$\overline{K'_{sup}}$, we conclude that $s_m \notin K'_{sup}$. Therefore, all string s_m removed from $L_m(H_s)$ does not belong to K'_{sup},which implies that $K'_{sup} \subseteq L_m(H_s)$ at the beginning of the (i+1)-st iteration.

Finally, since H_{sup} is equal to the H_s obtained in the last iteration of Algorithm 3.2, then $K'_{sup} \subseteq L_m(H_{sup})$, which concludes the proof. ■

We will now illustrate the application of Algorithm 3.2.

Example 3.3 *Let us consider automata G, A and H of Example 3.1, shown in Figures 3.1(a,b,c). When we apply Algorithm 3.2 with inputs G, A and H, we obtain, in Step 1, automaton H_{sp}, which is depicted in Figure 3.3, and, in the first iteration, we obtain the verifier automaton V depicted in Figure 3.4(a). Therefore, according to Step 4, we obtain $X\Sigma = \{((1',0),\alpha)\} \neq \emptyset$. This implies that Steps 5.1 to 5.3 must be performed: in Step 5.1 we remove transition $(1',\alpha,2)$ of H_s ($H_s = H_{sp}$), because this transition is associated with string $\mu\alpha$ that violates $\overline{C}$-observability. In Step 5.3(a), we exclude state $1'$ of H_s by applying the CoAc operation, because this state becomes a blocking one after the exclusion of transition $(1',\alpha,2)$. Finally, at the end of the first iteration, we obtain automaton H_s depicted in Figure 3.4(b). In the second iteration, we obtain $X\Sigma = \emptyset$. Therefore automaton H_{sup} is equal to automaton H_s, shown in Figure 3.4(b), and marks the supremal $\overline{C}$-observable sublanguage of K with respect to G and P_o.*



(a) (b)

Figure 3.4: Automata obtained in the first iteration of Algorithm 3.2: V (a) and H_s (b).

Remark 3.3 *It could be argued that instead of using $\overline{C_s} = \overline{K}\Sigma_{uo}^* \cap \overline{C}$, we could use $\overline{C}$ to compute V in order to identify the transitions that must be removed. However, in doing so, we could eliminate strings that do not violate $\overline{C}$-observability. In order to illustrate this fact consider verifier V_c, shown in Figure 3.5, computed by using, in Algorithm 3.1, $\overline{C}$ in the place of $\overline{C_s}$. Notice that, transition $((2,1),\sigma,(3,2))$ of V_c satisfies condition (a) in Step 8 of Algorithm 3.1, since string $s = \mu\alpha \in \overline{K}$ is such that $s\sigma \in \overline{K}$ violates the $\overline{C}$-observability, because $s' = \alpha \in \overline{C}$, $s'\sigma \in L(G) \setminus \overline{K}$*

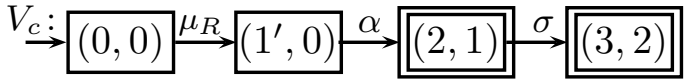


Figure 3.5: Verifier obtained with Algorithm 3.1 by using $\overline{C}$ instead of $\overline{C_s}$.

and $P_o(s) = P_o(s')$. However, if we remove transition $(2, \sigma, 3)$ from H_{sp} (depicted in Figure 3.3), we also eliminate string $\beta\alpha\sigma$, which must remain. On the other hand, if we use the reduced ambient language $\overline{C_s}$, string $s\sigma$ does not violate $\overline{C_s}$-observability, but its prefix s does, which leads to the exclusion of transition $(1', \alpha, 2)$, and consequently the removal of string $s\sigma$ from K, as expected from the proof of Lemma 3.1.

3.4 Computation of Controllable and Observable Sublanguages by Using Relative Observability Property

Consider a system modeled by an automaton G whose set of events is $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} = \Sigma_o \dot{\cup} \Sigma_{uo}$, and let K be a language such that $K \subseteq L_m(G)$. In this section, we will present an algorithm for the computation of a controllable and observable sublanguage of K by using the concept of relative observability. This algorithm follows the procedure presented in Subsection 2.5.3 for the computation of the supremal controllable and $\overline{C}$-observable sublanguage of K with respect to a previously defined ambient language $\overline{C}$. However, in the algorithm we will propose here, we iteratively shrink the ambient language, as done in [15], with a view to obtaining a more permissive controllable and observable sublanguage. The difference between the algorithm proposed by CAI *et al.* [15] and our algorithm is that the former only shrink the ambient language at the beginning of each computation of a relatively observable sublanguage, whereas in our algorithm, we shrink the ambient language at the beginning and during each computation of a relatively observable sublanguage.

Algorithm 3.3 (Computation of $K^{\uparrow CRO}$)

Inputs:

- $G = (X_g, \Sigma, f_g, \Gamma_g, x_{0_g}, X_{m_g})$*: automaton whose marked language is* $L_m(G)$*;*

• $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h}, X_{m_h})$: *nonblocking automaton whose marked language is* K.

Output:

• H_{CRO}: *nonblocking automaton whose marked language is a controllable (with respect to* $L(G)$ *and* Σ_{uc}*) and observable (with respect to* G *and* P_o*) sublanguage of* K.

Step 1: Compute $H_{sp} = H \| Obs(H, \Sigma_o)$;

Step 2: Set $H_{sr} := H_{sp} \times G = (X_{sr}, \Sigma, f_{sr}, \Gamma_{sr}, x_{0_{sr}}, X_{m_{sr}})$;

Step 3: Form the following set of states:

$$X_{cont} = \{((x_h, x_{obs}), x_g) \in X_{sr} : \Gamma_g(x_g) \cap \Sigma_{uc} \subseteq \Gamma_{sr}(((x_h, x_{obs}), x_g))\}.$$

Step 4: If $(X_s \setminus X_{cont}) \neq \emptyset$*, then:*

- *4.1: Define* $X'_{sr} = X_{cont}$, $f'_{sr} = f_{sr}|_{X_{cont}}$, $\Gamma'_{sr} = \Gamma_{sr}|_{X_{cont}}$ *and* $X'_{m_{sr}} = X_{m_{sr}} \cap X_{cont}$;

- *4.2: Set* $H_{sr} = Trim(X'_{sr}, \Sigma, f'_{sr}, \Gamma'_{sr}, x_{0_{sr}}, X'_{m_{sr}})$*, and, if* $x_{0_{sr}}$ *is deleted in the previous calculation, then make* H_{CRO} *be equal to an empty automaton and STOP the algorithm;*

- *4.3: Return to Step 3;*

Step 5: Compute verifier automaton $V = (X_v, \Sigma \cup \Sigma_R, f_v, \Gamma_v, x_{0_v}, X_{m_v})$*, by using Algorithm 3.1 with inputs* G, H_{sr} *and* H_{sr};

Step 6: If V *is not an empty automaton, then form the following set:*

$$\begin{aligned} X\Sigma \;\; = \;\; & \{(x_v, \sigma) \in X_v \times \Sigma : (f_v(x_v, \sigma) \in X_{m_v}) \;\wedge \\ & ((\sigma \in \Sigma_o) \vee ((\sigma \notin \Sigma_o) \wedge (R(\sigma) \in \Gamma_v(x))))\}. \end{aligned}$$

Step 7: If $X\Sigma \neq \emptyset$*, then:*

- *7.1: For all* $(x_v, \sigma) \in X\Sigma$*, exclude from* H_{sr} *transition* $(x, \sigma, f_s(x, \sigma))$*, where* x *is the state of* H_{sr} *equal to the first component of* x_v;

- *7.2:* $H_{sr} \leftarrow Trim(H_{sr})$;

- *7.3: Return to Step 3;*

Step 8: $H_{CRO} \leftarrow H_{sr}$.

In Step 1 of Algorithm 3.3, we compute the state partition automaton H_{sp} from H, and, in Step 2, we compute automaton $H_{sr} = H_{sp} \times G$ that refines G, *i.e.*, $\forall s_1, s_2 \in L(H_{sr})$, when $f_{sr}(x_{0_{sr}}, s_1) = f_{sr}(x_{0_{sr}}, s_2)$, then $f_g(x_{0_g}, s_1) = f_g(x_{0_g}, s_2)$. It can be checked that Lemma 3.5 still holds when we replace H_{sp} with $H_{sr} = H_{sp} \times G$. Steps 3 and 4 of Algorithm 3.3 correspond to the procedure proposed by [87] for the computation of the supremal controllable sublanguage of a regular language. Finally, Steps 5 to 7 are similar to Steps 3 to 5 of Algorithm 3.2, except for the following difference:

- When we remove some transition in Step 5 of Algorithm 3.2, we return to Step 3 with a view to checking if the new $L_m(H_s)$ has strings that violate the relative observability with respect to ambient language $\overline{L_m(A)}$;
- When we remove some transition in Step 7 of Algorithm 3.3, we return to Step 3 in order to compute the supremal controllable sublanguage of the new $L_m(H_s)$, which agrees with the fact that, since we only seek controllable sublanguages, we can reduce the ambient language to the supremal controllable sublanguage computed in Steps 3 and 4, and, thus, to achieve a more permissive sublanguage by applying this smaller ambient language in Step 5.

The following proposition demonstrates the correctness of Algorithm 3.3 and the improvement achieved by it in the language permissiveness. To this end, let $supCRO(K, \overline{K})$ denote the supremal controllable and $\overline{K}$-observable sublanguage of K, which can be obtained by applying Algorithm 3.2 in the procedure described in Subsection 2.5.3.

Proposition 3.1 *Consider automaton G with set of events $\Sigma = \Sigma_c \cup \Sigma_{uc} = \Sigma_o \cup \Sigma_{uo}$, $P_o : \Sigma^* \rightarrow \Sigma_o^*$, and nonblocking automaton H such that $L_m(H) = K$ and $K \subseteq L_m(G)$, and let H_{CRO} denote the automaton obtained by Algorithm 3.3 with inputs G and H. Then,* (i) *$L_m(H_{CRO})$ is controllable with respect to $L(G)$ and Σ_{uc} and observable with respect to $L(G)$ and P_o, and* (ii) *$supCRO(K, \overline{K}) \subseteq L_m(H_{CRO})$.*

Proof: (i) Algorithm 3.3 finishes when $((X_s \setminus X_{cont}) = \emptyset) \wedge (X\Sigma = \emptyset)$. Then, according to [87, Proposition 6.1], $L_m(H_{CRO})$ is controllable with respect to $L(G)$

and Σ_{uc} since $(X_s \setminus X_{cont}) = \emptyset$. In addition, according to Theorem 3.1, $X\Sigma = \emptyset$ implies that $L_m(H_{CRO})$ is $\overline{L_m(H_{CRO})}$-observable with respect to G and P_o, and, consequently, it is also observable with respect to $L(G)$ and P_o.

(ii) The proof will be done using mathematical induction over the (finite) set of iterations in Algorithm 3.3.

Basis step. At the beginning of the first iteration, $H_{sr} = H_{sp} \times G$. Therefore, $supCRO(\overline{K}, \overline{K}) \subseteq L_m(H_{sr}) = K$;

Induction hypothesis. Suppose that $supCRO(K, \overline{K}) \subseteq L_m(H_{sr})$, up to the beginning of the i-th iteration;

Inductive step. Let us now consider the (i+1)-st iteration. Notice that in the i-th iteration, we have removed states and/or transitions from H_{sr} either in Step 4 or in Step 7. According to [87, Proposition 6.1], every string s_m removed from $L_m(H_{sr})$ in Step 4 violates the controllability condition, that is, $\exists s \in \overline{\{s_m\}}$ such that $\{s\}\Sigma_{uc} \cap L(G) \not\subset \overline{L_m(H_{sr})}$. Thus, in accordance with the induction hypothesis, $\{s\}\Sigma_{uc} \cap L(G) \not\subset \overline{supCRO(K, \overline{K})}$, which implies that $s_m \notin supCRO(K, \overline{K})$ since $supCRO(K, \overline{K})$ is controllable. In addition, it can be concluded, by using Lemma 3.5, that for each string s_m removed from $L_m(H_{sr})$ in Step 7 of Algorithm 3.3, there exists $s \in \overline{\{s_m\}}$ that violates $\overline{L_m(H_{sr})}$-observability, that is, there exist $s' \in \overline{L_m(H_{sr})}$ and $\sigma \in \Sigma$ such that $s\sigma \in \overline{\{s_m\}}$, $s'\sigma \in L(G) \setminus \overline{L_m(H_{sr})}$ and $P_o(s) = P_o(s')$. As a consequence, $s' \in \overline{K}$ since $L_m(H_{sr}) \subseteq K$ and, in accordance with the induction hypothesis, $s'\sigma \in L(G) \setminus \overline{supCRO(K, \overline{K})}$, which implies that $s_m \notin supCRO(K, \overline{K})$ since $supCRO(K, \overline{K})$ is $\overline{K}$-observable.

Finally, since H_{CRO} is equal to the H_{sr} obtained in the last iteration of Algorithm 3.3, then $supCRO(K, \overline{K}) \subseteq L_m(H_{CRO})$, which concludes the proof. ■

3.5 Computational Complexity Analysis of The Proposed Algorithms

3.5.1 Computational Complexity of Algorithm 3.1

Steps 1 to 5 of Algorithm 3.1 are employed to construct automaton H_c by the product composition of N, M and G_m that have $(|X_h| + 2)$, $(|X_a| + 1)$ and $|X_g|$

states, respectively. Then, H_c has $(|X_h| + 2) \cdot (|X_a| + 1) \cdot |X_g|$ states at most. Since $V = H_m^R \| H_c$ and H_m^R has $|X_h|$ states, then V has, at most, $|X_h| \cdot (|X_h| + 2) \cdot (|X_a| + 1) \cdot |X_g|$ states. The search for transitions of V executed in Step 8 of Algorithm 3.1 can be done with linear complexity with respect to the number of transitions of V, therefore, the computational complexity of Algorithm 3.1 is equal to $|X_h| \cdot (|X_h| + 2) \cdot (|X_a| + 1) \cdot |X_g| \cdot |\Sigma|$, *i.e.*, $O(|X_h|^2 \cdot |X_a| \cdot |X_g| \cdot |\Sigma|)$.

Consider now the following proposition.

Proposition 3.2 *Let $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1}, X_{m_1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2}, X_{m_2})$ such that $L(G_1) \subseteq L(G_2)$, $L_m(G_1) \subseteq L_m(G_2)$. In addition, assume that for every $s, s' \in L(G_1)$ such that $f_1(x_{0_1}, s) = f_1(x_{0_1}, s')$, then $f_2(x_{0_2}, s) = f_2(x_{0_2}, s')$. Construct two automata G_1' and G_2' from G_1 and G_2 by adding n_1 and n_2 new states, respectively, and only adding new transitions from the states of G_1 and G_2 to the new states and between the new states. Then, automaton $G_1' \times G_2'$ has at most $|X_1| + n_1(|X_2| + n_2)$ states.*

Proof: In accordance with the construction of G_1' and G_2', we conclude that $(G_1 \times G_2) \sqsubseteq (G_1' \times G_2')$ and every state of $G_1' \times G_2'$ outside $G_1 \times G_2$ has the first component equal to one of the new states added to G_1. Therefore, $G_1' \times G_2'$ has, at most, $|X_{1\times 2}| + n_1(|X_2| + n_2)$ states, where $X_{1\times 2}$ denotes the set of states of automaton $G_1 \times G_2$. Define function $\Theta : X_{1\times 2} \to X_1$ as the mapping $\Theta((x_1, x_2)) = x_1$, $\forall (x_1, x_2) \in X_{1\times 2}$. Function Θ is bijective since, for every $s, s' \in L(G_1)$, $f_1(x_{0_1}, s) = f_1(x_{0_1}, s') \Rightarrow f_2(x_{0_2}, s) = f_2(x_{0_2}, s')$ and $L(G_1) \subseteq L(G_2)$. Therefore, $|X_{1\times 2}| = |X_1|$, which concludes the proof. ■

In the verification of observability, $H = A$ and, thus, applying Proposition 3.2 with $G_1 = H$, $G_2 = A = H$, $G_1' = N$ and $G_2' = M$, we conclude that automaton $N \times M$ has at most $3|X_h| + 2$ states, and, consequently, automaton $V = H_m^R \| (N \times M \times G_m)$ has $|X_h| \cdot (3|X_h| + 2) \cdot |X_g|$ states, at most. Therefore, the complexity of the verification of language observability by applying Algorithm 3.1 is $O(|X_h|^2 \cdot |X_g| \cdot |\Sigma|)$, which is equal to the complexity of the algorithm for the verification of observability proposed by [78].

3.5.2 Computational Complexity of Algorithm 3.2

In the first step of Algorithm 3.2, we compute automaton H_{sp}, that has, at most, $2^{|X_h|} \cdot |X_h|$ states and $2^{|X_h|} \cdot |X_h| \cdot |\Sigma|$ transitions. Since we execute Steps 3 to 5 iteratively, and, for each iteration, at least one transition is removed from automaton H_{sp}, then, the number of iterations is at most equal to the number of transitions of H_{sp}. In addition, at each iteration, we compute verifier V using Algorithm 3.1 with inputs H_s (subautomaton of H_{sp}), A and G. Since H_s and A satisfy Assumption **A1**, by applying Proposition 3.2 with $G_1 = H_s$, $G_2 = G_2' = A$ and $G_1' = N$, we conclude that automaton $N \times A$ has at most $|X_s| + 2|X_a|$ states, where $|X_s| \leq 2^{|X_h|} \cdot |X_h|$. Moreover, since A and G satisfy Assumption **A2**, by using Proposition 3.2 with $G_1 = N \times A$, $G_2 = G_2' = G_m$ and $G_1' = N \times M$, we conclude that automaton $H_c = N \times M \times G_m$ has at most $|X_s| + 2(|X_a| + |X_g|)$ states and, consequently, automaton V has, at most, $|X_s|^2 + |X_s| \cdot 2(|X_a| + |X_g|)$ states and, thus, the computational complexity of one iteration of Algorithm 3.2 is $O([|X_s|^2 + |X_s| \cdot (|X_a| + |X_g|)] \cdot |\Sigma|)$. Therefore, the complexity of Algorithm 3.2 is $O\left(\left[2^{3|X_h|} \cdot |X_h|^3 + 2^{2|X_h|} \cdot |X_h|^2 \cdot (|X_a| + |X_g|)\right] \cdot |\Sigma|^2\right)$.

When Algorithm 3.2 is applied for the computation of the supremal $\overline{K}$-observable sublanguage of K with respect to G and P_o, then $A = H$, and, thus, the complexity of Algorithm 3.2 becomes $O\left(\left[2^{3|X_h|} \cdot |X_h|^3 + 2^{2|X_h|} \cdot |X_h|^2 \cdot |X_g|\right] \cdot |\Sigma|^2\right)$, which is smaller than those complexities of the algorithms proposed in [15] and [20], which are $O(2^{[(2^{|X_h|} \cdot |X_h|+1)|X_g|+|X_h|]} \cdot |X_h| \cdot |\Sigma|)$ and $O\left(2^{3|X_g| \cdot |X_h|} \cdot |X_g|^6 \cdot |X_h|^7 \cdot |\Sigma|\right)$, respectively.

Remark 3.4 *It is important to remark that, when H is already a state partition automaton, i.e. when $H_{sp} = H$, the complexity of Algorithm 3.2 becomes $O\left((|X_h|^3 + |X_h|^2 \cdot |X_g|) \cdot |\Sigma|^2\right)$, being therefore polynomial. Notice that the state partition assumption was made in [15], but the complexity of the algorithm proposed in [15] is still exponential, being $O(2^{(|X_h|+1)|X_g|} \cdot |X_h| \cdot |\Sigma|)$. Therefore, even in this situation, the algorithm proposed here performs better.*

3.5.3 Computational Complexity of Algorithm 3.3

In Step 2 of Algorithm 3.3, we compute automaton H_{sr}, that has, at most, $2^{|X_h|} \cdot |X_h| \cdot |X_g|$ states and $2^{|X_h|} \cdot |X_h| \cdot |X_g| \cdot |\Sigma|$ transitions. Since we execute

Steps 3 to 7 iteratively, and, for each iteration, at least either one state (and its associated transitions) or one transition of H_{sr} is removed from automaton H_{sr}, then, the number of iterations is at most equal to the number of transitions of H_{sr}. Steps 3 and 4 are computed with linear complexity with respect to the number of transitions of H_{sr}. In Step 5, we compute verifier V using Algorithm 3.1 with inputs G, H_{sr}, and H_{sr}. Notice that, since $A = H_{sr}$, we conclude that automaton $M \times N$ has $|X_{sr}| + 2$ states, where the new states are (x_d, x_{d1}) and (x_d, x_{d2}). Moreover, in accordance with the construction of H_{sr} in Step 2, $\forall s_1, s_2 \in L(H_{sr})$, if $f_{sr}(x_{0_{sr}}, s_1) = f_{sr}(x_{0_{sr}}, s_2)$, then $f_g(x_{0_g}, s_1) = f_g(x_{0_g}, s_2)$. Thus, by using Proposition 3.2 with $G_1 = H_{sr}$, $G_2 = G_2' = G_m$ and $G_1' = M \times N$, we conclude that automaton $H_c = N \times M \times G_m$ has at most $|X_{sr}| + 2|X_g|$ states and, consequently, automaton V has, at most, $|X_{sr}|^2 + 2|X_{sr}| \cdot |X_g|$ states and $(|X_{sr}|^2 + 2|X_{sr}| \cdot |X_g|) \cdot |\Sigma|$ transitions, which implies that the computational complexities of Steps 5 to 7 of Algorithm 3.3 are $O\left((|X_{sr}|^2 + |X_{sr}| \cdot |X_g|) \cdot |\Sigma|\right)$. From the computational complexities of Steps 3 to 7, we can conclude that each iteration of Algorithm 3.3 is $O\left((|X_{sr}|^2 + |X_{sr}| \cdot |X_g|) \cdot |\Sigma|\right)$. Therefore the complexity of Algorithm 3.3 is $O\left(2^{3|X_h|} \cdot |X_h|^3 \cdot |X_g|^3 \cdot |\Sigma|^2\right)$.

