COPPE
UFRJ
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

NEW APPROACHES TO FAULT PREDICTION AND OPACITY ENFORCEMENT OF DISCRETE-EVENT SYSTEMS

Raphael Julio Barcelos

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
Março de 2022

NEW APPROACHES TO FAULT PREDICTION AND OPACITY ENFORCEMENT OF DISCRETE-EVENT SYSTEMS

Raphael Julio Barcelos

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

Orientador: João Carlos dos Santos Basilio

Aprovada por: Prof. João Carlos dos Santos Basilio
Prof. Marcos Vicente de Brito Moreira
Prof. José Eduardo Ribeiro Cury
Prof^{a}. Patrícia Nascimento Pena
Prof. Antonio Eduardo Carrilho da Cunha

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2022

Barcelos, Raphael Julio

New approaches to fault prediction and opacity enforcement of discrete-event systems/Raphael Julio Barcelos. – Rio de Janeiro: UFRJ/COPPE, 2022.

XIII, 128 p.: il.; $29,7$cm.

Orientador: João Carlos dos Santos Basilio

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2022.

Referências Bibliográficas: p. 118 – 128.

1. Discrete event systems. 2. Opacity. 3. Opacity-enforcement. 4. Utility. 5. Copredictability. 6. Fault prediction. I. Basilio, João Carlos dos Santos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

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

NOVAS ABORDAGENS PARA PREDIÇÃO DE FALHAS E FORÇAMENTO DE OPACIDADE EM SISTEMAS A EVENTOS DISCRETOS

Raphael Julio Barcelos

Março/2022

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

Este trabalho se concentra no desenvolvimento de novas abordagens para problemas relacionados a duas propriedades de sistemas a eventos discretos: opacidade e preditibilidade. A opacidade requer que um comportamento secreto do sistema seja escondido de um intruso, e, neste quesito, o primeiro problema abordado é como tornar opaco um sistema que inerentemente não é (forçamento de opacidade). Para esse fim, propõe-se uma estratégia capaz de manipular a estimação de estados do intruso de forma que a ordem da observação dos eventos seja embaralhada, podendo alguma dessas observações ser excluídas quando estritamente necessário. Investiga-se também como mitigar o efeito negativo do forçamento de opacidade na capacidade do receptor autêntico de estimar o estado atual do sistema com precisão. O segundo problema abordado é relacionado à principal crítica às estratégias de forçamento de opacidade, mais especificamente o fato de que, ao ofuscar o comportamento secreto do sistema ao intruso, algumas das informações transmitidas também são escondidas do receptor autêntico. Neste respeito, introduz-se a noção de utilidade (baseada em estados), a qual se refere a algum comportamento crucial do sistema que deve estar sempre disponível para o receptor autêntico, mesmo quando a opacidade estiver sendo forçada. O último problema abordado é a predição de falhas. Um sistema é preditível se, para todo comportamento de falha, é possível previamente ter certeza da ocorrência da falha. Na sequência, apresentam-se duas estratégias para a verificação de preditibilidade disjuntiva de falhas (copreditibilidade de falhas): a primeira baseada em autômatos teste e a segunda, em verificadores. São abordados também a predição online de falhas e, para tanto, é apresentada uma estratégia para o projeto de preditores locais de falha, e a K-copreditibilidade, que requer que todas as falhas sejam preditas pelo menos K eventos antes de suas ocorrências.

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

NEW APPROACHES TO FAULT PREDICTION AND OPACITY ENFORCEMENT OF DISCRETE-EVENT SYSTEMS

Raphael Julio Barcelos

March/2022

Advisor: João Carlos dos Santos Basilio

Department: Electrical Engineering

In this work, we are concerned with developing new approaches to problems related to two discrete-event system properties, namely, opacity and predictability. Opacity is a property that ensures that a secret behavior of the system is kept hidden from an intruder, and, in this regard, the first problem we address is how to make opaque a system which inherently is not (opacity enforcement). To this end, we propose a strategy which is capable of manipulating the state estimation of the intruder by shuffling event observations and also deleting some of them when strictly necessary. We also investigate how it is possible to mitigate the negative effect of opacity enforcement on the capability of the legitimate receiver to accurately estimate the current state of the system. The second opacity problem addressed in this work is related to the main criticism of the opacity enforcement strategies existing in the literature, namely that in order to obfuscate the secret behavior of the system from intruders, some transmitted information is also concealed from the legitimate receiver. In this regard, we introduce the notion of (state-based) utility to refer to some crucial behaviors that must always be available to the legitimate receiver, even when opacity is being enforced. The last problem we address is fault predictability, where a system is predictable with respect to the fault event if, for every faulty behavior, we can be sure of the fault prior to its occurrence. We then present two strategies for the verification of disjunctive fault predictability (fault copredictability), one that is based on a diagnoser-like test automaton, and the second one that deploys verifiers. We also consider online fault prediction, present a strategy for designing local fault predictor systems, and address K-copredictability, namely, if all fault occurrences can be predicted at least K events prior to their occurrences.