3.6 Concluding Remarks

In this chapter, we presented a new property of relative observability which was leveraged in order to derive three new algorithms: the first one for the verification of relative observability and the second one for the computation of the supremal relatively observable sublanguage; the former has polynomial time complexity, whereas the latter, although, in general, has exponential complexity, it will have polynomial complexity when the automaton that marks the specification language is state partition. The third algorithm, which is based on the second one, can be used to compute a controllable and observable sublanguage of a regular language by using the concept of relative observability.

Chapter 4

Supervisory Control of Networked Discrete Event Systems With Timing Structure

In the standard supervisory control problem described in Section 2.5 [23, 24], it was assumed that supervisors observe the occurrence of an event immediately after it has been executed by the plant and without loss of observation. In [1] and [53], we have weakened this assumption by assuming loss of observations caused by either intermittent sensor malfunction (when a sensor fails intermittently to issue a signal associated with an event occurrence) or intermittent communication problems (a signal issued by a sensor sometimes does not reach the supervisor). As a consequence, the event occurrence is not observed by the supervisor. Nevertheless, we still assumed, in [53] and [1], that the supervisor instantaneously observes the occurrences of observable events. When the plant and supervisors are either far from each other or a more complex network is used to connect them, communication delays are unavoidable and must be taken into account. Such a control problem is referred, in the literature, to as supervisory control of networked discrete event systems [14].

In this chapter, we consider the supervisory control problem of Networked Discrete Event Systems With Timing Structure (NDESWTS) subject to bounded communication delays and intermittent loss of observations. We assume that the communication between the plant and the supervisor is carried out through a network that can have several channels, so that communication delays can cause changes in

the order of the observations. Preliminary versions of the results presented in this chapter have been published in [88, 89]. In addition, the model for NDESWTS proposed here has also been used to address the codiagnosability of NDESWTS subject to event communication delays and loss of observations [66, 67].

It is worth remarking that, in this chapter, we will not address the blocking properties of supervisory control, which is equivalent to assuming only prefix-closed admissible languages. Therefore, the sets of marked states will be, from this point onwards, omitted in the automaton definition.

The remainder of this chapter is structured as follows. In Section 4.1, we formally define NDESWTS, and present a motivating example, and propose a new model for NDESWTS in Section 4.2. In section 4.3, we formulate the supervisory control problem for NDESWTS (Subsection 4.3.1), present some new results, such as, a necessary and sufficient condition for the existence of networked supervisors (Subsection 4.3.2), and combine these results to present a systematic way to design networked supervisors (Subsection 4.3.3). We present some final comments in Section 4.4.

4.1 Networked Discrete Event Systems With Timing Structure

We will consider the system structure depicted in Figure 4.1, where the plant is modeled by an automaton $G = (X, \Sigma, f, \Gamma, x_0)$ which communicates with the supervisory control system through a communication network; the information flow is indicated by dashed lines. The occurrence of observable events are communicated to the supervisor through m different communication channels (referred here to as observation channels), and the supervisor control action is transmitted to the low level controllers through n different communication channels (referred here to as control channels).

We make the following assumptions.

A1. No event can be transmitted through two different observation channels.

A2. Each observation channel och_i, $i = 1, \ldots, m$, is modeled by a first-in first-out

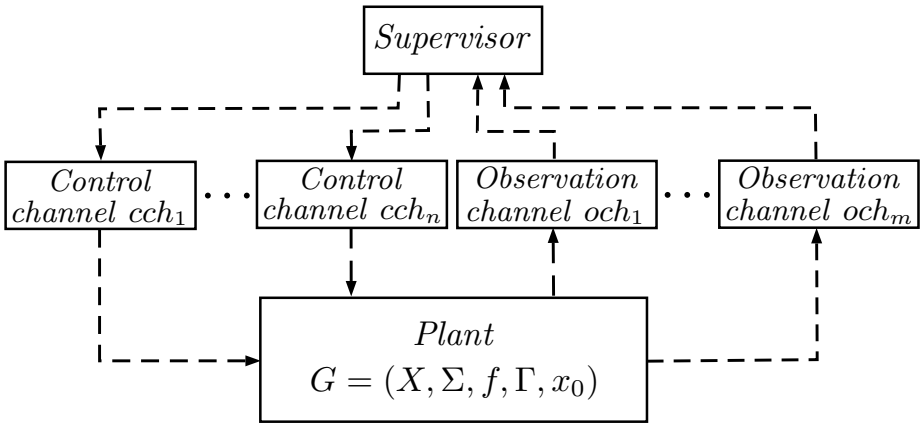


Figure 4.1: Networked supervisory control architecture.

(FIFO) queue, and the events whose occurrences are transmitted through och_i are subject to observation delays[1] whose upper bound is known a priori to be equal to T_i, where T_i is a finite positive real number different from zero.

A3. The set of observable events is partitioned, with respect to loss of observations, as $\Sigma_o = \Sigma_{lo} \dot{\cup} \Sigma_{nlo}$, where Σ_{lo} (resp. Σ_{nlo}) is the set of observable events that are subject (resp. not subject) to loss of observations.

A4. The control actions corresponding to controllable events transmitted through control channels are not subject to transmission delays or losses.

According to Assumption **A1**, the set of observable events is partitioned as $\Sigma_o = \dot{\bigcup}_{i=1}^{m} \Sigma_{o,i}$, where $\Sigma_{o,i}$, $i = 1, \ldots, m$, is the set of observable events whose occurrences are transmitted through observation channel och_i. Assumption **A2** implies that the maximal observation delays must be different from zero for all observable events and are determined in accordance with their associated observation channels. According to Assumption **A3**, the events that belong to Σ_{lo} are subject to losses of observations caused either by package losses in the observation channels or malfunctioning sensors. Finally, according to Assumption **A4**, we will not consider delays and losses of control actions.

In order to model the effects of the dynamic behavior of real systems, we define the partial function $t_{min} : X \times \Sigma \to \mathbb{R}_+$ ($\mathbb{R}_+$ denotes the set of non-negative real numbers) where $t_{min}(x, \sigma) = \tau$, for $\sigma \in \Gamma(x)$, means that event σ can only occur at

[1]The observation delay of an event corresponds to the time interval elapsed between its occurrence in the plant and its successful observation by the supervisor, that is, the observation delay includes the time intervals taken to record and to transmit the event occurrence.

state x if the time elapsed from the last transition is greater than (but not equal to) τ. In addition, we assume that, for all $x \in X$ and $\sigma \in \Gamma(x)$, $t_{min}(x, \sigma) > 0$. The need for excluding $\tau = 0$ is imposed by a technical constraint to allow the NDESWTS to be converted into an untimed finite-state automaton. With a slight abuse of language, we will refer to $t_{min}(s, \sigma)$ as the minimal activation time. Partial function t_{min} is extended to domain $X \times \Sigma^*$ in the following recursive manner: for all $x \in X$, $t_{min}(x, \varepsilon) = 0$, and $t_{min}(x, s\sigma) = t_{min}(x, s) + t_{min}(f(x, s), \sigma)$ for $s \in \Sigma^*$ and $\sigma \in \Sigma$.

Example 4.1 *Consider the NDESWTS whose communication network is depicted in Figure 4.2(a). The plant is modeled by automaton* $G = (X, \Sigma, f, \Gamma, x_0)$ *depicted in Figure 4.2(b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$, $\Sigma_c = \{\alpha, \gamma, \mu, \eta\}$ *and* $\Sigma_{lo} = \emptyset$. *In Figure 4.2(b), the time value attached to each transition* $(x, \sigma, f(x, \sigma))$ *of* G *represents the minimal activation time* $t_{min}(x, \sigma)$. *According to Figure 4.2(a), the occurrences of events in* $\Sigma_{o,1} = \{\alpha\}$ *are transmitted through observation channel*

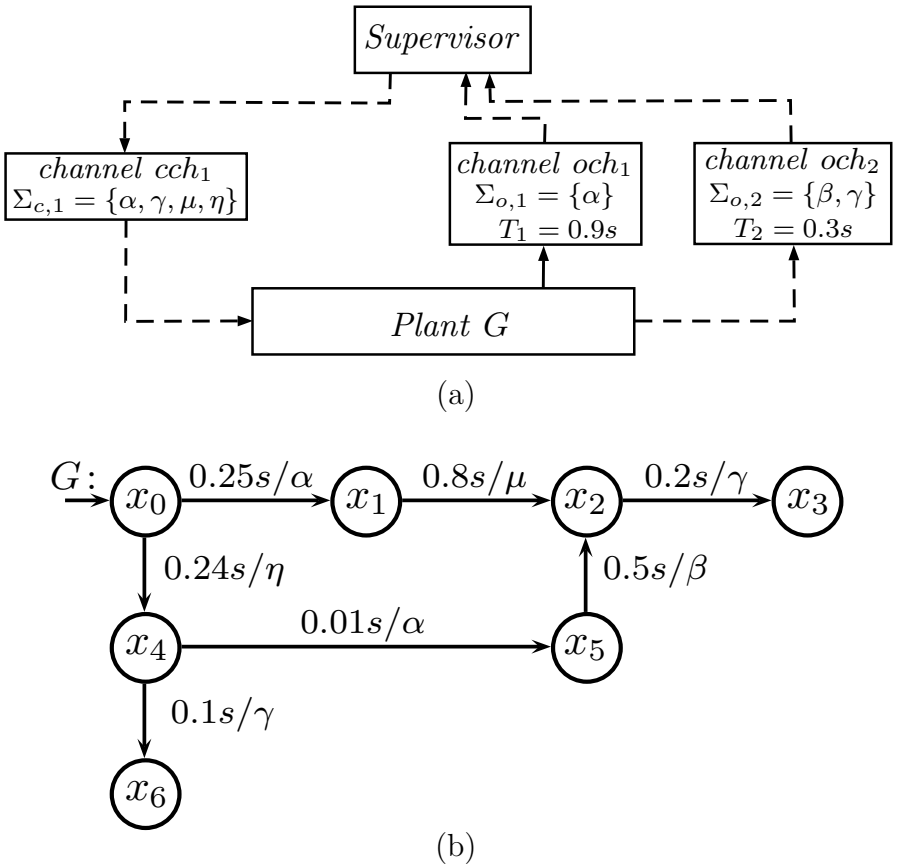


Figure 4.2: Networked discrete event system: communication network (a) and automaton G (b).

och_1 and their maximal observation delay is $T_1 = 0.9s$, and the occurrences of events in $\Sigma_{o,2} = \{\beta, \gamma\}$ are transmitted through observation channel och_2 and their maximal observation delay is $T_2 = 0.3s$.

*Let us assume that plant G generates string $s_1 = \eta\alpha\beta\gamma$, formed with a single unobservable event (η). As illustrated in Figure 4.3, the occurrence of event α is transmitted through channel och_1, and, thus, its maximal observation delay is $T_1 = 0.9s$, whereas the occurrence of events β and γ are transmitted through channel och_2, and, thus, their maximal observation delay is $T_2 = 0.3s$. As a consequence, event α may be observed by the supervisor in the following ways: (i) before the occurrence of γ, when the delay of the observation of α is smaller than the time elapsed between the occurrences of α and γ; (ii) between the occurrence and observation of event γ, since the observation of α can be delayed by at most $0.9s$ and the plant can generate $\beta\gamma$ after $0.7s$ has elapsed since the occurrence of event α; (iii) after the observation of γ, since, as shown before, α can be observed after the occurrence of γ and the occurrences of α and γ are transmitted through different channels. The supervisor may also observe event β after the occurrence of event γ, since $T_2 = 0.3s$ and γ may occur after $0.2s$ has elapsed since the execution of event β. However, regardless of the interval of observation of β overlaps the interval of observation of γ (red interval in Figure 4.3), the supervisor cannot observe γ before observing β, since β and γ are transmitted through the same channel, which, according to Assumption **A2**, is modeled by a FIFO queue.*

Let us now assume that plant G generates string $s_2 = \alpha\mu\gamma$. As illustrated in Figure 4.4, in this case, although event α may be observed after the occurrence of

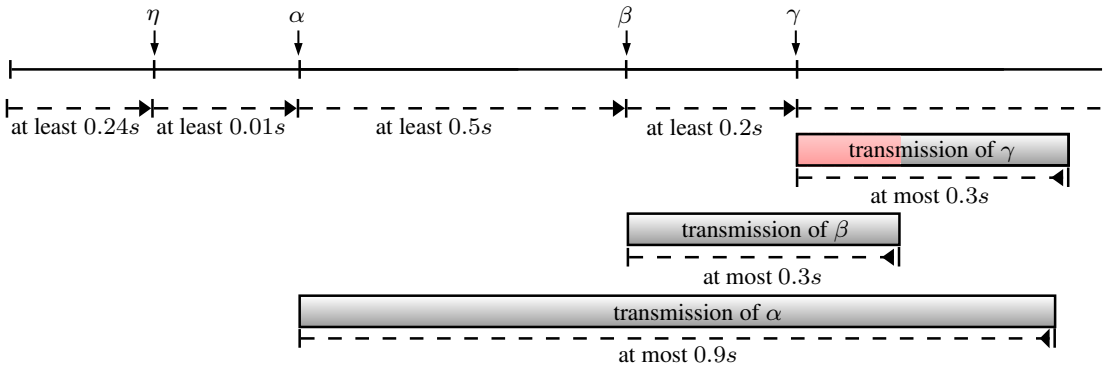


Figure 4.3: Possible cases of observation of string $s_1 = \eta\alpha\beta\gamma$.

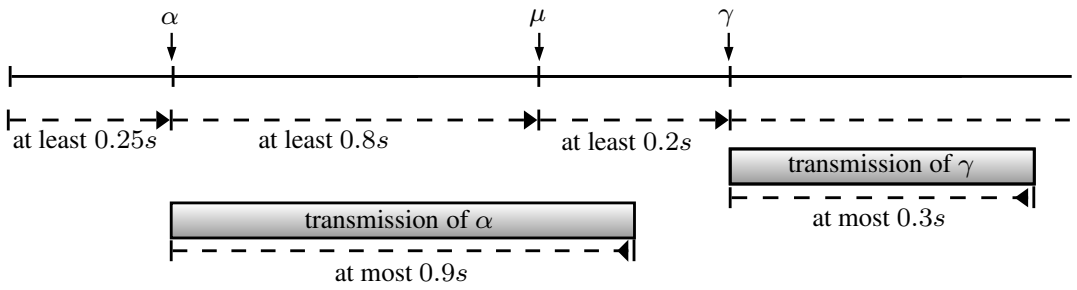


Figure 4.4: Possible cases of observation of string $s_2 = \alpha\mu\gamma$.

event μ (which is unobservable), it cannot be observed after the occurrence of γ, since the plant can only generate string $\mu\gamma$ at least $1s$ after executing event α, whose observation can be delayed by at most $0.9s$.

Therefore, although both strings $s_1 = \eta\alpha\beta\gamma$ and $s_2 = \alpha\mu\gamma$ have one event between α and γ, event α can be observed by the supervisor after the occurrence (or observation) of γ when the plant generates string s_1, but it cannot be observed after the occurrence (or observation) of γ when the plant generates string s_2.

Based on the previous example we may draw the following conclusions:

1. When several communication channels are used to transmit event occurrences to the supervisor, changes in the order of the received observations are likely to occur. Such a possibility does not arises in the previous approaches [14, 30–37, 39–47], since they assume that the observations are always received in the same order as the event occurrences.

2. The concept of *step* [14, 32–36, 39–46, 49, 50, 54–56, 59, 60] cannot be used to correctly model the system presented in the example. This is so because string $s_1 = \eta\alpha\beta\gamma$ requires two *steps* to model all possible delays in the observation of event α, whereas string $s_2 = \alpha\mu\gamma$ requires only one *step* to model all possible delays in the observation of event α. Therefore, if we assume $T_1 = 2$ *steps*, then the supervisory control problem may become more restrictive, since we are considering that α can be observed after the occurrence (or observation) of γ when the plant executes string s_2, which cannot occur in practice. On the other hand, assuming $T_1 = 1$ *step* may lead to wrong control actions when α is observed after either the occurrence or the observation of γ during the execution of string s_1.

3. If we use the model based on the TDES framework proposed in [38], which is used in [37, 47, 58], the state dimension of the corresponding untimed model would be huge; for example, as we can see in Figure 4.2(b), $t_{min}(x_1, \mu) = 80 \times t_{min}(x_4, \alpha)$, which implies that, in the automaton model proposed in [38], transition (x_1, μ, x_2) becomes a path with at least 80 states connected by transitions labeled by the *tick* event since $t_{min}(x_4, \alpha) = 0.01s$ forces the *tick* event to be smaller than $0.01s$.

4.2 An Untimed Discrete Event System Characterization of NDESWTS

In this section, we will propose a new model formalism for NDESWTS subject to delays and losses of observations that allows us to represent distributed systems with heterogeneous temporal behavior and a multi-channel communication network. We will improve the observation delay model proposed in [49] to directly apply time information instead of using the concept of *step*. The modeling of loss of observations will be carried out by using the Dilation operation approach proposed in [62] to model intermittent loss of observations in the context of codiagnosability of DES.

We will model the behavior of a NDESWTS by means of an automaton $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e})$ whose generated language is defined over the extended set of events,

$$\Sigma_e := \Sigma \cup \Sigma_o^s \cup \Sigma_o^l, \tag{4.1}$$

where the added event sets Σ_o^s and Σ_o^l are used to model, respectively, the successful observation and the loss of observation of observable events, being defined as follows:

$$\Sigma_o^s := \{\sigma_s : \sigma \in \Sigma_o\} \text{ and } \Sigma_o^l := \{\sigma_l : \sigma \in \Sigma_{lo}\}. \tag{4.2}$$

As shown in the motivating example presented in Subsection 4.1, a given string executed by the plant can generate a set of different observations. As a consequence, every string $s \in L(G)$ can be represented by a set of extended strings in Σ_e^* defined in accordance with $\Sigma_{o,i}$ and T_i, $i = 1, ..., m$, t_{min} and Σ_{lo}, as illustrated in the following example.

Example 4.2 *Consider the NDESWTS presented in Example 4.1, and depicted in Figures 4.2(a,b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$ *and* $\Sigma_o = \{\alpha, \beta, \gamma\}$*. Assume that event* β *is subject to loss of observations, i.e.,* $\Sigma_{lo} = \{\beta\}$*. Let us suppose that the plant generates string* $s = \eta\alpha\beta$*. Then, due to delays and loss of observations, the following extended strings can be generated:*

- $s_{e,1} = \eta\alpha\alpha_s\beta\beta_s$*, that models the case when the delay of the observation of* α *is smaller than the time interval between the occurrences of* α *and* β *(inside Time Interval 1 of Figure 4.5);*

- $s_{e,2} = \eta\alpha\beta\alpha_s\beta_s$*, that models the case when the supervisor observes* α *between the occurrence of* β *and its observation (inside Time Interval 2 of Figure 4.5);*

- $s_{e,3} = \eta\alpha\beta\beta_s\alpha_s$*, that models the case when the supervisor firstly observes event* β *and only observes event* α *after observing* β *(inside Time Interval 3 of Figure 4.5);*

- $\eta\alpha\alpha_s\beta\beta_l$*,* $\eta\alpha\beta\alpha_s\beta_l$ *and* $\eta\alpha\beta\beta_l\alpha_s$ *that model the cases when the observation of* β *is lost. Notice that these strings have been obtained from the previous ones by replacing* β_s *with* β_l*.*

- *Extended strings* $\eta\alpha\alpha_s\beta$*,* $\eta\alpha\beta$*,* $\eta\alpha\beta\alpha_s$*,* $\eta\alpha\beta\beta_s$ *and* $\eta\alpha\beta\beta_l$ *model the cases when some observations have not reached the supervisor yet, being, therefore, prefixes of the aforementioned extended strings.*

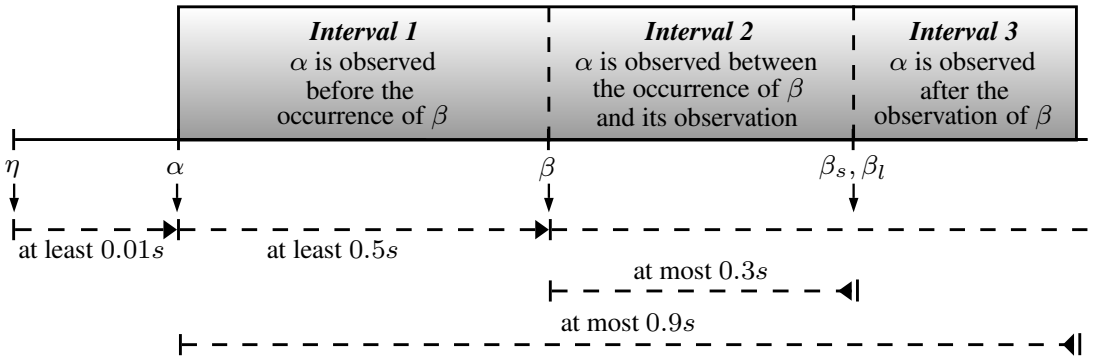


Figure 4.5: Possible cases of observation of string $s = \eta\alpha\beta$ modeled by extended strings over Σ_e.

Let us now define the following functions:

- $\psi : \Sigma_o^* \rightarrow \Sigma_o^{s*}$, where $\psi(\varepsilon) = \varepsilon$, $\psi(\sigma) = \sigma_s$, for all $\sigma \in \Sigma_o$, and $\psi(s\sigma) = \psi(s)\psi(\sigma)$, for all $s \in \Sigma_o^*$ and $\sigma \in \Sigma_o$. Function ψ is extended to languages by applying it to all strings in the language.

- $\psi^{-1} : \Sigma_o^{s*} \rightarrow \Sigma_o^*$, which is the inverse function of ψ.

In addition, let us define the following projections: $P_e : \Sigma_e^* \rightarrow \Sigma^*$ and $P_{e,s} : \Sigma_e^* \rightarrow \Sigma_o^{s*}$.

Example 4.3 *Consider the NDESWTS presented in Example 4.1, which is depicted in Figures 4.2(a,b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$ *and* $\Sigma_{lo} = \{\beta\}$, *and assume again that string* $s = \eta\alpha\beta$ *has been generated by the plant.*

Firstly, notice that, all extended strings presented in the previous example have the same projection over Σ^*, *being equal to* s.

Now, observe that an extended string $s_e \in \Sigma_e^*$ *models the case when the plant generates string* $P_e(s_e)$, *and the supervisor observes string* $\psi^{-1}(P_{e,s}(s_e))$; *for example, extended strings* $s_{e,1} = \eta\alpha\alpha_s\beta\beta_s$ *and* $s_{e,2} = \eta\alpha\beta\alpha_s\beta_s$ *represent the case when the supervisor observes* $\alpha\beta$ *since* $\psi^{-1}(P_{e,s}(s_{e,1})) = \psi^{-1}(P_{e,s}(s_{e,2})) = \alpha\beta$, *whereas extended string* $s_{e,3} = \eta\alpha\beta\beta_s\alpha_s$ *models the case when the supervisor observes* $\psi^{-1}(P_{e,s}(s_{e,3})) = \beta\alpha$. *On the other hand, extended string* $s_{e,4} = \eta\alpha\beta\beta_l\alpha_s$ *models a case when the supervisor makes its decision based on the observation of* $\psi^{-1}(P_{e,s}(s_e)) = \alpha$, *since the observation of* β *is lost.*

In order to characterize the set of extended strings associated with a string, or a sublanguage, of $L(G)$, we will first introduce a function that models the effects of possible observation delays. To this end, let us define projections $P_{e,i} : \Sigma_e^* \rightarrow \Sigma_{o,i}^*$ and $P_{e,s_i} : \Sigma_e^* \rightarrow [\psi(\Sigma_{o,i})]^*$. In addition, for $\sigma \in (\Sigma \cup \Sigma_o^s)$ and $w \in (\Sigma \cup \Sigma_o^s)^*$, let $\sigma^{(j)} \in w$ denote that there are, at least, j occurrences of event σ in string w, being $\sigma^{(j)}$ the j-th occurrence of σ in w. In addition, let $w_{\sigma^{(j)}}$ be the prefix of w that finishes at the j-th occurrence of σ in w, if $\sigma^{(j)} \in w$, or w, if $\sigma^{(j)} \notin w$.

Definition 4.1 (delay extension function) *The Delay extension function is the mapping*

$$\begin{array}{rccc} D_d : & L(G) & \rightarrow & 2^{(\Sigma \cup \Sigma_o^s)^*} \\ & s & \mapsto & D_d(s) \end{array}$$

where $w \in D_d(s)$ if w satisfies the following conditions:

1. $P_e(w) = s$.

2. *For all $\sigma \in \Sigma_{o,i}$, $i = 1, \ldots, m$, and $\sigma^{(j)} \in w$:*

$$t_{min}(x_0, P_e(w_{\sigma_s^{(j)}})) - t_{min}(x_0, P_e(w_{\sigma^{(j)}})) < T_i. \tag{4.3}$$

3. *For all $\sigma_s \in \psi(\Sigma_{o,i})$, $i = 1, \ldots, m$, and $\sigma_s^{(j)} \in w$:*

$$\sigma^{(j)} \in w_{\sigma_s^{(j)}} \tag{4.4}$$

and

$$\|P_{e,i}(w_{\sigma^{(j)}})\| = \|P_{e,s_i}(w_{\sigma_s^{(j)}})\|. \tag{4.5}$$

The extension of D_d to domain $2^{L(G)}$ is defined as $D_d(L) := \bigcup_{s \in L} D_d(s)$.

Condition 1 of Definition 4.1 ensures that w is obtained from s by inserting events only from Σ_o^s. Condition 2 enforces that the delay between the occurrence of an event $\sigma \in \Sigma_{o,i}$ and its observation $\sigma_s = \psi(\sigma)$ is not larger than the delay upper bound T_i (Equation (4.3)). Finally, Condition 3 ensures that the observation of an event $\sigma_s \in \Sigma_o^s$ can only occur after event $\sigma = \psi^{-1}(\sigma_s)$ has occurred in w (Equation (4.4)), and that observations of events transmitted through the same observation channel must be in the same order of their occurrences in s (Equation (4.5)).

We now redefine dilation operation [1, 62] in domain $(\Sigma \cup \Sigma_o^s)^*$, which is denoted by D_l and models the effects of loss of observations in extended languages, *i.e.*, when an event $\sigma \in \Sigma_{lo}$ occurs, either σ_s or σ_l will occur.

Definition 4.2 (dilation function [62]) *The Dilation function $D_l : (\Sigma \cup \Sigma_o^s)^* \to 2^{\Sigma_e^*}$ is recursively defined as:*

$$\begin{aligned}
D_l(\varepsilon) &:= \{\varepsilon\}, \\
D_l(\sigma) &:= \{\sigma\}, \forall \sigma \in \Sigma, \\
D_l(\sigma_s) &:= \begin{cases} \{\sigma_s\}, \text{ if } \sigma_s \in \Sigma_o^s \setminus \psi(\Sigma_{lo}), \\ \{\sigma_s, \sigma_l\}, \text{ if } \sigma_s \in \psi(\Sigma_{lo}), \end{cases} \\
D_l(w\sigma) &:= D_l(w)D_l(\sigma), \ \forall w \in (\Sigma \cup \Sigma_o^s)^*, \ \forall \sigma \in (\Sigma \cup \Sigma_o^s).
\end{aligned}$$

The extension of D_l to domain $2^{(\Sigma\cup\Sigma_o^s)^*}$ is defined as $D_l(L) := \bigcup_{w\in L} D_l(w)$.

Finally, in order to characterize the joint effects of delays and losses of observations, we define the extension operation as the composition of the delay extension function with the dilation function, as follows.

Definition 4.3 (extension function) *The Extension function is the mapping*

$$\begin{aligned} E: \ 2^{L(G)} &\rightarrow 2^{\Sigma_e^*} \\ L &\mapsto E(L) := D_l(D_d(L)). \end{aligned}$$

According to Definition 4.3, for a given string $s \in L(G)$, extended language $E(\{s\})$ (hereafter, denoted simply by $E(s)$) is formed by those extended strings that can be generated when s is executed by the plant in the presence of delays and loss of observations. The following example illustrates operations D_d, D_l and E.

Example 4.4 *Consider again the NDESWTS depicted in Figures 4.2(a,b), where $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$ and $\Sigma_{lo} = \{\beta\}$.*

In order to highlight how Definition 4.1 works, consider string $s_1 = \alpha\mu\gamma \in L(G)$ and extended strings $w = \eta\alpha\alpha_s$, $v = \alpha\mu\gamma\alpha_s\gamma_s$ and $u = \alpha\alpha_s\mu\gamma\beta_s\gamma_s$. Notice that these strings cannot be generated when string s_1 is executed by the plant since: (i) string w cannot be generated when s_1 is executed since $P_e(w) = \eta\alpha$, which implies, according to Condition 1 of Definition 4.1, that it corresponds to the execution of string $\eta\alpha$; (ii) regarding string v, notice that, although the delay upper bound of the observation of α is $0.9s$, its observation, modeled by event α_s in v, accounts for a delay of, at least, $1s$. This is so because, according to Condition 2 of Definition 4.1 (Equation (4.3)), $t_{min}(x_0, P_e(v_{\alpha_s^{(1)}})) - t_{min}(x_0, P_e(v_{\alpha^{(1)}})) = t_{min}(x_0, \alpha\mu\gamma) - t_{min}(x_0, \alpha) = 1.25 - 0.25 = 1 > 0.9$, and; (iii) finally, u has one occurrence of event β_s which is not possible since, although β is an observable event, there has not been any occurrence of event β in string s_1, which is captured by Condition 3 of Definition 4.1 (Equation (4.4)), since $\beta^{(1)} \notin u_{\beta_s^{(1)}} = \alpha\alpha_s\mu\gamma\beta_s$. Consider now string $s_2 = \eta\alpha\beta\gamma \in L(G)$ and extended string $z = \eta\alpha\alpha_s\beta\gamma\gamma_s\beta_s$. Notice that this extended string cannot be generated when the plant executes string s_2 since the observations of events β and γ (β_s and γ_s, respectively) are incorrect, since these events are transmitted through

the same observation channel and event γ has occurred after event β in s_2, which is also captured by Condition 3 of Definition 4.1 (Equation (4.5)).

Finally, consider that string $s = \eta\alpha\beta$ has been generated by the plant. By applying Definition 4.1, we obtain, as shown in Example 4.2, the following set of extended strings in $(\Sigma \cup \Sigma_o^s)^*$ that can be generated when the plant executes s and only delays of observations are taken into account:

$$D_d(s) = \{\eta\alpha\beta, \eta\alpha\alpha_s\beta, \eta\alpha\beta\alpha_s, \eta\alpha\beta\beta_s, \eta\alpha\alpha_s\beta\beta_s, \eta\alpha\beta\alpha_s\beta_s, \eta\alpha\beta\beta_s\alpha_s\}.$$

Notice that all strings in $D_d(s)$ satisfy Conditions 1, 2 and 3 of Definition 4.1. By applying Dilation operation to $D_d(s)$, we obtain, according to Definitions 4.2 and 4.3, the following set that is formed with all extended strings in Σ_e^* that can be generated when the plant executes s in the presence of delays and loss of observations:

$$E(s) = D_l(D_d(s)) = D_d(s) \cup \{\eta\alpha\beta\beta_l, \eta\alpha\alpha_s\beta\beta_l, \eta\alpha\beta\alpha_s\beta_l, \eta\alpha\beta\beta_l\alpha_s\}.$$

We will now propose an algorithm to construct, recursively, automaton G_e such that $L(G_e) = E(L(G))$. In order to do so, each state of this automaton will have two components, as follows: the first component will account for the corresponding state x of plant G, and the second component will account for the observable events that were generated by G in order to reach state x and whose observations are being transmitted to the supervisor together with the minimum time elapsed between the occurrences of these observable events. For example, as illustrated in Figure 4.6, state $(x_3, \alpha 0.5\beta 0.2\gamma)$ of G_e corresponds to the case when the plant has reached state x_3 after the execution of a string $s \in L(G)$ that contains, in that order, the observable events α, β and γ whose observations are still being transmitted to the supervisor, and the times elapsed between the occurrences of α and β (resp. β and γ) are, at least, equal to 0.5 (resp. 0.2) time units. Finally, it is important to remark that state $(x_3, \alpha 0.5\beta 0.2\gamma)$ is viable since it is possible, as shown in Example 4.1, for string $\eta\alpha\beta\gamma$ to occur before α is observed.

The definition of the second components of the states of G_e is carried out by manipulating observable events and non-negative real numbers. To this end, let $\mathcal{Q} := \{q = q_1 q_2 \cdots q_l : \forall i \in \{1, 2, ..., l\}, (q_i \in \Sigma_o) \vee (q_i \in \mathbb{R}_+)\}$ be a set whose

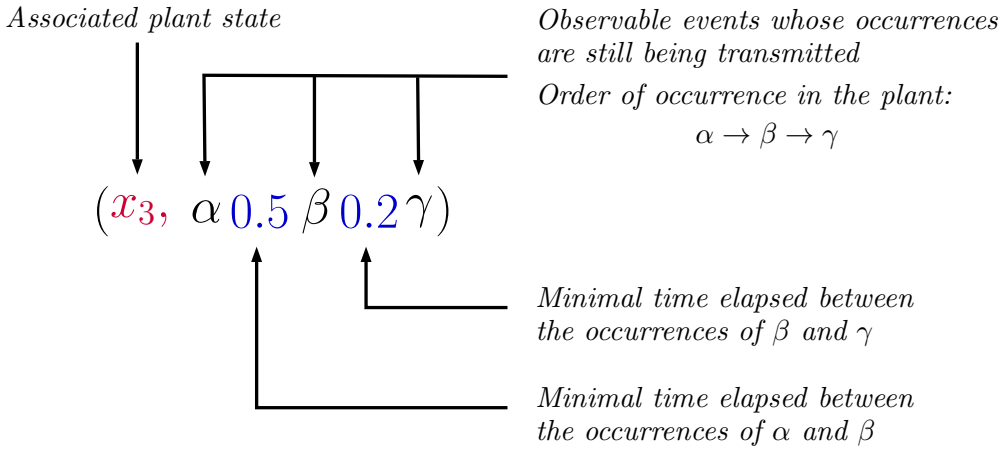


Figure 4.6: Example of a possible state of automaton G_e.

elements are strings formed by observable events and non-negative real numbers. In the following definition, we propose some operations over $\mathcal{Q}$, and a function that relates an observable event with the observation channel used to transmit its occurrence to the supervisor.

Definition 4.4

- *The function link* : $\mathcal{Q} \times \mathcal{Q} \rightarrow \mathcal{Q}$ *is a mapping where, for every pair* $q = q_1 \cdots q_k$ *and* $p = p_1 \cdots p_v$ *belonging to* $\mathcal{Q}$,

$$link(q,p) := \begin{cases} q_1 \cdots q_{k-1}\,(q_k + p_1)\,p_2 \cdots p_v, \text{ if } q_k, p_1 \in \mathbb{R}_+ \\ q_1 \cdots q_k\, p_1 \cdots p_v, \text{ otherwise.} \end{cases}$$

- *The function cut* : $\mathcal{Q} \rightarrow \mathcal{Q}$ *is a mapping where, for all* $q = q_1 q_2 \cdots q_k \in \mathcal{Q}$,

$$cut(q) := \begin{cases} q_y\, q_{y+1} \cdots q_k, \text{ if } (\exists y \leq k)[(q_y \in \Sigma_o) \wedge (q_j \in \mathbb{R}_+, \forall j \in \{1, ..., y-1\})] \\ 0, \text{ if } q_y \in \mathbb{R}_+, \forall y \in \{1, 2, ..., k\}. \end{cases}$$

- *The function add* : $\mathcal{Q} \times X \times \Sigma \rightarrow \mathcal{Q}$ *is a mapping where, for all* $q \in \mathcal{Q}$, $x \in X$ *and* $\sigma \in \Sigma$,

$$add(q,x,\sigma) := \begin{cases} cut(link(q, t_{min}(x,\sigma)\sigma)), \text{ if } (\sigma \in \Sigma_o) \wedge (\sigma \in \Gamma(x)) \\ cut(link(q, t_{min}(x,\sigma))), \text{ if } (\sigma \in \Sigma_{uo}) \wedge (\sigma \in \Gamma(x)) \\ undefined, \text{ otherwise.} \end{cases}$$

- *The function removal,* $rem : \mathcal{Q} \times \mathbb{N} \rightarrow \mathcal{Q}$ *is a mapping where, for all* $q =$

$q_1 q_2 \cdots q_k \in \mathcal{Q}$,

$$rem(q,y) := \begin{cases} cut(q_2 \cdots q_k),\ if\ (y = 1) \\ link(q_1 \cdots q_{y-1}, q_{y+1} \cdots q_k),\ if\ (1 < y < k) \\ cut(q_1 \cdots q_{k-1}),\ if\ (y = k) \\ undefined,\ otherwise. \end{cases}$$

• *The observation channel index function is the mapping* $och : \Sigma_o \to \{1, ..., m\}$ *defined as* $och(\sigma) = i$, *for all* $\sigma \in \Sigma_{o,i}$ *and* $i \in \{1, ..., n\}$.

According to Definition 4.4, string $link(q, p)$ is obtained by concatenating strings q and p in the usual manner except when the last element of q and the first element of p are numbers; in this case, these elements are added. String $cut(q)$ is formed from q either by removing the largest prefix that is composed with numbers only, or is set as 0, when q is solely formed by numbers. Function $add()$ will be used to define the second component of the state reached from a state (x, q) of G_e by a transition that is labeled by an event $\sigma \in \Sigma$, and, thus, string $add(q, x, \sigma)$ is formed by concatenating q with the minimal activation time $t_{min}(x, \sigma)$ and, in the case when $\sigma \in \Sigma_o$, by also concatenating with σ. Function $rem()$ will be used to define the second component of the state reached by a transition labeled by an event in $(\Sigma_o^s \cup \Sigma_o^l)$ (*i.e.*, a transition that represents either a successful observation or a loss of observation), and, consequently, string $rem(q, y)$ is formed from q by removing its y-th element, which corresponds to the event whose observation has been finished or lost. Finally, $och(\sigma)$ is equal to the index i associated with the observation channel used to transmit the occurrences of event σ.

We now present an algorithm for the construction of automaton G_e. The reader may find useful to follow the algorithm with the help of Example 4.5.

Algorithm 4.1 (Construction of automaton G_e)

Inputs:

• $G = (X, \Sigma, f, \Gamma, x_0)$*: automaton that models the plant of the NDESWTS;*

• $\Sigma_{o,i}$ *and* T_i, *for all* $i \in \{1, ..., m\}$*: sets of observable events and delay upper bounds of each observation channel;*

• Σ_{lo}*: set of events subject to losses of observations;*

- $t_{min}(x, \sigma)$, $\forall x \in X$ *and* $\forall \sigma \in \Gamma(x)$*: minimal activation times.*

Output:

- $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e})$.

Step 1: Define the initial state $x_{0_e} = (x_0, 0)$ *and* $X_e = \emptyset$*;*

Step 2: Define event sets Σ_o^s, Σ_o^l *and* Σ_e *according to Equations* (4.1) *and* (4.2)*;*

Step 3: Create a FIFO queue $F = [x_{0_e}]$*;*

Step 4: While $F \neq \emptyset$ *do:*

- *4.1:* $(x, q) \leftarrow head(F)$ *and* $dequeue(F)$.
- *4.2:* $X_e \leftarrow X_e \cup \{(x, q)\}$*;*
- *4.3: Let* $q = q_1\, q_2 \cdots q_k$*. Define the set of indexes* $I_o = \{j \in \{1, \ldots, k\} : q_j \in \Sigma_o\}$*;*
- *4.4: If* $I_o \neq \emptyset$*, then,* $\forall y \in I_o$*, set* $min_{et}(y)$ *as the sum of the real numbers on the right of* q_y *in* q*;*
- *4.5: For all* $\sigma \in \Gamma(x)$*:*

(a) $\tilde{x}_e = f_e((x, q), \sigma) =$

$$\begin{cases} (f(x, \sigma), add(q, x, \sigma)), \text{ if } (I_o = \emptyset) \vee [(min_{et}(y) + \\ \qquad\qquad + t_{min}(x, \sigma)) < T_{och(q_y)}, \forall y \in I_o]; \\ \text{undefined, otherwise.} \end{cases}$$

(b) If $\tilde{x}_e$ *is defined and* $(\tilde{x}_e \notin X_e) \wedge (\tilde{x}_e \notin F)$*, then* $enqueue(F, \tilde{x}_e)$*;*

- *4.6: For each observation channel* och_i, $i = 1, ..., m$*, form set* $Y_i = \{j \in I_o : q_j \in \Sigma_{o,i}\}$*. If* $Y_i \neq \emptyset$*, then:*

(a) Compute $y = min(Y_i)$*;*

(b) Define $\hat{x}_e = f_e((x, q), \psi(q_y)) = (x, rem(q, y))$*;*

(c) If $q_y \in \Sigma_{lo}$*, then define* $f_e((x, q), q_y l) = \hat{x}_e$*;*

(d) If $(\hat{x}_e \notin X_e) \wedge (\hat{x}_e \notin F)$*, then* $enqueue(F, \hat{x}_e)$*;*

Step 5: Define $\Gamma_e(x_e) = \{\sigma \in \Sigma_e : f_e(x_e, \sigma)$ *is defined*$\}$*, for all* $x_e \in X_e$.

Example 4.5 *Consider the NDESWTS depicted in Figures 4.2(a,b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$ *and* $\Sigma_{lo} = \{\beta\}$*. Using Algorithm 4.1 with inputs* G,

$\Sigma_{o,1} = \{\alpha\}$, $\Sigma_{o,2} = \{\beta, \gamma\}$, $T_1 = 0.9s$, $T_2 = 0.3s$, $\Sigma_{lo} = \{\beta\}$, *and* t_{min}, *defined according to Figure 4.2(b), we obtain automaton* G_e *depicted in Figure 4.7, whose construction can be explained as follows.*

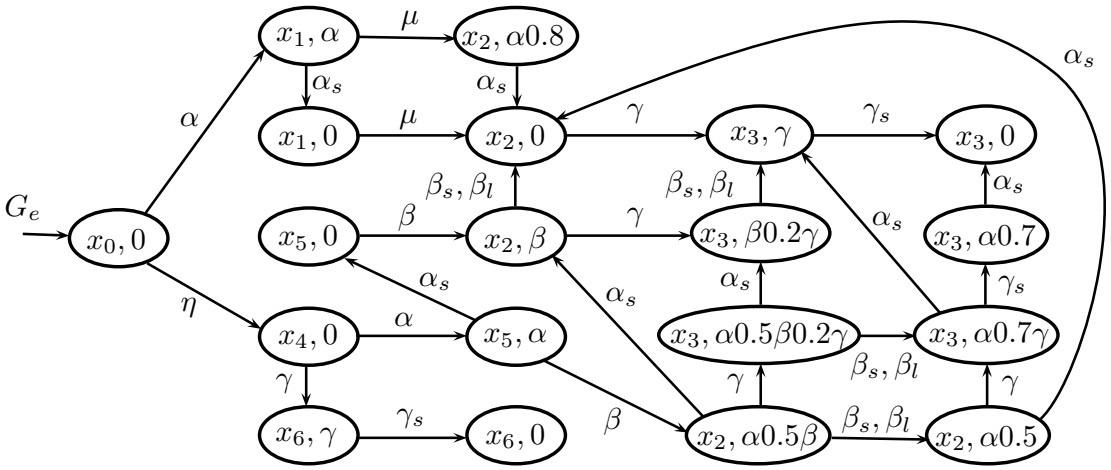


Figure 4.7: Automaton G_e obtained by using Algorithm 4.1 in Example 4.5.