Contents

List of Figures

List of Symbols

$Ac(G)$	Accessible part of automaton G, p. 13
$CoAc(G)$	Coaccessible part of automaton G, p. 13
D	Automaton which models changes in the order of event observations, p. 42
$Dil(L)$	Dilation operation over languages, p. 30
$G_1 \times G_2$	Product between automata G_1 and G_2, p. 14
$G_{est,R}$	Automaton that models the state estimation of the receiver, p. 57
$G^e_{est,R}$	Automaton that models the improved state estimation of the receiver, p. 59
$G_{est,int}$	Automaton that models the state estimation of the intruder, p. 57
G_e	Automaton whose generated language corresponds to the actual sequences to be executed by the system G, p. 59
G_{obs}	Observer automaton $\mathrm{Obs}(G, \Sigma_o)$, p. 14
L/s	Post-language of L after a sequence s, p. 11
LP_i	i-th local predictor, p. 86
L_a	Augmented language, p. 34
L_{obs}	Language generated by the observer automaton G_{obs}, p. 14
$P_{a,b}(s)$	Projection of sequence $s \in \Sigma_a$ over Σ_b^*, p. 11
$P_{a,b}^{-1}(s)$	Inverse projection of sequence $s \in \Sigma_b$ over Σ_a^*, p. 11
$SD(k)$	Step delay bound set, p. 35

S_p	Sequence permutation, p. 29
$Trim(G)$	Trim automaton obtained from automaton G, p. 13
X_s	Set of secret states with respect to automaton G, p. 16
$X_{s,obs}$	Set of secret states with respect to automaton G_{obs}, p. 20
$\mathcal{A}(s_a)$	Language augmentation function, p. 33
$\varphi_d(s)$	Deletion function, p. 32
G_{scc}	Test automaton, p. 91
G_v	Verifier automaton, p. 97
$\mathcal{L}(G)$	Language generated by automaton G, p. 12
L_a^{OE}	Bounded delay augmented language that satisfies **OEC1-OEC2**, p. 39
L_a^{SD}	Bounded delay augmented language, p. 37
$\mathcal{L}_m(G)$	Language marked by automaton G, p. 12
$P_a(s_a)$	Projection of sequence $s_a \in \Sigma_a$ over Σ^*, p. 33
$P_d(s_a)$	Projection of sequence $s_a \in \Sigma_a$ over Σ_d^*, p. 33
$P_{dil}(s)$	Projection of sequence $s \in \Sigma_{dil}$ over Σ^*, p. 30
$P_o(s)$	Projection of sequence $s \in \Sigma$ over Σ_o^*, p. 14
$P_{o,i}(s)$	Projection of sequence $s \in \Sigma$ over $\Sigma_{o,i}^*$, p. 87
$P_r(s_a)$	Projection of sequence $s_a \in \Sigma_a$ over Σ_r^*, p. 33
$\Psi(\sigma)$	Set composed of all sequences in $L(G)$ whose last event is σ, p. 20
$\varphi_r(s)$	Release function, p. 32
$\varphi_r^{-1}(s)$	Inverse release function, p. 32
$R_i(\sigma)$	Renaming function, p. 97
R_i^{-1}	Inverse renaming function, p. 97
R_{OE}	Automaton that realizes the Opacity-Enforcer, p. 51

Σ	Event set, p. 11
Σ_D	Set of observable events whose observations are allowed to be deleted, p. 29
Σ_a	Augmented event set, p. 32
Σ_d	Set of deleted observations, p. 29
Σ_{dil}	Set of events that can be dilated, p. 29
Σ_f	Set of fault events, p. 21
Σ^*	Kleene-Closure of event set Σ, p. 11
Σ_o	Set of observable events, p. 14
$\Sigma_{o,i}$	Set of local observable events, p. 86
Σ_{uo}	Set of unobservable events, p. 14
X_u	Set of useful states, p. 62
$\chi(s)$	Rearrangement function, p. 45
$pre(s_a, \sigma^{i_\sigma})$	Prefix of an augmented sequence $s_a \in \Sigma_a^*$ whose last event is the i_σ-th occurrence of $\sigma \in \Sigma$, p. 36
$\mathbb{N}$	Non negative integers, p. 18
$\mathcal{N}(s, \sigma)$	Number of occurrences of event σ in sequence s, p. 32
$\text{Obs}(G, \Sigma_a)$	Observer automaton of G with respect to event set Σ_a, p. 14
$\overline{s}$	Prefix-closure of a sequence s, p. 11
$\varepsilon R(x)$	ε-reach, p. 14
ε	Empty sequence, p. 11
$cut(q)$	Cut function, p. 42
$rep(q, i)$	Replacement function, p. 42