In Step 1, we define the initial state of automaton G_e *as* $x_{0_e} = (x_0, 0)$, *where the second component indicates that no observation is being transmitted to the supervisor at state* $(x_0, 0)$. *In Step 2, we create the sets of events* $\Sigma_e = \{\alpha, \beta, \gamma, \mu, \eta, \alpha_s, \beta_s, \gamma_s, \beta_l\}$, $\Sigma_o^s = \{\alpha_s, \beta_s, \gamma_s\}$ *and* $\Sigma_o^l = \{\beta_l\}$. *In Step 3, we create a queue of states* $F = [(x_0, 0)]$, *and, in the sequence, we repeat Step 4 until* F *becomes empty.*

In Steps 4.1 and 4.2, we set $(x, q) = (x_0, 0)$, $F = [\,]$ *and* $X_e = \{(x_0, 0)\}$. *In Step 4.3, we create the set of indexes* I_o *that records the indexes of the events belonging to* Σ_o *inside* q. *Thus,* $I_o = \emptyset$, *since* $q = 0$. *The next step, Step 4.4, should be skipped since* $I_o = \emptyset$. *In Step 4.5 (resp. Step 4.6), we define the transitions from state* (x, q), *labeled by events in* Σ *(resp.* Σ_o^s *or* Σ_o^l*) associated with the occurrences of events in the plant (resp. successful or lost observations). Since* $\Gamma(x_0) = \{\alpha, \eta\}$ *and* $q = 0$, *we define, in Step 4.5, two transitions from state* $(x_0, 0)$ *labeled by events* α *and* η, *which define new states* (x_1, α) *and* $(x_4, 0)$*; thus* $F = [(x_1, \alpha), (x_4, 0)]$. *Notice that, event* α *(resp.* η*) is added (resp. not added) to the second component of the reached state because it is an observable (resp. unobservable) event. Finally, since* $I_o = \emptyset$, *then sets* Y_i, *to be formed in Step 4.6 for each observation channel* och_i, *are also empty. Therefore, no transition will be defined in Step 4.6.*

Assume that after some iterations of Step 4, state $(x, q) = (x_5, \alpha)$ *is the first*

state of queue F. Then, in Steps 4.3 and 4.4, we obtain $I_o = \{1\}$ and $min_{et}(1) = 0$. In Step 4.5, we define a unique transition labeled by β, because $\Gamma(x_5) = \{\beta\}$, and since $min_{et}(1) + t_{min}(x_5, \beta) = 0.5 < T_1 = 0.9$, β can occur before the observation of α. Thus the state reached through this transition will be $(f(x_5, \beta), add(\alpha, x_5, \beta)) = (x_2, \alpha 0.5\beta)$. Finally, in step 4.6, we define the transition labeled by α_s to state $(x_5, 0)$, that represents the successful observation of α.

Let us now assume that $(x, q) = (x_2, \alpha 0.8)$ at the beginning of Step 4. Notice that although $\gamma \in \Gamma(x_2)$, a transition labeled by γ from state $(x_2, \alpha 0.8)$ cannot be defined since the time elapsed between the occurrences of α and γ is larger than the delay of the observation of α.

To conclude the example, let us consider state $(x, q) = (x_3, \alpha 0.5\beta 0.2\gamma)$. Notice that although γ is in q, its observation cannot be recorded by the supervisor since event β has occurred before γ, β has not been observed yet and both β and γ are transmitted through the same observation channel.

The following results concern automaton G_e obtained by Algorithm 4.1.

Lemma 4.1 *Let $w \in L(G_e)$ and define $(x, q) = f_e(x_{0_e}, w)$. Then:*

(a) $x = f(x_0, P_e(w))$;

(b) $q = 0$ if, and only if, every event σ in w that belongs to Σ_o has its occurrence either successfully transmitted (σ_s) or lost (σ_l) in w. Otherwise, $q = q_1 q_2 \dots q_k \in \mathcal{Q}$, where $q_1 \in \Sigma_o$ and every $q_y \in \Sigma_o$, $y \in \{1, 2, \dots, k\}$, corresponds to one occurrence of event q_y in w whose occurrence is still being transmitted, with $min_{et}(y)$ (stated in Step 4.4 of Algorithm 4.1) equal to the minimal time interval elapsed since the occurrence of q_y in the plant.

Proof: The proof is done by induction in the length of the strings $w \in L(G_e)$.

Basis step. According to Step 1 of Algorithm 4.1, the initial state of G_e is equal to $x_{0_e} = (x_0, 0)$. Thus, for $w = \varepsilon$, $f_e(x_{0_e}, w) = (x_0, 0)$, which agrees with the facts that: *(a)* $f(x_0, P_e(\varepsilon)) = x_0$, and *(b)* there is no event in w whose occurrence has not been transmitted.

Induction hypothesis. For all $w \in L(G_e)$, such that $\|w\| \leq p$, $f_e(x_{0_e}, w) = (f(x_0, P_e(w)), q)$, where $q = 0$ if, and only if, every event σ in w that belongs to Σ_o

has its occurrence either successfully transmitted (σ_s) or lost (σ_l) in w. Otherwise, $q = q_1 q_2 \dots q_k \in \mathcal{Q}$ where $q_1 \in \Sigma_o$ and every $q_y \in \Sigma_o$, $y \in \{1, 2, \dots, k\}$, corresponds to one occurrence of event q_y in w that has not been observed yet, with $min_{et}(y)$ being equal to the minimal time interval elapsed since the occurrence of q_y in the plant.

Inductive step. Consider a string $w\sigma \in L(G_e)$ such that $\|w\| = p$ and $\sigma \in \Sigma_e$. We will prove initially item *(a)* and, after that, item *(b)*.

(a) Notice that, according to the induction hypothesis, the first component of state $f_e(x_{0_e}, w)$ is equal to $f(x_0, P_e(w))$. Let us first consider the case when $\sigma \in \Sigma$. Then, according to Step 4.5 of Algorithm 4.1, $\sigma \in \Gamma(f(x_0, P_e(w)))$ and the first component of the reached state is equal to $f(f(x_0, P_e(w)), \sigma) = f(x_0, P_e(w)\sigma) = f(x_0, P_e(w\sigma))$. Let us now consider the case when $\sigma \in (\Sigma_o^s \cup \Sigma_o^l)$. Since, according to Steps 4.6(b) and 4.6(c) of Algorithm 4.1, the transitions of G_e labeled by events in $(\Sigma_o^s \cup \Sigma_o^l)$ do not modify the first component of the state, the first component of $f_e(x_{0_e}, w\sigma)$ is equal to $f(x_0, P_e(w))$, which is equal to $f(x_0, P_e(w\sigma))$ since $P_e(w) = P_e(w\sigma)$.

(b) Let q denote the second component of state $f_e(x_{0_e}, w)$. Then, according to the induction hypothesis, q satisfies part *(b)* of the lemma statement with respect to string w. According to Algorithm 4.1, the second component of the state reached from state $f_e(x_{0_e}, w)$ by a transition labeled by an event $\sigma \in \Sigma_e$ is determined as follows:

(i) If $\sigma \in \Sigma$, then, according to Step 4.5(a), the second component of the reached state is $add(q, x, \sigma)$, where, according to Definition 4.4, function *add* links, to the right of q, either string $t_{min}(x, \sigma)\sigma$, if $\sigma \in \Sigma_o$, or $t_{min}(x, \sigma)$, if $\sigma \in \Sigma_{uo}$, and also removes the largest prefix formed only with non-negative real numbers. In this case, the second component of the state reached from $f_e(x_{0_e}, w)$ by the transition labeled by σ will be as follows: $t_{min}(x, \sigma)$ must be added to the right of q after the occurrence of σ in the plant with a view to enforcing that $min_{et}(y)$, defined according Step 4.4 of Algorithm 4.1, be equal to the minimal time interval elapsed since the occurrence of the y-th event of the second component of the reached state; in addition, σ has to be added to q when $\sigma \in \Sigma_o$, since its occurrence has not been observed in $w\sigma$, whereas, in

the case when $\sigma \in \Sigma_{uo}$, nothing else has to be added to q.

(ii) If $\sigma \in \Sigma_o^s$ or $\sigma \in \Sigma_o^l$, then, according to Steps 4.6(b) and 4.6(c), respectively, the second component of the reached state is $rem(q, y)$, where y, computed in Step 4.6(a), is the index of the first occurrence of event $\psi^{-1}(\sigma)$ in q and, according to Definition 4.4, function rem removes q_y from q, and also removes the largest prefix formed only with non-negative real numbers. Thus, the occurrence of an event σ belonging to either Σ_o^s or Σ_o^l represents, respectively, either the successful or the loss of the observation of an event in w, namely, it models either the successful or the loss of the observation of event q_y stored in q. Thus, it is straightforward to conclude that we must remove q_y from q to obtain the second component of the reached state, as done by using function rem in Algorithm 4.1. In addition, notice that, functions add and rem are defined using function cut, which removes, from $q \in \mathcal{Q}$, the largest prefix formed only with non-negative real numbers.

Finally, notice that, in both cases, *(i)* and *(ii)*, functions add and rem guarantee that the first element of the second component of the reached state belongs to Σ_o, if the second component of the reached state has at least one element belonging to Σ_o, or that the second component of the reached state is equal to 0, otherwise. In both cases, the lemma statement holds true, and the proof is complete. ■

Lemma 4.2 $L(G_e) = E(L(G))$.

Proof: Let us initially consider the case when $\Sigma_{lo} = \emptyset$, that is, there is no event subject to loss of observations. Thus, $\Sigma_o^l = \emptyset$ and, according to Definitions 4.2 and 4.3, $E(L(G)) = D_d(L(G))$. The proof that $L(G_e) = D_d(L(G))$, in the case when $\Sigma_{lo} = \emptyset$, is done by induction in the length of strings $w \in \Sigma_e^*$.

Basis step. Let $w = \varepsilon$. Then, $w \in D_d(L(G))$ since $P_e(\varepsilon) = \varepsilon \in L(G)$ and ε satisfies Conditions 2 and 3 of Definition 4.1. In addition, we can also conclude that $w \in L(G_e)$ since the initial state of G_e is defined (which is equal to $(x_0, 0)$).

Induction hypothesis. For all extended string $w \in \Sigma_e^*$ such that $\|w\| \leq p$, $w \in L(G_e) \Leftrightarrow w \in D_d(L(G))$.

Inductive step. Let $w\sigma \in \Sigma_e^*$ be such that $\|w\| = p$ and $\sigma \in \Sigma_e$. It can be seen, from Definition 4.1, that if $w \notin D_d(L(G))$, then $w\sigma \notin D_d(L(G))$. In addition,

since $L(G_e)$ is, by definition, prefix-closed, if $w \notin L(G_e)$, then $w\sigma \notin L(G_e)$. As a consequence of the induction hypothesis, we can only consider the case when $w \in L(G_e)$ and $w \in D_d(L(G))$. The last statement implies, according to Definition 4.1, that there exists $s \in L(G)$ such that $s = P_e(w)$ and that w satisfies Conditions 2 and 3 of Definition 4.1. We will first consider the case when $\sigma \in \Sigma$, and, in the sequel, the case when $\sigma \in \Sigma_o^s$.

(i) $\sigma \in \Sigma$. In this case, Condition 3 of Definition 4.1 is satisfied for $w\sigma$ since it is satisfied for w. Regarding Condition 1 of Definition 4.1, notice that transitions from state $f_e(x_{0_e}, w)$ labeled by events in Σ are defined, in Step 4.5 of Algorithm 1, for those events that belong to $\Gamma(f(x_0, s))$ since, according to statement *(a)* of Lemma 4.1, the first component of state $f_e(x_{0_e}, w)$ is equal to $f(x_0, s)$. This agrees with the fact that, since $P_e(w\sigma) = s\sigma$, $w\sigma$ satisfies Condition 1 of Definition 4.1 only for string $s\sigma$, which implies that $s\sigma$ must be in $L(G)$ for $w\sigma$ to be in $D_d(L(G))$, or equivalently, σ must be in $\Gamma(f(x_0, s))$. In order to verify if $w\sigma$ satisfies Condition 2, let q denote the second component of state $f_e(x_{0_e}, w)$, and consider the problem of evaluating the possibility of occurrence of event $\sigma \in \Sigma$ before the observation of one of the events belonging to Σ_o that form q. According to Step 4.5(a), this evaluation is made by checking if $q = 0$ or, when $q \neq 0$, by comparing, for every $q_y \in \Sigma_o$ that forms q, the minimal time elapsed since the occurrence of q_y with the delay bound of the channel that transmits the occurrences of q_y to the supervisor. Thus, the transition labeled with an event $\sigma \in \Gamma(f(x_0, s))$ from state $f_e(x_{0_e}, w)$ is defined if, and only if, either $q = 0$ or, for every $q_y \in \Sigma_o$ that forms q, the delay bound $T_{och(q_y)}$ is bigger than $min_{et}(y) + t_{min}(f(x_0, P_e(w)), \sigma)$. Notice that, in accordance with statement *(b)* of Lemma 4.1, verifying this condition is equivalent to checking if every event in Σ_o, that has occurred in $w\sigma$ and whose observation has not occurred, satisfies Equation (4.3). In addition, since w satisfies Condition 2, every event in w whose occurrence has been observed in w also satisfies Equation (4.3). Therefore, we can conclude that, when $\sigma \in \Sigma$, $w\sigma \in L(G_e) \Leftrightarrow w\sigma \in D_d(L(G))$.

(ii) $\sigma \in \Sigma_o^s$. In this case, $P_e(w\sigma) = P_e(w)$, which implies that $w\sigma$ satisfies Condition 1 of Definition 4.1 for $s \in L(G)$. Moreover, $w\sigma$ also satisfies Condition 2

of Definition 4.1 since it is satisfied for w. Thus, it remains to check if Condition 3 holds true for string $w\sigma$. In order to do so, consider the possibility of creating a transition from state $f_e(x_{0_e}, w)$, labeled by event $\sigma \in \psi(\Sigma_{o,i})$, carried out in Step 4.6, which is repeated for each observation channel och_i, $i = 1, \ldots, m$. In Step 4.6, the set of indexes Y is computed with respect to the second component of state $f_e(x_{0_e}, w)$, denoted by q, and set $\Sigma_{o,i}$. Notice that, in accordance with Lemma 4.1, $w\sigma$ satisfies Equation (4.4) if, and only if, there exists $\psi^{-1}(\sigma)$ in q. Thus, when Y is nonempty, index $y = min(Y)$, computed in Step 4.6(a), determines event q_y that corresponds to the first event in q whose occurrence is transmitted through channel och_i. Consequently, $\psi(q_y)$ is the unique event in $\psi(\Sigma_{o,i})$ such that $w\psi(q_y)$ satisfies Equations (4.4) and (4.5), and, according to Step 4.6(b), it is also the unique event in $\psi(\Sigma_{o,i})$ that is used to create a new transition from state $f_e(x_{0_e}, w)$. Therefore, it can be concluded that, when $\sigma \in \Sigma_o^s$, $w\sigma \in L(G_e) \Leftrightarrow w\sigma \in D_d(L(G))$.

Let us now consider the case when $\Sigma_{lo} \neq \emptyset$. As stated in [62], in this case, an automaton that generates the dilation of $D_d(L(G))$, that is $E(L(G)) = D_l(D_d(L(G)))$, can be constructed from an automaton that generates $D_d(L(G))$ by adding to all transitions labeled with events in $\Sigma_o^s \cap \psi(\Sigma_{lo})$, parallel transitions labeled with the corresponding events in Σ_o^l. Notice that, according to Step 4.6(c), a parallel transition labeled by $\sigma_l \in \Sigma_o^l$ is defined whenever a transition labeled by an event $\sigma_s \in \Sigma_o^s \cap \psi(\Sigma_{lo})$ is defined in Step 4.6(b), which completes the proof. ■

Theorem 4.1 *Let $L \subseteq L(G)$. Then, $E(L) = P_e^{-1}(L) \cap L(G_e)$, where $P_e : \Sigma_e^* \rightarrow \Sigma^*$.*

Proof:

($\subseteq$) Let us consider a string $w \in E(L)$. Then, $w \in E(L(G))$ since, according to Definitions 4.1, 4.2 and 4.3, $E(L) \subseteq E(L(G))$. Thus, using Lemma 4.2, we can conclude that $w \in L(G_e)$. In addition, because $E(L) = D_l(D_d(L))$ is formed from $D_d(L)$ by adding new strings obtained, from the strings in $D_d(L)$, by replacing some events in $\Sigma_o^s \cap \psi(\Sigma_{lo})$ with events in Σ_o^l, we can see that either *(i)* $w \in D_d(L)$ or *(ii)* there exists $t \in D_d(L)$ such that $w \in (D_l(t) \setminus \{t\})$ and $P_e(w) = P_e(t)$. As a consequence $w \in P_e^{-1}(L)$ since, according to Condition 1 of Definition 4.1, there exists $s \in L$ such that, in case *(i)* $s = P_e(w)$ and, in case *(ii)*, $s = P_e(t) = P_e(w)$. Therefore, $w \in P_e^{-1}(L) \cap L(G_e)$.

($\supseteq$) Consider now a string $w \in P_e^{-1}(L) \cap L(G_e)$, and define string $s = P_e(w)$. Thus, $w \in L(G_e)$ and $w \in P_e^{-1}(L)$, where the last inclusion relation implies that $s \in L$. Let us now analyzing $w \in L(G_e)$. In this case, according to Lemma 4.2, $w \in E(L(G))$, which implies, according to Definitions 4.2 and 4.3, that either *(i)* $w \in D_d(L(G))$, and, in this case, $w \in D_d(s) \subseteq E(L)$, or *(ii)* there exists $t \in D_d(L(G))$ such that $w \in (D_l(t) \setminus \{t\})$ and $P_e(w) = P_e(t)$, and thus, $t \in D_d(s)$, which also implies that $w \in E(s) \subseteq E(L)$. Therefore, in both cases, *(i)* and *(ii)*, $w \in E(L)$. ■

Remark 4.1 (size of the state space of G_e) *Let us define the following variables: $T = max\{T_i : i \in \{1, \ldots, m\}\}$, $t = min\{t_{min}(x, \sigma) : ((x, \sigma) \in X \times \Sigma) \wedge (\sigma \in \Gamma(x))\}$, and $\mathcal{T} = max\{z \in \mathbb{Z} : z < T/t\}$, where $\mathbb{Z}$ is the set of integer numbers.*

In order to compute the maximal number of states of G_e, let us assume that $link(q, p)$ only concatenates q and p, i.e., it does not add the last element of q with the first element of p when they are real numbers. In this case, every state of G_e has either (i) the form (x, q_0) where $x \in X_g$ and $q_0 \in \Sigma_o \cup \{0\}$, or (ii) the form $(x, q_0 e_1 e_2 \ldots e_k)$ where $x \in X_g$, $q_0 \in \Sigma_o$ and, for $j = 1, 2, \ldots, k$, either e_j is a real number (which is the minimal activation time that corresponds to either an event in Σ_{uo} or an event in Σ_o whose transmission has been finished), or e_j is a real number concatenated with an event belonging to Σ_o (which corresponds to a minimal activation time and its associated event in Σ_o whose occurrence is still being transmitted). In the worst case, G_e will have $|X_g| \cdot (|\Sigma_o| + 1)$ states with the form (i), and $|X_g|^2 \cdot |\Sigma_o| \cdot (|\Sigma| + |\Sigma_o|)^k$ states with the form (ii). In addition, by assuming that all minimal activation times are equal to t and all maximal observation delays are equal to T, it can be seen that $k \leq \mathcal{T}$ since, if $k > \mathcal{T}$, then the maximal observation delay of event q_0 is violated. Thus, we can conclude that, in the worst case,

$$|X_e| = |X_g| \cdot (|\Sigma_o| + 1) + |X_g|^2 \cdot |\Sigma_o| \cdot \sum_{k=1}^{\mathcal{T}} (|\Sigma| + |\Sigma_o|)^k.$$

Therefore, $|X_e|$ is $O\left(|X_g|^2 \cdot |\Sigma_o| \cdot (|\Sigma| + |\Sigma_o|)^{\mathcal{T}}\right)$. Finally, notice that the number of states in the worst case may only decrease if we compute $link(q, p)$ as stated by Definition 4.4, i.e., by adding the last element of q with the first element of p when they are real numbers.

In the design of networked supervisors for a given admissible language $K \subset L(G)$,

we will use the extended language $E(K)$ in order to model the execution of the strings belonging to K in the general case, *i.e.*, when we assume that delays or loss of observations may occur. Besides considering the performance of the compensated system in the general case, we also intend to take into account the performance of the compensated system under nominal operating conditions, *i.e.*, assuming no observation loss and only negligible delays, which, in practice, corresponds to the case when all delays of observations are smaller than the time intervals between the event occurrences. To this end, we characterize, in the following definition, the sublanguage of $E(K)$ that models the execution of the strings belonging to a language K under nominal operating conditions.

Definition 4.5 (null extension function) *The null extension function is the mapping* $E_n : L(G) \rightarrow 2^{D_d(L(G))}$ *defined as:*

(i) $E_n(\varepsilon) := \{\varepsilon\}$;

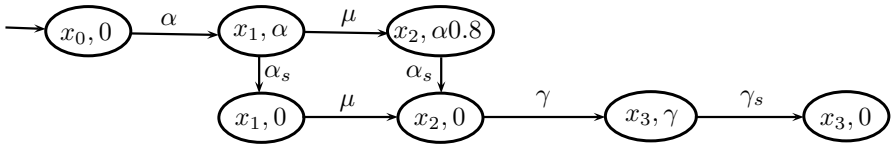
(ii) $E_n(s) := \{\sigma_1 \ldots \sigma_k \in D_d(s) : (\nexists j \in \{1, \ldots, k-1\})[(\sigma_j \in \Sigma_o) \wedge (\sigma_{j+1} \neq \psi(\sigma_j))]\}$, $\forall s \in L(G) \setminus \{\varepsilon\}$.

The extension of E_n *to domain* $2^{L(G)}$ *is defined as* $E_n(L) := \bigcup_{s \in L} E_n(s)$.

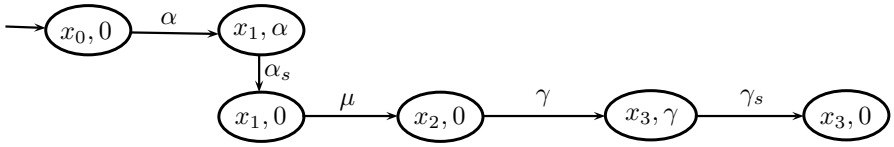
According to Definition 4.5, for a string $s \in L(G)$, $E_n(s)$ is formed by those extended strings in $D_d(s)$ in which every occurrence of an event $\sigma \in \Sigma_o$ is followed by the occurrence of event $\sigma_s \in \Sigma_o^s$, where the latter models the successful observation of the former. The following example illustrates the null extension function.

Example 4.6 *Consider the NDESWTS depicted in Figures 4.2(a,b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$ *and* $\Sigma_{lo} = \{\beta\}$. *Figure 4.8(a) shows the subautomaton of* G_e *(which was constructed in Example 4.5) that generates those extended strings that form* $E(\overline{\{\alpha\mu\gamma\}})$. *It can be seen, with the help of Figure 4.8(a), that the strings in* $\overline{\{\alpha\mu\alpha_s\gamma\gamma_s\}}$ *correspond to those strings in* $E(\overline{\{\alpha\mu\gamma\}})$ *that violate Condition (ii) of Definition 4.5, since event* μ *occurs before the observation of event* α, *which is denoted by* α_s. *Thus, we can conclude that the null extended language* $E_n(\overline{\{\alpha\mu\gamma\}})$ *is equal to the language generated by the automaton depicted in Figure 4.8(b).*

Notice that, for every string $s \in L(G)$, $E_n(s)$ *will have just one element if the last event of* s *belongs to* Σ_{uo}, *for instance,* $E_n(\alpha\mu) = \{\alpha\alpha_s\mu\}$, *whereas* $E_n(s)$ *will have*



(a)



(b)

Figure 4.8: Automata whose generated languages are $E(\overline{\{\alpha\mu\gamma\}})$ (a) and $E_n(\overline{\{\alpha\mu\gamma\}})$ (b).

just two elements in the case when the last event of s belongs to Σ_o, for example, $E_n(\alpha\mu\gamma) = \{\alpha\alpha_s\mu\gamma, \alpha\alpha_s\mu\gamma\gamma_s\}$.

The following algorithm can be used to construct automata whose generate languages are the extension and the null extension of a given prefix-closed sublanguage of $L(G)$.

Algorithm 4.2 (Construction of automata H_e and H_n)

Inputs:

- $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e})$*: automaton whose generated language is* $E(L(G))$*;*
- $H = (X_h, \Sigma, f_h, \Gamma_h, x_{0_h})$*: automaton whose generated language is* $K \subseteq L(G)$*;*
- Σ_o *and* Σ_{uo}*: sets of observable and unobservable events of automaton* G*, respectively.*

Outputs:

- H_e*: automaton whose generated language is* $E(K)$*.*
- H_n*: automaton whose generated language is* $E_n(K)$*.*

Step 1: Compute $H_e := H \| G_e$

Step 2: Define automaton $A := (\{y_0, y_1\}, \Sigma_e, f_a, \Gamma_a, y_0)$*, where: (i)* $\Gamma_a(y_0) = \Sigma$ *and* $\Gamma_a(y_1) = \Sigma_o^s$*, and; (ii)* $f_a(y_0, \sigma) = y_0$*, if* $\sigma \in \Sigma_{uo}$*,* $f_a(y_0, \sigma) = y_1$*, if* $\sigma \in \Sigma_o$*, and* $f_a(y_1, \sigma) = y_0$*, if* $\sigma \in \Sigma_o^s$*.*

Step 3: Compute $H_n := A \| H_e$*.*

Example 4.7 *Consider the NDESWTS depicted in Figures 4.2(a,b), where* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$ *and* $\Sigma_{lo} = \{\beta\}$, *and the language* K *generated by automaton* H, *depicted in Figure 4.9(a). Consider also automaton* G_e *computed in Example 4.5, which is depicted in Figure 4.7. Automaton* H_e, *computed in Step 1 of Algorithm 4.2, is depicted in Figure 4.9(b). Notice that, automaton* A *computed in Step 2 of Algorithm 4.2 has always the state transition diagram, which is presented in Figure 4.9(c). Finally, automaton* H_n, *computed in Step 3 of Algorithm 4.2 is depicted in Figure 4.9(d).*

The following proposition asserts the correctness of Algorithm 4.2.

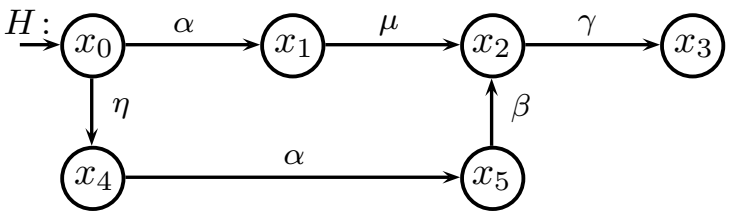
Proposition 4.1 *Let* $K \subseteq L(G)$, *and let* H *and* G_e *be automata such that* $L(H) = K$ *and* $L(G_e) = E(L(G))$. *Consider automata* H_e *and* H_n *computed by using Algorithm 4.2 with inputs* G_e, H, Σ_o *and* Σ_{uo}. *Then, (a)* $L(H_e) = E(K)$, *and (b)* $L(H_n) = E_n(K)$.

Proof: (a) $L(H_e) = L(H \| G_e) = P_e^{-1}(L(H)) \cap L(G_e)$, which implies, according to Theorem 4.1, that $L(H_e) = E(K)$.

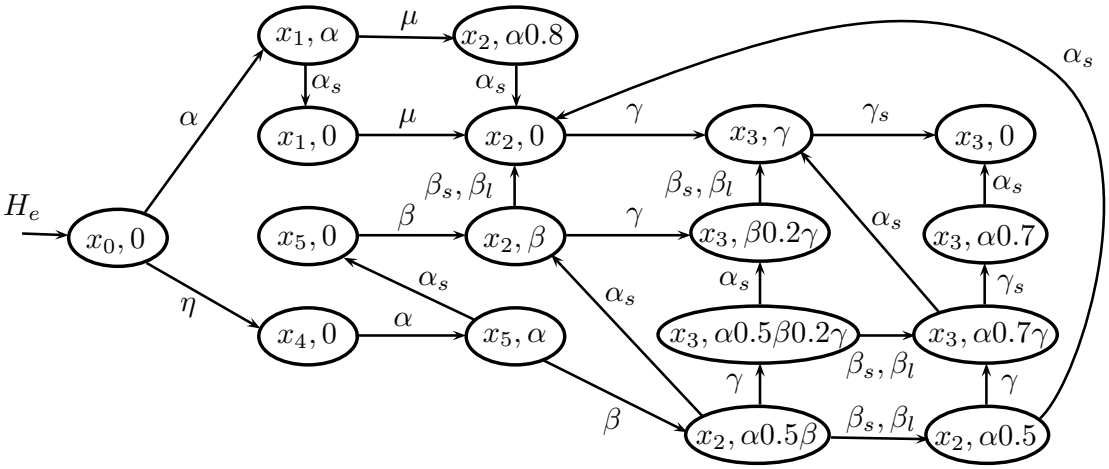
(b) Since $L(H_e) = E(K)$ and $H_n = A \| H_e$, it is straightforward to see that, if $K = \emptyset$, then $L(H_n) = E_n(K) = \emptyset$, and, otherwise, $\varepsilon \in L(H_n)$ and $\varepsilon \in E_n(K)$. In addition, notice that $E(K)$ can be partitioned as $E(K) = D_d(K) \dot{\cup} (E(K) \setminus D_d(K))$, where $(E(K) \setminus D_d(K))$ is formed with those extended strings that have at least one event belonging to Σ_o^l, *i.e.*, those extended strings that model the cases when loss of observations occurs. Thus, since automaton A does not allow the occurrence of events in Σ_o^l and $L(H_n) = L(A) \cap L(H_e)$, we can infer that $L(H_n) \subseteq D_d(K)$. Therefore, it remains to be verified if, $\forall \sigma_1 \ldots \sigma_k \in D_d(K)$ with $k \geq 1$, string $\sigma_1 \ldots \sigma_k \in L(H_n)$ if, and only if, it satisfies the following condition

$$(\nexists j \in \{1, \ldots, k-1\})[(\sigma_j \in \Sigma_o) \wedge (\sigma_{j+1} \neq \psi(\sigma_j))] \tag{4.6}$$

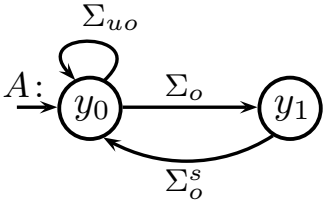
To do so, let us first consider a string $\sigma_1 \ldots \sigma_k \in D_d(K)$ that satisfies (4.6). Notice that $\sigma_1 \ldots \sigma_k \in L(H_e)$ since $L(H_e) = E(K) \supseteq D_d(K)$. In addition, according to Condition 3 of Definition 4.1 (Equation (4.4)), the first event of every string in $D_d(K)$ belongs to Σ, which implies that either $\sigma_1 \in \Sigma_o$ or $\sigma_1 \in \Sigma_{uo}$, and, additionally, according to (4.6), all event in $\sigma_1 \ldots \sigma_{k-1}$ that belongs to Σ_o is followed by an



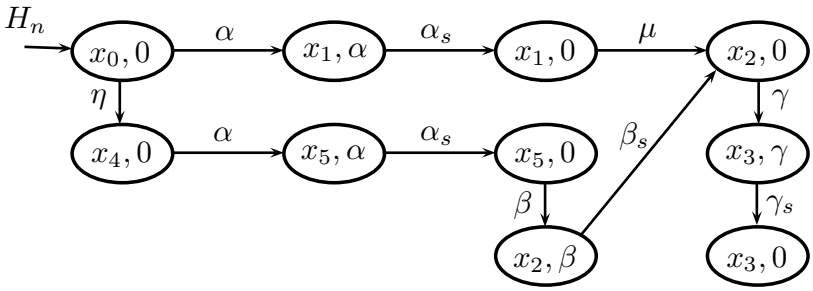
(a)



(b)



(c)



(d)

Figure 4.9: Automaton whose generated language is K (a), and automata H_e (b), A (c) and H_n (d) computed using Algorithm 4.2 in Example 4.7.

event belonging to Σ_o^s. Thus, we can conclude, from Step 2 of Algorithm 4.2, that $\sigma_1 \dots \sigma_k \in L(A)$, and, consequently, $\sigma_1 \dots \sigma_k \in L(H_n)$.

Let us now suppose a string $\sigma_1 \dots \sigma_k \in L(H_n)$ that does not satisfy (4.6), and

let j denote the smaller index for which $(\sigma_j \in \Sigma_o) \wedge (\sigma_{j+1} \neq \psi(\sigma_j))$. Thus, since all transitions that are active in A after the occurrence of an event in Σ_o are labeled by events in Σ_o^s, we can infer that $\sigma_{j+1} \in \Sigma_o^s \setminus \{\psi(\sigma_j)\}$. However, since j is the smaller index for which $(\sigma_j \in \Sigma_o) \wedge (\sigma_{j+1} \neq \psi(\sigma_j))$, all events in $\sigma_1 \ldots \sigma_{j-1}$ that belong to Σ_o were observed in $\sigma_1 \ldots \sigma_{j-1}$, and, consequently, extended string $\sigma_1 \ldots \sigma_k$ does not satisfy Condition 3 of Definition 4.1 (Equation (4.4)), which implies that $\sigma_1 \ldots \sigma_k \notin D_d(K)$ and, thus, $\sigma_1 \ldots \sigma_k \notin L(H_n)$, which is a contradiction and completes the proof. ■

4.3 Supervisory Control of NDESWTS

4.3.1 Networked Supervisory Control Problem

Because of the presence of delays and loss of observations, the feedback structure for networked supervisory control is no longer that presented in Figure 2.8 (Subsection 2.5.2) for supervisory control under partial observation. As seen in Figure 4.10,

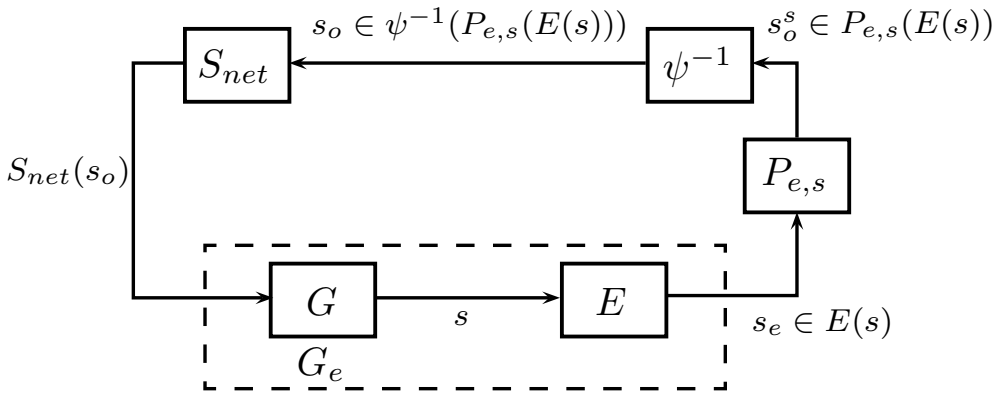


Figure 4.10: Feedback structure for the networked supervisory control problem under delays and loss of observations.

the observed language is no longer $P_o(L(G))$ but $\psi^{-1}(P_{e,s}(E(L(G))))$, namely, when a string s is generated by the plant, the networked supervisor S_{net} makes its decision based on one of the strings in $\psi^{-1}(P_{e,s}(E(s)))$. Thus, there may exist ambiguities regarding the control action, which requires that three languages associated with the compensated system S_{net}/G be taken into consideration.

Definition 4.6 (languages associated with S_{net}/G)

(a) Extended language $L_e(S_{net}/G)$, which is formed solely by those extended strings belonging to $E(L(G))$ that can be generated by the compensated system S_{net}/G with or without the occurrence of delays and loss of observations, is recursively defined as:

(i) $\varepsilon \in L_e(S_{net}/G)$;

(ii) $\forall s_e \in \Sigma_e^*$ *and* $\forall \sigma \in \Sigma_e$, $s_e\sigma \in L_e(S_{net}/G) \Leftrightarrow s_e \in L_e(S_{net}/G) \wedge s_e\sigma \in E(L(G)) \wedge \sigma \in [(\Sigma_e \setminus \Sigma_c) \cup S_{net}(\psi^{-1}(P_{e,s}(s_e)))]$.

(b) Language $L_n(S_{net}/G)$, which corresponds to the language generated by the compensated system under nominal operating conditions (i.e., assuming no loss of observation and only negligible delays), is defined as:

$$L_n(S_{net}/G) := \{s \in L(G) : E_n(s) \subseteq L_e(S_{net}/G)\}.$$

(c) Language $L^{\downarrow}(S_{net}/G)$, which is the language upper bound formed solely by those strings belonging to $L(G)$ that can be generated by the compensated system S_{net}/G with or without the occurrence of delays and loss of observations, is characterized as:

$$L^{\downarrow}(S_{net}/G) := \{s \in L(G) : E(s) \cap L_e(S_{net}/G) \neq \emptyset\}.$$

Based on these language definitions, we may consider the supervisory control problem where, for a given nonempty prefix-closed language K such that $K \subset L(G)$, we intend to find a networked supervisor $S_{net} : \psi^{-1}(P_{e,s}(E(L(G)))) \rightarrow 2^{\Sigma}$ that satisfies the following requirements: *(i)* $L_n(S_{net}/G) = K$, and *(ii)* $L^{\downarrow}(S_{net}/G) \subseteq K$. The first requirement ensures that, under nominal operating conditions, the compensated system S_{net}/G must be able to achieve K, and requirement *(ii)* ensures that S_{net}/G must stay inside K when there are non-negligible delays and/or loss of observations. The following proposition makes possible to combine requirements *(i)* and *(ii)* to form a single condition.

Proposition 4.2 *A networked supervisor* $S_{net} : \psi^{-1}(P_{e,s}(E(L(G)))) \rightarrow 2^{\Sigma}$ *satisfies requirements (i)* $L_n(S_{net}/G) = K$ *and (ii)* $L^{\downarrow}(S_{net}/G) \subseteq K$*, if, and only if,* $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$*.*

Proof: It is straightforward to see that, if $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$, then S_{net} satisfies requirements *(i)* and *(ii)*. In order to prove that the reverse statement also holds true, suppose a networked supervisor S_{net} that satisfies requirements *(i)* and *(ii)*. Then, $K = L_n(S_{net}/G)$ and $L^{\downarrow}(S_{net}/G) \subseteq K$. Thus, we only need to prove that $L_n(S_{net}/G) \subseteq L^{\downarrow}(S_{net}/G)$. To this end, suppose a string $s \in L_n(S_{net}/G)$. Then, according to Definition 4.6(b), $E_n(s) \subseteq L_e(S_{net}/G)$, and, according to the null extension definition (Definition 4.5), $E_n(s) \subseteq E(s)$. Therefore, we can conclude, from Definition 4.6(c), that $s \in L^{\downarrow}(S_{net}/G)$, which implies that $L_n(S_{net}/G) \subseteq L^{\downarrow}(S_{net}/G)$. ■

In accordance with Proposition 4.2, we can formulate the following supervisory control problem.

Problem 4.1 *Given a NDESWTS that satisfies Assumptions* ***A1*** *to* ***A4*** *and whose plant is modeled by an automaton* G*, and a nonempty prefix-closed language* K *such that* $K \subset L(G)$*, find a networked supervisor* $S_{net} : \psi^{-1}(P_{e,s}(E(L(G)))) \rightarrow 2^{\Sigma}$ *that satisfies the following condition:*

$$L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K. \tag{4.7}$$

It is worth remarking that, although the extended language $L_e(S_{net}/G)$ was not directly used to propose the previous networked supervisory control problem, it has a key role in the characterization of the language permissiveness of networked supervisors.

4.3.2 Existence and Design of Networked Supervisors

In order to tackle the design of networked supervisors, let Σ_{euc} denote the extended set of uncontrollable events $\Sigma_{euc} = \Sigma_{uc} \dot{\cup} \Sigma_o^s \dot{\cup} \Sigma_o^l$. Notice that, for every prefix-closed extended language $K_e \subseteq E(L(G))$, controllable with respect to $E(L(G))$ and Σ_{euc} and observable with respect to $E(L(G))$ and $P_{e,s}$, we are able to compute a P-supervisor $S_e : P_{e,s}(E(L(G))) \rightarrow 2^{\Sigma_e}$ such that $L(S_e/G_e) = K_e$ by following the

standard supervisory control theory [68, 77], considering plant G_e and assuming sets Σ_{euc} and Σ_o^s as the sets of uncontrollable and observable events, respectively. In addition, we can also obtain, from S_e, the following networked supervisor:

$$\begin{array}{rrcl} S_{net}: & \psi^{-1}(P_{e,s}(E(L(G)))) & \rightarrow & 2^{\Sigma} \\ & s_o & \mapsto & S_{net}(s_o) = S_e(\psi(s_o)) \cap \Sigma. \end{array} \tag{4.8}$$

The languages achieved by the NDESWTS under the action of the networked supervisor defined in Expression (4.8) are stated by the following result.

Lemma 4.3 *Let $K_e \subseteq E(L(G))$ be a prefix-closed extended language that is controllable with respect to $E(L(G))$ and Σ_{euc} and observable with respect to $E(L(G))$ and $P_{e,s}$, and consider a P-supervisor $S_e : P_{e,s}(E(L(G))) \rightarrow 2^{\Sigma_e}$ such that $L(S_e/G_e) = K_e$, and the networked supervisor S_{net} defined from S_e in accordance with Expression* (4.8)*. Then, the following statements hold true:*

(a) $L_e(S_{net}/G) = K_e$.

(b) $L_n(S_{net}/G) = P_e(K_e \cap E_n(L(G)))$.

(c) $L^{\downarrow}(S_{net}/G) = P_e(K_e)$.

Proof:

(a) The proof that $L_e(S_{net}/G) = K_e$ is done by induction in the length of strings $w \in E(L(G))$.

Basis step. Let $w = \varepsilon$. Then, by definition, $w \in L_e(S_{net}/G)$, and $w \in K_e$ since $K_e = \overline{K_e}$.

Induction hypothesis. For all $w \in E(L(G))$ such that $\|w\| \leq p$, $w \in L_e(S_{net}/G) \Leftrightarrow w \in K_e$.

Inductive step. Consider a string $w\sigma \in L(G_e)$ such that $\|w\| = p$ and $\sigma \in \Sigma_e$.

Suppose that $w\sigma \in L_e(S_{net}/G)$. Then, according to Definition 4.6(a), $w \in L_e(S_{net}/G)$ and $\sigma \in [(\Sigma_e \setminus \Sigma_c) \cup S_{net}(\psi^{-1}(P_{e,s}(w)))]$. Thus, in accordance with the induction hypothesis, $w \in K_e$. In addition, it can be seen that *(i)* if $\sigma \in (\Sigma_e \setminus \Sigma_c) = \Sigma_{euc}$, then $w\sigma \in K_e$ since K_e is controllable with respect to $L(G_e)$ and Σ_{euc}, and; *(ii)* if $\sigma \in S_{net}(\psi^{-1}(P_{e,s}(w)))$, then, according to Expression (4.8), $\sigma \in S_e(P_{e,s}(w))$,

which implies, according to Definition 2.7, that $w\sigma \in L(S_e/G_e) = K_e$. Therefore, in both cases, *(i)* and *(ii)*, $w\sigma \in K_e$.

Suppose now that $w\sigma \in K_e$. Then, according to Definition 2.7, $w \in K_e$ and $\sigma \in S_e(P_{e,s}(w))$. Thus, in accordance with the induction hypothesis, $w \in L_e(S_{net}/G)$, and, additionally, we can conclude, from Expression (4.8), that $\sigma \in S_{net}(\psi^{-1}(P_{e,s}(w)))$. Therefore, according to Definition 4.6(a), $w\sigma \in L_e(S_{net}/G)$.
(b) Suppose $s \in P_e(K_e \cap E_n(L(G)))$. Then, there exists $w \in P_e^{-1}(s)$ such that $w \in (K_e \cap E_n(L(G)))$, which implies, according to Definition 4.5, that:

$$(\exists w \in E_n(s))[w \in K_e]. \tag{4.9}$$

We will first consider *(i)* the case when the last event of s belongs to Σ_{uo}, and, in the sequence, *(ii)* the case when the last event of s belongs to Σ_o. In case *(i)*, it can be seen, from Definition 4.5, that $E_n(s) = \{w\}$. Thus, according to statement (4.9), $E_n(s) \subseteq K_e$. Therefore, according to Definition 4.6(b), in this case, $s \in L_n(S_{net}/G)$. In case *(ii)*, it can be seen, from Definition 4.5, that $E_n(s) = \{w, v\}$ with either $v = w\psi(\sigma)$ or $w = v\psi(\sigma)$ where σ is the last event of s. Thus, since $\psi(\sigma) \in \Sigma_{euc}$ and K_e is controllable with respect to $L(G_e)$ and Σ_{euc}, statement (4.9) implies that $E_n(s) \subseteq K_e$. Therefore, in this case, $s \in L_n(S_{net}/G)$.

Suppose now $s \in L_n(S_{net}/G)$. Then, according to Definition 4.6(b), $E_n(s) \subseteq L_e(S_{net}/G) = K_e$. Since, by definition, $E_n(s) \subseteq E_n(L(G))$, we can conclude that $E_n(s) \subseteq K_e \cap E_n(L(G))$, and, thus, $s \in P_e(K_e \cap E_n(L(G)))$.
(c) Notice that, for all $s \in L(G)$,

$$\begin{aligned} s \in L^{\downarrow}(S_{net}/G) &\Leftrightarrow E(s) \cap L_e(S_{net}/G) \neq \emptyset && \text{(Defition 4.6(c))} \\ &\Leftrightarrow E(s) \cap K_e \neq \emptyset && \text{(Since } L_e(S_{net}/G) = K_e\text{)} \\ &\Leftrightarrow (\exists w \in K_e)[P_e(w) = s], \end{aligned}$$

where the last equivalence relation is a consequence that, in accordance with Theorem 4.1, $E(s) = P_e^{-1}(s) \cap L(G_e)$. Therefore, we can conclude, by using the definition of natural projection, that $s \in L^{\downarrow}(S_{net}/G) \Leftrightarrow s \in P_e(K_e)$. ■

In order to obtain networked supervisors that solve Problem 4.1 (*i.e.*, $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$), we define, by taking into account the previous

result, the following class of extended languages:

$$\begin{aligned}\mathcal{K}_{CO}(K) := \{K_e \subseteq \Sigma_e^* : (E_n(K) \subseteq K_e = \overline{K_e} \subseteq E(K)) \\ \wedge (K_e \text{ is controllable wrt } E(L(G)) \text{ and } \Sigma_{euc}) \\ \wedge (K_e \text{ is observable wrt } E(L(G)) \text{ and } P_{e,s})\}.\end{aligned}$$

Notice that, every extended language $K_e \in \mathcal{K}_{CO}(K)$ satisfies the inclusion relation illustrated by the Venn diagram depicted in Figure 4.11. In addition, K_e is controllable with respect to $E(L(G))$ and Σ_{euc} and observable with respect to $E(L(G))$ and $P_{e,s}$, and, thus, we can design a P-supervisor $S_e : P_{e,s}(E(L(G))) \rightarrow 2^{\Sigma_e}$ such that $L(S_e/G_e) = K_e$, and also define a networked supervisor S_{net} as stated by Expression (4.8). The following results concern the use of the extended languages in $\mathcal{K}_{CO}(K)$ to design networked supervisors.

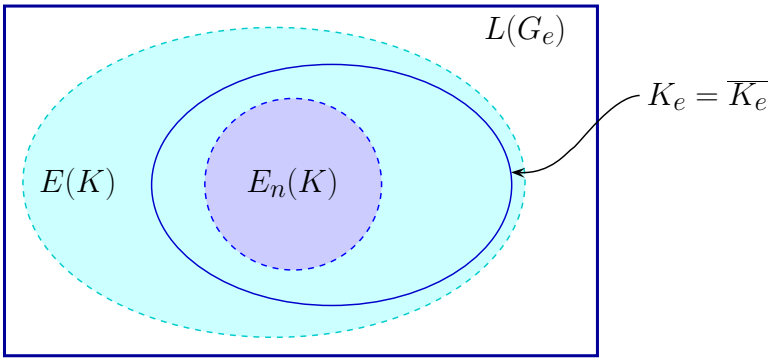


Figure 4.11: The Venn diagram that illustrates the inclusion relations between $K_e \in \mathcal{K}_{CO}(K)$, $E(K)$, $E_n(K)$ and $L(G_e)$.

Lemma 4.4 *Let $K_e \in \mathcal{K}_{CO}(K)$, and consider a P-supervisor $S_e : P_{e,s}(E(L(G))) \rightarrow 2^{\Sigma_e}$ such that $L(S_e/G_e) = K_e$, and the networked supervisor S_{net} defined from S_e in accordance with Expression* (4.8)*. Then, $L_e(S_{net}/G) = K_e$ and $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$.*

Proof: Let us consider extended language $K_e \in \mathcal{K}_{CO}(K)$. Then, according to Lemma 4.3, $L_e(S_{net}/G) = K_e$, $L_n(S_{net}/G) = P_e(K_e \cap E_n(L(G)))$ and $L^{\downarrow}(S_{net}/G) = P_e(K_e)$. Thus, it remains to be verified if *(i)* $P_e(K_e) = K$ and *(ii)* $P_e(K_e \cap E_n(L(G))) = K$.

(i) Notice that, according to the definition of $\mathcal{K}_{CO}(K)$, $E_n(K) \subseteq K_e \subseteq E(K)$, which implies that $P_e(E_n(K)) \subseteq P_e(K_e) \subseteq P_e(E(K))$. Moreover, according to

Definition 4.5, $P_e(E_n(K)) = K$ and, according to Definition 4.3, $P_e(E(K)) = K$. Therefore, $P_e(K_e) = K$.

(ii) Notice that, $P_e(K_e \cap E_n(L(G))) \subseteq P_e(K_e) \cap P_e(E_n(L(G))) = K \cap L(G) = K$. On the other hand, $E_n(K) \subseteq K_e$ implies that $P_e(K_e \cap E_n(L(G))) \supseteq P_e(E_n(K) \cap E_n(L(G))) = P_e(E_n(K)) = K$. Therefore, $P_e(K_e \cap E_n(L(G))) = K$.

■

Lemma 4.5 *Let $S_{net} : \psi^{-1}(P_{e,s}(E(L(G)))) \to 2^{\Sigma}$ be a networked supervisor such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. Then, $L_e(S_{net}/G) \in \mathcal{K}_{CO}(K)$.*

Proof: Let us consider a networked supervisor S_{net} such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. In addition, assume, without loss of generality, that $L_e(S_{net}/G) = K'_e$ where $K'_e \subseteq E(L(G))$. Notice that, according to Definition 4.6(a), K'_e is prefix-closed. Moreover, according to Definition 4.6(b), $E_n(K) \subseteq K'_e$ since $L_n(S_{net}/G) = K$, and, according to Definition 4.6(c), $K'_e \subseteq E(K)$ since $L^{\downarrow}(S_{net}/G) = K$. Therefore, $E_n(K) \subseteq K'_e = \overline{K'_e} \subseteq E(K)$. Thus, it remains to be verified if K'_e is *(i)* controllable with respect to $L(G_e)$ and Σ_{euc} and *(ii)* observable with respect to $L(G_e)$ and $P_{e,s}$. Let us first recover that $\Sigma_{euc} = (\Sigma_e \setminus \Sigma_c)$ and, according Theorem 4.1, $L(G_e) = E(L(G))$.

(i) Suppose that K'_e is not controllable with respect to $L(G_e)$ and Σ_{euc}. Then, there exist $w \in K'_e$ and $\sigma \in \Sigma_{euc}$ such that $w\sigma \in L(G_e) \setminus K'_e$, which is a contradiction since, according to Definition 4.6(a),

$$(w \in K'_e) \wedge (w\sigma \in L(G_e)) \wedge (\sigma \in \Sigma_{euc}) \Rightarrow (w\sigma \in K'_e).$$

(ii) Suppose that K'_e is not observable with respect to $L(G_e)$ and $P_{e,s}$. Then, there exist $w, w' \in K'_e$ and $\sigma \in \Sigma_e$ such that $w\sigma \in L(G_e) \setminus K'_e$, $w'\sigma \in K'_e$ and $P_{e,s}(w) = P_{e,s}(w')$, where the last equality implies that $\psi^{-1}(P_{e,s}(w)) = \psi^{-1}(P_{e,s}(w'))$. As a consequence, the control actions of S_{net} for the extended strings w and w' are the same, and, thus, $w'\sigma \in K'_e$ implies that $w\sigma \in K'_e$, which is a contradiction.

Therefore, we can conclude that $K'_e \in \mathcal{K}_{CO}(K)$. ■

We now present a necessary and sufficient condition for the existence of a networked supervisor S_{net} that solves Problem 4.1.

Theorem 4.2 *Consider a NDESWTS that satisfies Assumptions **A1** to **A4**, and whose plant is modeled by an automaton G. Let K be a prefix-closed language such that $K \subset L(G)$. Then, there exists a supervisor $S_{net} : \psi^{-1}(P_{e,s}(E(L(G)))) \rightarrow 2^{\Sigma}$ such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$ if, and only if, $\mathcal{K}_{CO}(K) \neq \emptyset$.*

Proof:

($\Leftarrow$) If $\mathcal{K}_{CO}(K) \neq \emptyset$, then we can use a $K_e \in \mathcal{K}_{CO}(K)$ to design a networked supervisor S_{net} as stated by Expression (4.8), and, consequently, the obtained S_{net} achieves, in accordance with Lemma 4.4, $L_e(S_{net}/G) = K_e$ and $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$.

($\Rightarrow$) Suppose that there exists a networked supervisor S_{net} such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. According to Lemma 4.5, $L_e(S_{net}/G) \in \mathcal{K}_{CO}(K)$, which implies that $\mathcal{K}_{CO}(K) \neq \emptyset$. ■

Notice that, since the class of prefix-closed observable and controllable superlanguages of $E_n(K)$ is closed under intersection, the infimal prefix-closed controllable (with respect to $E(L(G))$ and Σ_{euc}) and observable (with respect to $E(L(G))$ and $P_{e,s}$) superlanguage of $E_n(K)$, denoted here by $E_n(K)^{\downarrow CO}$, always exists. Such a language can be used to check the existence of networked supervisors, and, additionally, it can be applied to design a networked supervisor S_{net}, when there exists one, such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$, as shown by the following result.

Proposition 4.3 *Let K be a prefix-closed language such that $K \subset L(G)$. Then,*

$$E_n(K)^{\downarrow CO} \subseteq E(K) \;\Leftrightarrow\; E_n(K)^{\downarrow CO} \in \mathcal{K}_{CO}(K) \;\Leftrightarrow\; \mathcal{K}_{CO}(K) \neq \emptyset.$$

Proof: $(a) \Rightarrow (b)$. Since $E_n(K)^{\downarrow CO}$ is the infimal prefix-closed controllable and observable superlanguage of $E_n(K)$, we can conclude, by using the definition of the class $\mathcal{K}_{CO}(K)$, that $E_n(K)^{\downarrow CO} \in \mathcal{K}_{CO}(K)$ if $E_n(K)^{\downarrow CO} \subseteq E(K)$.
$(b) \Rightarrow (c)$. This implication relation holds true by definition.
$(c) \Rightarrow (a)$. Suppose that *(i)* $\mathcal{K}_{CO}(K) \neq \emptyset$ and *(ii)* $E_n(K)^{\downarrow CO} \not\subset E(K)$. Notice that, according to the definition of class $\mathcal{K}_{CO}(K)$, statement *(i)* implies that there exists

a prefix-closed controllable and observable language K'_e such that $E_n(K) \subseteq K'_e \subseteq E(K)$. On the other hand, since $E_n(K)^{\downarrow CO}$ is the infimal prefix-closed controllable and observable superlanguage of $E_n(K)$, statement *(ii)* implies that every prefix-closed controllable and observable superlanguage of $E_n(K)$ is not a sublanguage of $E(K)$, which is a contradiction. ■

Example 4.8 *Consider the NDESWTS presented in Example 4.1, whose communication network and plant are depicted again in Figures 4.12(a,b), respectively, and suppose that* $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$, $\Sigma_{lo} = \{\beta\}$ *and* $\Sigma_c = \{\alpha, \gamma, \mu, \eta\}$. *Automaton* G_e *that generates* $E(L(G))$ *is depicted again in Figure 4.12(c). Consider also the same language* K *assumed in Example 4.7, which is generated by automaton* H *depicted in Figure 4.12(d). Automata* H_n *and* H_e, *whose generated languages are* $E_n(K)$ *and* $E(K)$ *are depicted again in Figures 4.12(e,f), respectively.*

We want to design a networked supervisor S_{net} *that satisfies the admissible language* K, *i.e.* $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. *In other words,* S_{net}/G *must be able to execute all string in* $K = \overline{\{\alpha\mu\gamma, \eta\alpha\beta\gamma\}}$ *under nominal operating conditions, but* S_{net} *must prevent the occurrence of the forbidden string* $s' = \eta\gamma$ *even when delays or loss of observations occur. Therefore, since* $\eta \in \Sigma_{uo}$, *the networked supervisor must initially disable event* γ, *and enable it only after observing some event occurrence.*

Automaton H_{co} *that generates* $E_n(K)^{\downarrow CO}$ *is depicted in Figure 4.13. By comparing Figures 4.12(f) and 4.13, we can see that* $E_n(K)^{\downarrow CO} \subseteq E(K)$, *which implies, according to Proposition 4.3, that* $\mathcal{K}_{CO}(K) \neq \emptyset$.

Since $\mathcal{K}_{CO}(K) \neq \emptyset$, *according to Theorem 4.2, there exists a networked supervisor* S_{net}. *Moreover, according to Proposition 4.3,* $E_n(K)^{\downarrow CO} \in \mathcal{K}_{CO}(K)$. *Thus, in order to compute* S_{net}, *we can first compute a P-supervisor* S_e *assuming plant* G_e, $K_e = E_n(K)^{\downarrow CO}$, $\Sigma_{euc} = \{\alpha_s, \beta, \beta_s, \beta_l, \gamma_s\}$ *and* $\Sigma_o^s = \{\alpha_s, \beta_s, \gamma_s\}$. *The P-supervisor* S_e *obtained, whose realization is depicted in Figure 4.14, is defined as:* $S_e(\varepsilon) = \Sigma_e \setminus \{\mu, \gamma, \gamma_s\}$, $S_e(\alpha_s) = \Sigma_e \setminus \{\alpha, \alpha_s, \eta\}$, $S_e(\beta_s) = \{\alpha_s\}$, $S_e(\alpha_s\beta_s) = \{\gamma, \gamma_s\}$, *and* $S_e(\beta_s\alpha_s) = S_e(\alpha_s\gamma_s) = S_e(\alpha_s\beta_s\gamma_s) = \emptyset$. *Then, the networked supervisor* S_{net} *computed from* S_e *as stated by Expression* (4.8) *is defined as follows:* $S_{net}(\varepsilon) = \{\alpha, \beta, \eta\}$, $S_{net}(\alpha) = \{\beta, \gamma, \mu\}$, $S_{net}(\alpha\beta) = \{\gamma\}$ *and* $S_{net}(\beta) = S_{net}(\beta\alpha) = S_{net}(\alpha\gamma) = S_{net}(\alpha\beta\gamma) = \emptyset$.

According to Lemma 4.3, $L_e(S_{net}/G) = E_n(K)^{\downarrow CO}$. *Thus, with the help of Fig-*

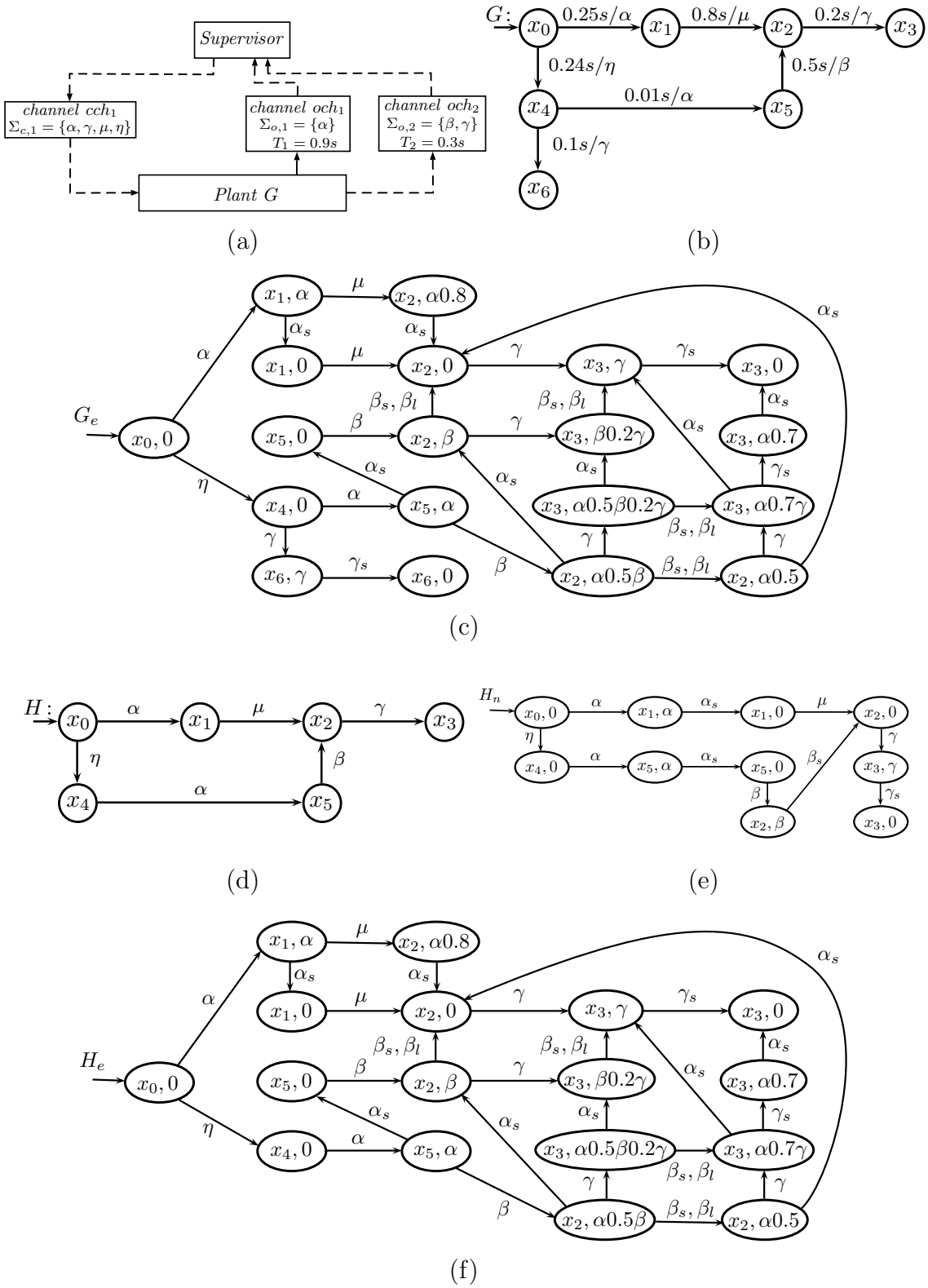


Figure 4.12: Network (a) and plant G (b) of the NDESWTS, automata G_e (c), H (d), H_n (e) and H_e (f) considered in Example 4.8.

ure 4.13, we can check the following facts regarding the performance of the networked supervisor S_{net} computed by using extended language $K_e = E_n(K)^{\downarrow CO}$:

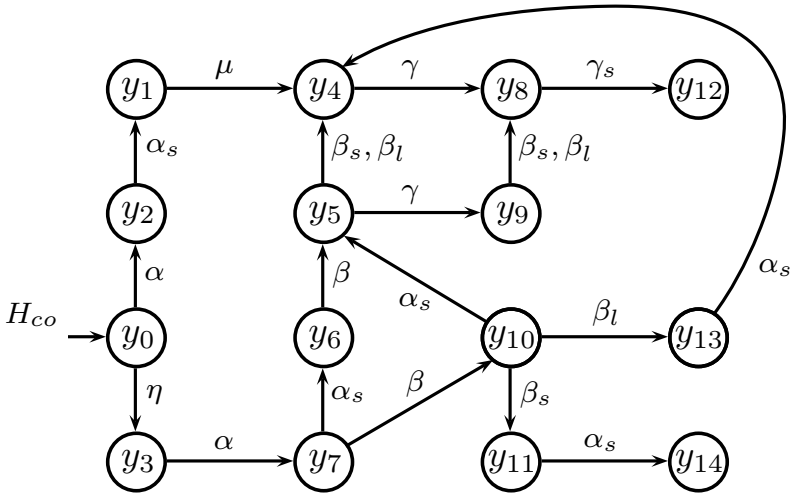


Figure 4.13: Automaton that generates $E_n(K)^{\downarrow CO}$ computed in Example 4.8.

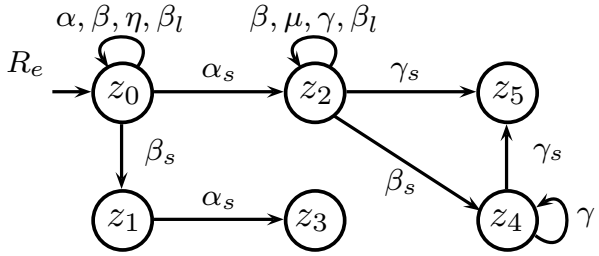


Figure 4.14: Realization of P-supervisor S_e used to define networked supervisor S_{net} in Example 4.8.

- *Networked supervisor S_{net} prevents the execution of the forbidden string $s' = \eta\gamma$ since $E(s') \cap L_e(S_{net}/G) = \emptyset$, where $E(s') = \{\eta\gamma, \eta\gamma\gamma_s\}$;*

- *The strings in K can be generated by S_{net}/G under nominal operating conditions since $E_n(K) = \overline{\{\alpha\alpha_s\mu\gamma, \alpha\alpha_s\mu\gamma\gamma_s, \eta\alpha\alpha_s\beta\beta_s\gamma, \eta\alpha\alpha_s\beta\beta_s\gamma\gamma_s\}} \subseteq L_e(S_{net}/G)$.*

The comments presented above agree with Lemma 4.4, which determines that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. The following comments concern the language permissiveness of the networked supervisor S_{net} computed by using extended language $K_e = E_n(K)^{\downarrow CO}$, in the presence of delays and/or loss of observations:

- *The compensated system is able to generate string $s_1 = \eta\alpha\beta\gamma \in K$ in the cases: (i) when the occurrence of event α is observed before the observation of β, and (ii) when the observation of β is lost. On the other hand, since $\eta\alpha\beta\beta_s\alpha_s\gamma \notin L_e(S_{net}/G)$, the compensated system is not able to execute string*

s_1 in the case when the occurrence of event α is observed after the observation of β;

- *In some cases, the networked supervisor S_{net} computed using extended language $K_e = E_n(K)^{\downarrow CO}$, may unnecessarily postpone the permission for some events to take place. For example, during the execution of string $s_2 = \alpha\mu\gamma \in K$, networked supervisor S_{net} postpones the permission for event μ to take place until it observes the occurrence of α since $\alpha\mu \notin L_e(S_{net}/G)$.*

Based on this comments, we can conclude that, the networked supervisor S_{net} computed by using extended language $K_e = E_n(K)^{\downarrow CO}$ achieves the control specification $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$, but nevertheless the achieved language permissiveness in the presence of non-negligible delays and loss of observations can be increased by allowing the occurrences of extended strings that were unnecessarily prevented, such as, $\eta\alpha\beta\beta_s\alpha_s\gamma$ and $\alpha\mu$.

Improving Language Permissiveness by Using Relative Observability

In accordance with Lemma 4.4 and Proposition 4.3, we can compute a networked supervisor, when there exists one, by setting $K_e = E_n(K)^{\downarrow CO}$ and synthesizing S_{net} as determined by Expression (4.8). By following this procedure, the designed networked supervisor will be such that $L_e(S_{net}/G) = E_n(K)^{\downarrow CO}$. Nonetheless, the permissiveness of the extended language $L_e(S_{net}/G)$ may be improved by searching for extended languages belonging to $\mathcal{K}_{CO}(K)$ that are more permissive than $E_n(K)^{\downarrow CO}$, as illustrated by the Venn diagram depicted in Figure 4.15. However,

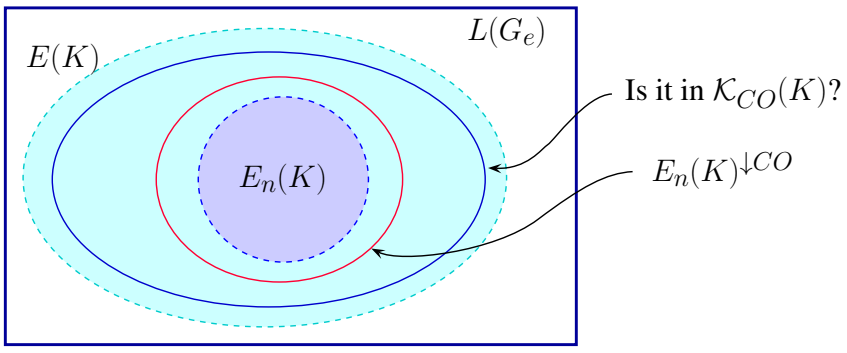


Figure 4.15: The Venn diagram that illustrates the search for extended languages belonging to $\mathcal{K}_{CO}(K)$ that are more permissive than $E_n(K)^{\downarrow CO}$.

since the observability is not preserved under set unions, the class of extended languages $\mathcal{K}_{CO}(K)$ does not have a supremal in general.

In order to circumvent this drawback, we will apply the relative observability property [15, 84] with a view to searching for controllable and relatively observable extended languages in $\mathcal{K}_{CO}(K)$ that are more permissive than $E_n(K)^{\downarrow CO}$. To this end, let $E(K)^{\uparrow CRO}$ denote the controllable and observable sublanguage of $E(K)$ computed by using Algorithm 3.3, presented in Section 3.4, with inputs G_e and H_e (by assuming fully marked automata, *i.e.*, $L_m(G_e) = L(G_e) = E(L(G))$ and $L_m(H_e) = L(H_e) = E(K)$). The following proposition determines the case when extended language $E(K)^{\uparrow CRO}$ can be used to design a networked supervisor S_{net} for Problem 4.1.

Proposition 4.4 *Let K be a prefix-closed language such that $K \subset L(G)$. Then, there exists a networked supervisor S_{net} such that $L_e(S_{net}/G) = E(K)^{\uparrow CRO}$ and $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$ if, and only if, $E(K)^{\uparrow CRO} \supseteq E_n(K)$.*

Proof: According to Lemmas 4.4 and 4.5, there exists S_{net} such that $L_e(S_{net}/G) = E(K)^{\uparrow CRO}$ and $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$ if, and only if, $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$. By definition, $E(K)^{\uparrow CRO} \subseteq E(K)$ and, according to Proposition 3.1, $E(K)^{\uparrow CRO}$ is controllable with respect to $E(L(G))$ and Σ_{euc} and observable with respect to $E(L(G))$ and $P_{e,s}$. Moreover, it can be checked that $E(K)$ and $E(K)^{\uparrow CRO}$ are prefix-closed since K is prefix-closed. Therefore, we can conclude, from the definition of class $\mathcal{K}_{CO}(K)$, that $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$ if, and only if, $E(K)^{\uparrow CRO} \supseteq E_n(K)$, which concludes the proof. ■

Example 4.9 *Consider the same networked supervisory control problem addressed in Example 4.8, where the communication network and the plant of the NDESWTS are depicted in Figures 4.12(a,b), respectively, and we suppose that $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$, $\Sigma_{lo} = \{\beta\}$ and $\Sigma_c = \{\alpha, \gamma, \mu, \eta\}$. Consider, also, the same admissible language K generated by automaton H depicted in Figure 4.12(d).*

Automaton H_{CRO} that generates $E(K)^{\uparrow CRO}$ is depicted in Figure 4.16. By comparing this automaton with that depicted in Figure 4.12(e), which generates language $E_n(K)$, we can see that $E(K)^{\uparrow CRO} \supseteq E_n(K)$. As a consequence, $E(K)^{\uparrow CRO} \in$

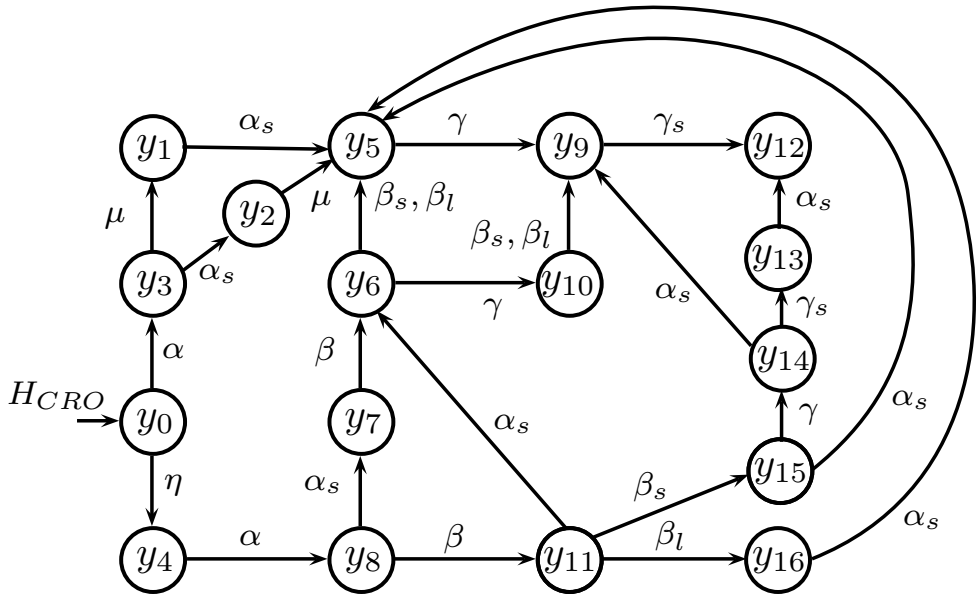


Figure 4.16: Automaton that generates $E(K)^{\uparrow CRO}$ computed in Example 4.9.

$\mathcal{K}_{CO}(K)$, and, in accordance with Proposition 4.4, there exists a networked supervisor S_{net} that solves Problem 4.1 and such that $L_e(S_{net}/G) = E(K)^{\uparrow CRO}$.

Since $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$, in accordance with Lemma 4.3, we can use $K_e = E(K)^{\uparrow CRO}$ to design a P-supervisor S_e by assuming plant G_e, $\Sigma_{euc} = \{\alpha_s, \beta, \beta_s, \beta_l, \gamma_s\}$ and $\Sigma_o^s = \{\alpha_s, \beta_s, \gamma_s\}$. The realization of the obtained P-supervisor is depicted in Figure 4.17, and it is defined as: $S_e(\varepsilon) = \Sigma_e \setminus \{\gamma, \gamma_s\}$, $S_e(\alpha_s) = \Sigma_e \setminus \{\alpha, \alpha_s, \eta\}$, $S_e(\beta_s) = \{\alpha_s, \gamma, \gamma_s\}$, $S_e(\alpha_s\beta_s) = S_e(\beta_s\alpha_s) = \{\gamma, \gamma_s\}$, $S_e(\beta_s\gamma_s) = \{\alpha_s\}$ and $S_e(\beta_s\alpha_s\gamma_s) = S_e(\beta_s\gamma_s\alpha_s) = S_e(\alpha_s\gamma_s) = S_e(\alpha_s\beta_s\gamma_s) = \emptyset$.

The networked supervisor, obtained from S_e in accordance with Expression (4.8),

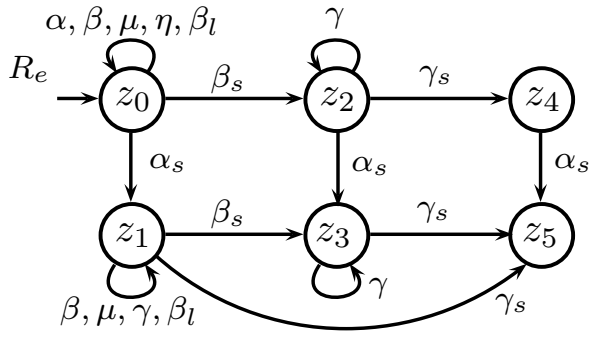


Figure 4.17: Realization of P-supervisor S_e used to define networked supervisor S_{net} in Example 4.9.

is defined as follows: $S_{net}(\varepsilon) = \Sigma \setminus \{\gamma\}$, $S_{net}(\alpha) = \{\beta, \gamma, \mu\}$, $S_{net}(\alpha\beta) = S_{net}(\beta\alpha) = \{\gamma\}$, $S_{net}(\beta) = \{\gamma\}$ *and* $S_{net}(\beta\alpha) = S_{net}(\alpha\gamma) = S_{net}(\alpha\beta\gamma) = S_{net}(\beta\gamma\alpha) = S_{net}(\beta\alpha\gamma) = \emptyset$.

Since $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$, *in accordance with Lemma 4.4,* $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$. *In addition, by comparing Figures 4.13 and 4.16, we can see that, for the case of this example,* $E_n(K)^{\downarrow CO} \subset E(K)^{\uparrow CRO}$, *and the following cases illustrate some situations where the language permissiveness has been improved by designing* S_{net} *setting extended language* K_e *as* $E(K)^{\uparrow CRO}$ *instead of* $E_n(K)^{\downarrow CO}$*:*

- *Under the action of the networked supervisor designed from* $K_e = E(K)^{\uparrow CRO}$, *the compensated system is able to generate string* $s_1 = \eta\alpha\beta\gamma \in K$ *regardless of the order of the observations of* α *and* β, *as opposed with the networked supervisor computed from* $K_e = E_n(K)^{\downarrow CO}$ *that prevent the occurrence of* γ *in the case when* α *is observed after the observation of* β*;*

- *During the execution of string* $s_2 = \alpha\mu\gamma \in K$, *the networked supervisor designed from* $L_e(S_{net}/G) = E(K)^{\uparrow CRO}$ *allows the occurrence of event* μ *before the observation of* α, *whereas the networked supervisor designed from* $L_e(S_{net}/G) = E_n(K)^{\downarrow CO}$ *only allows the occurrence of* μ *after the observation of* α.

Remark 4.2 *Notice that, in the extended model* G_e, *the set of controllable events stays being* Σ_c *whereas the set of observable events becomes* Σ_o^s, *and, thus,* $\Sigma_c \cap \Sigma_o^s = \emptyset$, *i.e., all controllable events of* G_e *are unobservable. Then, in accordance with Remark 2.1, it can be checked that, for all* $K \subset L(G)$ *with* $K \neq L(G)$, *there is no normal language belonging to* $\mathcal{K}_{CO}(K)$. *As a consequence, the property of normality, in contrast to the property of relative observability, cannot be used to increase the achieved language permissiveness in the networked supervisory control problem.*

Realization of Networked Supervisors

In Examples 4.8 and 4.9, we present the obtained networked supervisor by listing the control action $S_{net}(s_o)$, for each $s_o \in \psi^{-1}(P_{e,s}(K_e))$, where $K_e = L_e(S_{net}/G)$. In order to make possible the definition of a networked supervisor in a more convenient manner, we propose the following realization.

Definition 4.7 (realization of networked supervisors) *A realization of a networked supervisor S_{net} is a pair $\mathcal{R}_{net} = (R, \Omega)$, where the first element is an automaton $R = (X_{net}, \Sigma_o, f_{net}, \Gamma_{net}, x_{0_{net}})$ and the second element is a function $\Omega : X_{net} \to 2^{\Sigma}$, which are defined in order to ensure that,*

$$S_{net}(s_o) = \Omega(f_{net}(x_{0_{net}}, s_o)),\ \forall s_o \in \psi^{-1}(P_{e,s}(L_e(S_{net}/G))).$$

For a given networked supervisor S_{net}, a realization $\mathcal{R}_{net} = (R, \Omega)$ that agrees with Definition 4.7 can be obtained from a realization $R_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e})$ of a P-supervisor S_e such that $L(S_e/G_e) = L_e(S_{net}/G)$, which can be computed using Algorithm 2.2 presented in Subsection 2.5.2. To do so, $\mathcal{R}_{net} = (R, \Omega)$ is defined from R_e as follows:

$$R = (X_{net}, \Sigma_o, f_{net}, \Gamma_{net}, x_{0_{net}}), \tag{4.10}$$

where:

$$\begin{aligned}
X_{net} &= X_e, && (4.11)\\
f_{net}(x, \sigma) &= f_e(x, \psi(\sigma)),\ \forall (x, \sigma) \in X_{net} \times \Sigma_o, && (4.12)\\
\Gamma_{net}(x) &= \psi^{-1}(\Gamma_e(x) \cap \Sigma_o^s),\ \forall x \in X_{net}, && (4.13)\\
x_{0_{net}} &= x_{0_e}, && (4.14)
\end{aligned}$$

and, additionally,

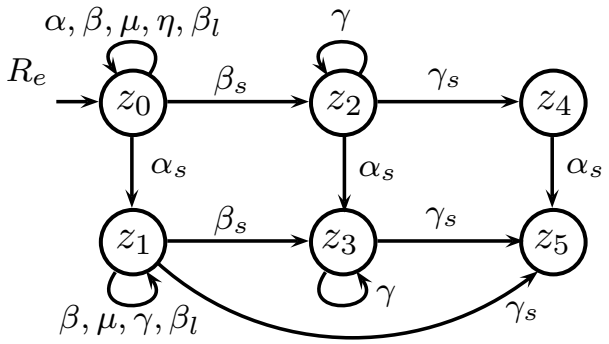
$$\Omega(x) = \Gamma_e(x) \cap \Sigma,\ \forall x \in X_{net}. \tag{4.15}$$

This procedure for obtaining a realization of S_{net} directly emerges from Lemma 4.3. The following example illustrates the construction of a networked supervisor realization.

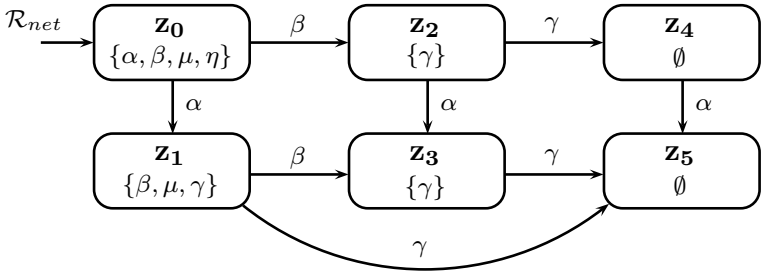
Example 4.10 *Consider the same networked supervisory control problem addressed in Example 4.9, where the communication network and the plant of the NDESWTS are depicted in Figures 4.12(a,b), respectively, and we suppose that $\Sigma = \{\alpha, \beta, \gamma, \mu, \eta\}$, $\Sigma_o = \{\alpha, \beta, \gamma\}$, $\Sigma_{lo} = \{\beta\}$ and $\Sigma_c = \{\alpha, \gamma, \mu, \eta\}$.*

In Example 4.9, we computed a networked supervisor S_{net} such that $L_e(S_{net}/G) = E(K)^{\uparrow CRO}$ and $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$, where K is the language generated by automaton H depicted in Figure 4.12(d). This networked supervisor was obtained

by applying Expression (4.8) *with P-supervisor* S_e *such that* $L(S_e/G_e) = E(K)^{\uparrow CRO}$ *and whose realization,* R_e*, is depicted again in Figure 4.18(a). Then, in oder to obtain a realization* $\mathcal{R}_{net} = (R, \Omega)$ *from* R_e *for networked supervisor* S_{net}*, we construct automaton* R *as stated by Equations* (4.10) *to* (4.14)*, which is depicted in Figure 4.18(b); the values of* Ω *for each state of* R *are defined in accordance with Equation* (4.15) *and are presented in Figure 4.18(b) inside of its corresponding state of* R*.*



(a)



(b)

Figure 4.18: Realizations of P-supervisor S_e (a) and networked supervisor S_{net} (b) computed using extended language $E(K)^{\uparrow CRO}$.

4.3.3 Procedure for Designing Networked Supervisors

In this subsection, we propose a systematic way to design networked supervisors for Problem 4.1 based on the results presented in Subsection 4.3.2.

The networked supervisor design should be carried out by following the flow chart shown in Figure 4.19. According to Figure 4.19, the design procedure starts by checking if $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$. In order to do so, we compute an automaton that generates extended language $E(K)^{\uparrow CRO}$ by using Algorithm 3.3 proposed in

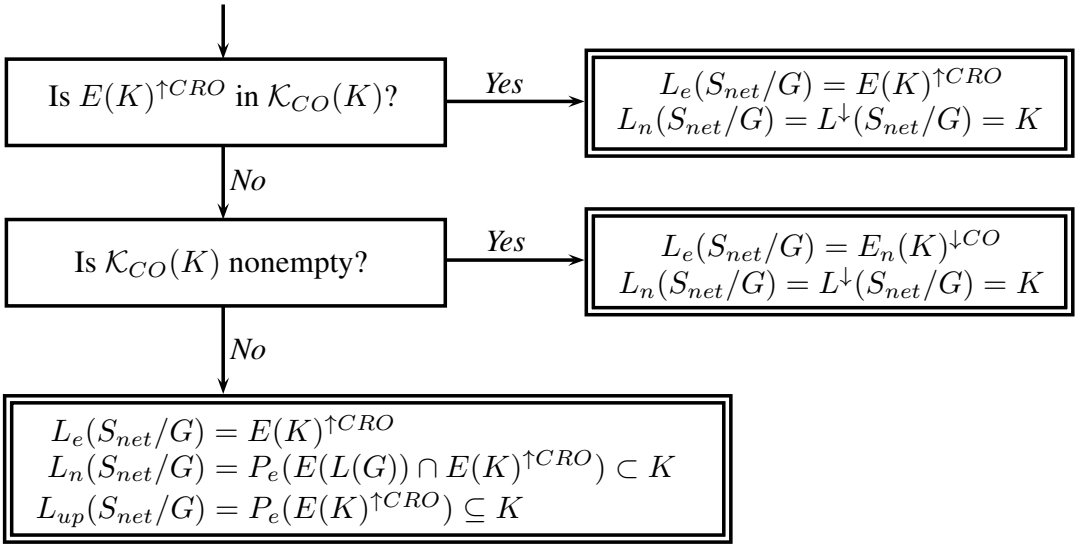


Figure 4.19: Flow chart for the design of networked supervisors.

Section 3.4. If $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$, then we compute a P-supervisor S_e for extended plant G_e assuming Σ_{euc} as the set of uncontrollable events, Σ_o^s as the set of observable events and $E(K)^{\uparrow CRO}$ as the admissible language, and, in the sequence, we compute S_{net} as stated by Expression (4.8). According to Proposition 4.4, the compensated system achieves the following behavior:

$$L_e(S_{net}/G) = E(K)^{\uparrow CRO} \text{ and } L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K.$$

It is worth noting that, as presented in Subsection 2.5.3, when $E(K)$ is observable with respect to $E(L(G))$ and $P_{e,s}$, it is also $E(K)$-observable. As a consequence of this fact, if $E(K)$ is controllable with respect to $E(L(G))$ and Σ_{euc} and observable with respect to $E(L(G))$ and $P_{e,s}$, then $E(K)^{\uparrow CRO} = E(K)$. Therefore, in this case, the designed networked supervisor achieves the maximal language permissiveness, *i.e.*, $L_e(S_{net}/G) = E(K)$.

When $E(K)^{\uparrow CRO} \notin \mathcal{K}_{CO}(K)$, the design procedure continues, in accordance with Figure 4.19, by checking if $\mathcal{K}_{CO}(K) \neq \emptyset$. This can be done, according to Proposition 4.3, by computing $E_n(K)^{\downarrow CO}$, the infimal prefix-closed controllable (with respect to $L(G_e)$ and Σ_{euc}) and observable (with respect to $L(G_e)$ and $P_{e,s}$) superlanguage of $E_n(K)$, and checking if $E_n(K)^{\downarrow CO} \subseteq E(K)$. If the answer is yes, then a P-supervisor S_e for G_e can be computed assuming Σ_{euc} and Σ_o^s as the sets of uncontrollable and

observable events, respectively, and $E_n(K)^{\downarrow CO}$ as the admissible language. In the sequence, the networked supervisor S_{net} is computed as stated by Expression (4.8). According to Lemma 4.4, the resulting compensated system achieves the following behavior:

$$L_e(S_{net}/G) = E_n(K)^{\downarrow CO} \text{ and } L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K.$$

Finally, if $\mathcal{K}_{CO}(K)$ is empty, then, according to Theorem 4.2, there does not exist a networked supervisor S_{net} such that $L_n(S_{net}/G) = L^{\downarrow}(S_{net}/G) = K$, *i.e.*, the admissible language K cannot be achieved. In order to circumvent this problem, the supervisor S_{net} will be computed as in the case when $E(K)^{\uparrow CRO} \in \mathcal{K}_{CO}(K)$, making, according to Lemma 4.3, the compensated system to achieve the following behavior:

$$\begin{aligned} L_e(S_{net}/G) &= E(K)^{\uparrow CRO} \\ L_n(S_{net}/G) &= P_e(E_n(L(G)) \cap E(K)^{\uparrow CRO}) \subset K \\ L^{\downarrow}(S_{net}/G) &= P_e(E(K)^{\uparrow CRO}) \subseteq K. \end{aligned}$$

4.4 Concluding Remarks

In this chapter, we have introduced an architecture of supervisory control of Discrete Event System, in which, a network that may have several channels performs the communication between the plant and the supervisor, and a timing structure provides time information about the behaviors of the plant and the communication network. We proposed a methodology to construct an equivalent untimed model for a NDESWTS that represents all possible implications of delays and losses of observations, and, based on this model, we have formulated a supervisory control problem of NDESWTS, where the networked supervisor ensures that, when there is neither non-negligible delays nor loss of observations, the compensated system achieves a given admissible language, and, when delays and loss of observations are present, it ensures that the compensated system behavior stays inside the admissible language. We have presented a necessary and sufficient condition for the existence of networked supervisors, and also a systematic way to design networked supervisors,

where the relative observability property is used to increase the achieved language permissiveness.

Chapter 5

Conclusions and Future Works

The main goal of this work was to study the supervisory control problem of networked discrete event systems with timing structure (NDESWTS) in the presence of delays and loss of observations. In this regard, the main contributions of this work are described below:

1. The introduction of an architecture of supervisory control of Discrete Event System, in which, a network that may have several channels performs the communication between the plant and the supervisor, together with a timing structure composed of the minimal activation time of the plant transitions and the upper bounds on the delays of the communication channels.

2. A methodology to construct an equivalent untimed model for NDESWTS. This methodology provides a complete characterization of all possible implications of delays and losses of observations by means of extended languages, and algorithms for the computation of untimed deterministic finite-state automata that generate the extended languages associated with the plant and the control specification. It is worth remarking that the model for NDESWTS proposed here is general enough to allow its application in other NDESWTS problems, for instance, it has also been used to study the codiagnosability of NDESWTS subject to event communication delays and loss of observations in [66, 67].

3. The formulation of a supervisory control problem of NDESWTS where the networked supervisor ensures that, when there is neither non-negligible delays nor loss of observations, the compensated system achieves the admissible

language, and, when delays and/or loss of observations are present, it ensures that the compensated system behavior stays inside the admissible language. In order to tackle this problem, we presented a necessary and sufficient condition for the existence of networked supervisors, and, additionally, a systematic way to design networked supervisors. To this end, we used relative observability property with a view to increasing the achieved language permissiveness.

A preliminary version of the results obtained here was published in [88, 89].

Another research topic addressed in this work was the concept of relative observability. In this context, the main contributions are:

1. An algorithm for the verification of relative observability that has polynomial computational complexity.

2. An algorithm for the computation of the supremal relatively observable sublanguage of a regular language, whose computational complexity is, in general, exponential, but decreases to polynomial when the automaton that marks the specification language is state partition.

3. An algorithm for the computation of a controllable and observable sublanguage of a given regular language, which applies the concept of relative observability and iteratively shrinks the ambient language in order to increase the achieved language permissiveness.

The results on relative observability have been published in [82–84].

Possible future research directions are listed below:

1. Supervisory control of NDESWTS subject to delays and loss of control actions. In this topic, the purpose is to remove Assumption **A4**, imposed in Chapter 4, and to consider communication delays and packet losses in observation and control channels.

2. Extension to other problems in supervisory control of NDESWTS, so as to address blocking properties of networked supervisory control or to extend the approach proposed here to modular and decentralized control architectures.

3. The study of other NDESWTS problems by using the modeling proposed in this work, for example, state estimation, prognosis, detectability and opacity of NDESWTS.

4. Networked discrete event systems with different timing structures. In this topic, the idea is to consider a timing structure, where, instead of assuming minimal activation times of the plant transitions and maximal communication delays, we could assume *a priori* knowledge of the time intervals in which the plant transitions can occur and the lower and upper bounds on the transmission delays, without relying on automata with guards.

Bibliographic References

[1] ALVES, M. V. S. *Controle supervisório robusto de sistemas a eventos discretos sujeitos a perdas intermitentes de observação*. Tese de Mestrado, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 2014.

[2] LEE, J., BAGHERI, B., KAO, H.-A. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems", *Manufacturing Letters*, v. 3, n. C, pp. 18 – 23, 2015.

[3] GILCHRIST, A. *Industry 4.0: the industrial internet of things*. 1st ed. Berkeley, USA, Apress, 2016.

[4] WANG, S., WAN, J., ZHANG, D., et al. "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination", *Computer Networks*, v. 101, pp. 158–168, 2016.

[5] HUI, T. K. L., SHERRATT, R. S., SÁNCHEZ, D. D. "Major requirements for building Smart Homes in Smart Cities based on Internet of Things technologies", *Future Generation Computer Systems*, v. 76, n. C, pp. 358 – 369, 2017.

[6] JAIN, B., BRAR, G., MALHOTRA, J., et al. "A novel approach for smart cities in convergence to wireless sensor networks", *Sustainable Cities and Society*, v. 35, n. C, pp. 440 – 448, 2017.

[7] KIM, K. D., KUMAR, P. R. "Cyber-Physical Systems: A Perspective at the Centennial", *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, pp. 1287–1308, 2012.

[8] KHAITAN, S. K., MCCALLEY, J. D. "Design Techniques and Applications of Cyberphysical Systems: A Survey", *IEEE Systems Journal*, v. 9, n. 2, pp. 350–365, 2015.

[9] WANG, L., TÖRNGREN, M., ONORI, M. "Current status and advancement of cyber-physical systems in manufacturing", *Journal of Manufacturing Systems*, v. 37, n. 2, pp. 517 – 527, 2015.

[10] HEHENBERGER, P., VOGEL-HEUSER, B., BRADLEY, D., et al. "Design, modelling, simulation and integration of cyber physical systems: Methods and applications", *Computers in Industry*, v. 82, n. C, pp. 273 – 289, 2016.

[11] LEE, E. A., SESHIA, S. A. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. MIT Press, 2017.

[12] HESPANHA, J. P., NAGHSHTABRIZI, P., XU, Y. "A Survey of Recent Results in Networked Control Systems", *Proceedings of the IEEE*, v. 95, n. 1, pp. 138–162, 2007.

[13] GUPTA, R. A., CHOW, M. Y. "Networked Control System: Overview and Research Trends", *IEEE Transactions on Industrial Electronics*, v. 57, n. 7, pp. 2527–2535, 2010.

[14] LIN, F. "Control of Networked Discrete Event Systems: Dealing with Communication Delays and Losses", *SIAM Journal on Control and Optimization*, v. 52, n. 2, pp. 1276–1298, 2014.

[15] CAI, K., ZHANG, R., WONHAM, W. M. "Relative Observability of Discrete-Event Systems and Its Supremal Sublanguages", *IEEE Transactions on Automatic Control*, v. 60, n. 3, pp. 659–670, 2015.

[16] KOMENDA, J., MASOPUST, T., VAN SCHUPPEN, J. H. "Relative Observability in Coordination Control". In: *IEEE International Conference on Automation Science and Engineering*, pp. 75–80, Gothenburg, Sweden, 2015. IEEE.

[17] CAI, K., ZHANG, R., WONHAM, W. M. "On relative coobservability of discrete-event systems". In: *American Control Conference*, pp. 371–376, Chicago, IL, USA, 2015.

[18] CAI, K., ZHANG, R., WONHAM, W. M. "Relative Observability and Coobservability of Timed Discrete-Event Systems", *IEEE Transactions on Automatic Control*, v. 61, n. 11, pp. 3382–3395, 2016.

[19] ZHANG, R., CAI, K. "On supervisor localization based distributed control of discrete-event systems under partial observation". In: *American Control Conference*, pp. 764–769, Boston, MA, USA, 2016.

[20] CAI, K., ZHANG, R., WONHAM, W. M. "Characterizations and effective computation of supremal relatively observable sublanguages", *Discrete Event Dynamic Systems*, 2017.

[21] LIN, F., WONHAM, W. M. "On observability of discrete-event systems", *Information Science*, v. 44, n. 3, pp. 173–198, 1988.

[22] KOMENDA, J. "Supervisory Control with Partial Observations". In: Seatzu, C., Silva, M., van Schuppen, J. (Eds.), *Control of Discrete-Event Systems*, v. 433, Springer, pp. 65–84, 2013.

[23] RAMADGE, P. J. G., WONHAM, W. M. "Supervisory control of a class of discrete event process", *SIAM Journal of Control Optimization*, v. 25, n. 1, pp. 206–230, 1987.

[24] 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.

[25] WONHAM, W. M., RAMADGE, P. J. "Modular supervisory control of discrete-event systems", *Mathematics of Control, Signals and Systems*, v. 1, n. 1, pp. 13–30, 1988.

[26] THISTLE, J. G. "Supervisory control of discrete event systems", *Mathematical and Computer Modelling*, v. 23, n. 11, pp. 25–53, 1996.

[27] BARRETT, G., LAFORTUNE, S. "Decentralized supervisory control with communicating controllers", *IEEE Transactions on Automatic Control*, v. 45, n. 9, pp. 1620–1638, 2000.

[28] MOODY, J. O., ANTSAKLIS, P. J. "Petri net supervisors for DES with uncontrollable and unobservable transitions", *IEEE Transactions on Automatic Control*, v. 45, n. 3, pp. 462–476, 2000.

[29] 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.

[30] BALEMI, S. "Communication delays in connections of input/output discrete event processes". In: *IEEE Conference on Decision and Control*, v. 4, pp. 3374–3379, Tucson, AZ, USA, 1992.

[31] BALEMI, S. "Input/output discrete event processes and communication delays", *Discrete Event Dynamic Systems*, v. 4, n. 1, pp. 41–85, 1994.

[32] PARK, S. J., CHO, K. H. "Delay-robust supervisory control of discrete-event systems with bounded communication delays", *IEEE Transactions on Automatic Control*, v. 51, n. 5, pp. 911–915, 2006.

[33] PARK, S. J., CHO, K. H. "Supervisory control of discrete event systems with communication delays and partial observations", *Systems & Control Letters*, v. 56, n. 2, pp. 106–112, 2007.

[34] PARK, S. J., CHO, K. H. "Decentralized supervisory control of discrete event systems with communication delays based on conjunctive and permissive decision structures", *Automatica*, v. 43, n. 4, pp. 738 – 743, 2007.

[35] PARK, S. J., CHO, K. H. "Delay-coobservability and its algebraic properties for the decentralized supervisory control of discrete event systems with communication delays", *Automatica*, v. 45, n. 5, pp. 1252 – 1259, 2009.

[36] LIU, F., LIN, H. "Reliable Decentralized Supervisors for Discrete-Event Systems Under Communication Delays: Existence and Verification", *Asian Journal of Control*, v. 15, n. 5, pp. 1346–1355, 2013.

[37] PARK, S. J., CHO, K. H. "Nonblocking supervisory control of timed discrete event systems under communication delays: The existence conditions", *Automatica*, v. 44, n. 4, pp. 1011 – 1019, 2008.

[38] 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.

[39] XU, S., KUMAR, R. "Asynchronous implementation of synchronous discrete event control". In: *9th International Workshop on Discrete Event Systems*, pp. 181–186, Göteborg, Sweden, 2008.

[40] LIN, F. "Control of networked discrete event systems". In: *24th Chinese Control and Decision Conference*, pp. 51–56, Taiyuan, China, 2012.

[41] SHU, S., LIN, F. "Decentralized control of networked discrete event systems with communication delays", *Automatica*, v. 50, n. 8, pp. 2108 – 2112, 2014.

[42] SHU, S., LIN, F. "Supervisor Synthesis for Networked Discrete Event Systems With Communication Delays", *IEEE Transactions on Automatic Control*, v. 60, n. 8, pp. 2183–2188, 2015.

[43] KOMENDA, J., LIN, F. "Modular supervisory control of networked discrete-event systems". In: *13th International Workshop on Discrete Event Systems (WODES)*, pp. 85–90, Xi'an, China, 2016.

[44] WANG, F., SHU, S., LIN, F. "Robust Networked Control of Discrete Event Systems", *IEEE Transactions on Automation Science and Engineering*, v. 13, n. 4, pp. 1528–1540, 2016.

[45] SHU, S., LIN, F. "Predictive Networked Control of Discrete Event Systems", *IEEE Transactions on Automatic Control*, v. 62, n. 9, pp. 4698–4705, 2017.

[46] SHU, S., LIN, F. "Deterministic Networked Control of Discrete Event Systems With Nondeterministic Communication Delays", *IEEE Transactions on Automatic Control*, v. 62, n. 1, pp. 190–205, 2017.

[47] ZHAO, B., LIN, F., WANG, C., et al. "Supervisory Control of Networked Timed Discrete Event Systems and Its Applications to Power Distribution Networks", *IEEE Transactions on Control of Network Systems*, v. 4, n. 2, pp. 146–158, 2017.

[48] HUO, Z., FANG, H., MA, C. "Networked control system: state of the art". In: *5th World Congress on Intelligent Control and Automation*, v. 2, pp. 1319–1322, Hangzhou, China, 2004.

[49] NUNES, C. E. V., MOREIRA, M. V., ALVES, M. V. S., et al. "Network co-diagnosability of Discrete-Event Systems subject to event communication delays". In: *13th International Workshop on Discrete Event Systems*, pp. 217–223, Xi'an, China, 2016.

[50] DEBOUK, R., LAFORTUNE, S., TENEKETZI, D. "On the Effect of Communication Delays in Failure Diagnosis of Decentralized Discrete Event Systems", *Discrete Event Dynamic Systems*, v. 13, n. 3, pp. 263–289, 2003.

[51] ROHLOFF, K. R. "Sensor Failure Tolerant Supervisory Control". In: *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 3493–3498, Seville, Spain, 2005.

[52] SÁNCHEZ, A. M., MONTOYA, F. J. "Safe supervisory control under observability failure", *Discrete Event Dynamic Systems*, v. 16, n. 4, pp. 493–525, 2006.

[53] ALVES, M. V. S., BASILIO, J. C., DA CUNHA, A. E. C., et al. "Robust Supervisory Control Against Intermittent Loss of Observations". In: *12th Workshop on Discrete Event Systems*, pp. 294–299, Cachan, France, 2014.

[54] TRIPAKIS, S. "Decentralized control of discrete-event Systems with bounded or unbounded delay communication", *IEEE Transactions on Automatic Control*, v. 49, n. 9, pp. 1489–1501, 2004.

[55] HIRAISHI, K. "On Solvability of a Decentralized Supervisory Control Problem With Communication", *IEEE Transactions on Automatic Control*, v. 54, n. 3, pp. 468–480, 2009.

[56] SADID, W. H., RICKER, L., HASHTRUDI-ZAD, S. "Robustness of synchronous communication protocols with delay for decentralized discrete-event control", *Discrete Event Dynamic Systems*, v. 25, n. 1, pp. 159–176, 2015.

[57] ZHANG, R., CAI, K., GAN, Y., et al. "Distributed supervisory control of discrete-event systems with communication delay", *Discrete Event Dynamic Systems*, v. 26, n. 2, pp. 263–293, 2016.

[58] ZHANG, R., CAI, K., GAN, Y., et al. "Delay-robustness in distributed control of timed discrete-event systems based on supervisor localisation", *International Journal of Control*, v. 89, n. 10, pp. 2055–2072, 2016.

[59] QIU, W., KUMAR, R. "Distributed Diagnosis Under Bounded-Delay Communication of Immediately Forwarded Local Observations", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 38, n. 3, pp. 628–643, 2008.

[60] TAKAI, S., KUMAR, R. "Distributed Failure Prognosis of Discrete Event Systems With Bounded-Delay Communications", *IEEE Transactions on Automatic Control*, v. 57, n. 5, pp. 1259–1265, 2012.

[61] ATHANASOPOULOU, E., LI, L., HADJICOSTIS, C. N. "Maximum Likelihood Failure Diagnosis in Finite State Machines Under Unreliable Observations", *IEEE Transactions on Automatic Control*, v. 55, n. 3, pp. 579–593, 2010.

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

[63] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., et al. "Robust diagnosis of discrete-event systems against permanent loss of observations", *Automatica*, v. 49, n. 1, pp. 223 – 231, 2013.

[64] KANAGAWA, N., TAKAI, S. "Diagnosability of discrete event systems subject to permanent sensor failures", *International Journal of Control*, v. 88, n. 12, pp. 2598–2610, 2015.

[65] YIN, X., LI, Z. "Reliable Decentralized Fault Prognosis of Discrete-Event Systems", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 46, n. 11, pp. 1598–1603, 2016.

[66] VIANA, G. S., ALVES, M. V. S., BASILIO, J. C. "Codiagnosticabilidade de sistemas a eventos discretos em rede com informações de temporização entre ocorrências de eventos e atrasos nos canais de comunicacão". In: *Simpósio Brasileiro de Automação Inteligente*, pp. 1247–1254, Porto Alegre, Brazil, 2017.

[67] VIANA, G. S., ALVES, M. V. S., BASILIO, J. C. "Codiagnosability of Timed Networked Discrete-Event Systems subject to event communication delays and intermittent loss of observation". In: *56th IEEE Conference on Decision and Control*, Melbourne, Australia, to be presented in December, 2017.

[68] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2nd ed. New York, Springer, 2008.

[69] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 2006.

[70] MURATA, T. "Petri nets: Properties, analysis, and applications", *Proceedings of the IEEE*, v. 77, n. 4, pp. 541–580, 1989.

[71] DAVID, R., ALLA, H. *Discrete, continuous and hybrid Petri nets*. New York, Springer, 2005.

[72] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Sba Controle & Automação*, v. 21, n. 5, pp. 510–533, 2010.

[73] CHO, H., MARCUS, S. I. "Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations", *Mathematical systems theory*, v. 22, n. 1, pp. 177–211, 1989.

[74] CHO, H., MARCUS, S. I. "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation", *Mathematics of Control, Signals and Systems*, v. 2, n. 1, pp. 47–69, 1989.

[75] JIRÁSKOVÁ, G., MASOPUST, T. "On properties and state complexity of deterministic state-partition automata". In: *International Conference on Theoretical Computer Science*, pp. 164–178, Amsterdam, The Netherlands, 2012. Springer.

[76] SCHUTZENBERGER, M. "On the definition of a family of automata", *Information and Control*, v. 4, n. 2-3, pp. 245–270, 1961.

[77] WONHAM, W. M. "Supervisory control of discrete-event systems", http://www.control.utoronto.ca/cgi-bin/dldes.cgi, 2017.

[78] TSITSIKLIS, J. N. "On the control of discrete-event dynamical systems", *Mathematics of Control, Signals and Systems*, v. 2, n. 2, pp. 95–107, 1989.

[79] RUDIE, K., WONHAM, W. M. "The infimal prefix-closed and observable superlanguange of a given language", *Systems & Control Letters*, v. 15, n. 5, pp. 361 – 371, 1990.

[80] KUMAR, R., SHAYMAN, M. A. "Formulae relating controllability, observability, and co-observability", *Automatica*, v. 34, n. 2, pp. 211 – 215, 1998.

[81] MASOPUST, T. "Complexity of Infimal Observable Superlanguages", DOI: 10.1109/TAC.2017.2720424, 2017.

[82] ALVES, M. V. S., BASILIO, J. C., CARVALHO, L. K., et al. "Algoritmos para verificação da observabilidade relativa e cálculo da sublinguagem observável relativa suprema". In: *Simpósio Brasileiro de Automação Inteligente*, pp. 1724–1729, Natal, Brazil, 2015.

[83] ALVES, M. V. S., CARVALHO, L. K., BASILIO, J. C. "New algorithms for verification of relative observability and computation of supremal relatively observable sublanguage". In: *IEEE Conference on Control Applications*, pp. 526–531, Buenos Aires, Argentina, 2016.

[84] ALVES, M. V. S., CARVALHO, L. K., BASILIO, J. C. "New Algorithms for Verification of Relative Observability and Computation of Supremal Relatively Observable Sublanguage", *IEEE Transactions on Automatic Control*, v. 62, n. 11, pp. 5902–5908, 2017.

[85] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684, 2011.

[86] SHU, S., LIN, F., YING, H. "Detectability of Discrete Event Systems", *IEEE Transactions on Automatic Control*, v. 52, n. 12, pp. 2356–2359, 2007.

[87] WONHAM, W. M., RAMADGE, P. J. "On the Supremal Controllable Sublanguage of a Given Language", *SIAM Journal on Control and Optimization*, v. 25, n. 3, pp. 637–659, 1987.

[88] ALVES, M. V. S., CARVALHO, L. K., BASILIO, J. C. "Controle supervisório de sistemas a eventos discretos em rede temporizados". In: *Simpósio Brasileiro de Automação Inteligente*, pp. 1382–1389, Porto Alegre, Brazil, 2017.

[89] ALVES, M. V. S., CARVALHO, L. K., BASILIO, J. C. "Supervisory Control of Timed Networked Discrete Event Systems". In: *56th IEEE Conference on Decision and Control*, Melbourne, Australia, to be presented in December, 2017.