List of Abbreviations

CSOE	Current-state opaque enforceable, p. 30
CSO	Current-state opacity, p. 16
DES	Discrete-event systems, p. 1, 10
EU-CSO	Ensured-utility current-state opaque, p. 70
EU	Ensured-utility, p. 62
FSA	Finite-state automaton, p. 10
IFO	Initial-and-final-state opacity, p. 19
ISO	Initial-state opacity, p. 17
LBO	Language-based opacity, p. 15
SCC	Strongly connected components, p. 22

Chapter 1

Introduction

Discrete-Event Systems (DES) are dynamic systems whose space state is a discrete set and its evolution is ruled by the asynchronous occurrence of events over time [1]. DES modeling formalisms have been proven to be a powerful tool to solve several problems, such as sensor failures [2–6], state estimation [7, 8], supervisory control [9–12] and cyber attacks [13–17], among many other topics. In order to approach the problems that arise in the field of automation and control, the DES community has been proposing and developing properties since then, for example: fault diagnosis [18, 19] and fault prediction (also referred to as prognosis) [20, 21], which are mostly concerned with identifying and foreseeing fault occurrences in the system, respectively; opacity notions [22–24], which are focused in privacy problems, where secret behaviors must be concealed from external agents; detectability [8, 25, 26], which is related to monitoring tasks, where the space states of the system must eventually be determined; controllability [27], observability [28] and relative observability [10, 11], which are developed in supervisory control theory [9], in the sense that the supervisory controller must never disable uncontrollable events and different behaviors that are equally observed must have the same control decision, respectively.

Among the problems the DES theory has been employed to solve, this work addresses opacity enforcement and fault predictability verification, and, as a consequence, it focuses on the properties of opacity and fault prediction, respectively, which will be reviewed in the two subsections as follows.

1.1 Opacity

Opacity is a property that ensures that a given secret behavior of the system is kept hidden from external observers with malicious intentions, usually referred to as Intruders, which are assumed to passively observe the information flow. It has emerged in the Discrete-Event Systems (DES) community [22] as a convenient way to deal with a class of security problems.

Different notions of opacity have been formulated based on how the secret behavior of the system is defined [23, 24]. The most usual ones are: strong/weak opacity [23], language-based opacity (LBO) [29, 30], current-state opacity (CSO) [31], initial-state opacity (ISO) [32], K-step opacity [31], and infinite-step opacity [33]. Another notion is the initial-and-final-state opacity (IFO) [24], which generalizes both CSO and ISO.

After these opacity notions having emerged, the first problem that the DES community has been concerned with is opacity verification, that is, given a system and a secret behavior, check whether opacity holds true or not. Early results on this topics have been presented by SABOORI and HADJICOSTIS [32, 33, 34, 35] and DUBREIL *et al.* [36]. These works differ from each other with respect to the opacity notion which is being considered, the strategy used to verify the considered notion, and also the computational cost to achieve the proposed verification. Notice however that the DES community still researches new strategies, for example: YIN and LAFORTUNE [37] improve both infinite-step and K-step opacity verification by using two-way observers, and; MA *et al.* [38] introduce the notion of strong infinite-step opacity and use state recognizers to verify this property and also strong K-step opacity, earlier proposed by FALCONE and MARCHAND [39].

Opacity has also been studied in other DES formalisms such as, timed systems [40, 41], stochastic automata [42–45], transition systems [46, 47], modular systems [48], networked DES [49], fuzzy DES [50], Petri-nets [51, 52] and linear time-invariant systems [53, 54]. Readers are referred to [55] and [56] for a more detailed overview on opacity topics.

Since not every system is inherently opaque, it is necessary to develop mechanisms to make the system opaque in accordance with some opacity notion of interest. In this regard, two major methods have been explored in the literature, which work as follows: (i) either they restrict the system so as its secret behavior is never inferred, (ii) or they modify the observation of the system for external agents.

The first mechanism for opacity enforcement is typically approached with the help of supervisory control theory (SCT), which is used to restrict the behavior of the system so as its secrets are not leaked, being, therefore, a conservative approach. In this regard, SABOORI and HADJICOSTIS [57], by using an *Initial State Estimator* (ISE), propose a supervisor that limits the system within a pre-specified legal behavior while enforcing ISO and disabling the least possible number of controllable events. SABOORI and HADJICOSTIS [58] improve the results of SABOORI and HADJICOSTIS [57] and extend them to infinite-step opacity. DUBREIL *et al.* [30] provide a reduction of LBO enforcement through SCT of partially observed DES to the same problem under full observation and then present a solution for the computation of a supremal controller that enforces LBO. YIN and LAFORTUNE [59] investigate CSO enforcement and propose the so-called *All Enforcement Structure* (AES), obtained by means of a game structure between the supervisor and the system, and thus, it embeds all valid supervisors, which allows the synthesis of a locally maximal supervisor that enforces CSO. TONG *et al.* [60] also investigates CSO enforcement by means of SCT for partially DES, but under incomparable observations, which means that no relation between the observation of the Intruder and the Supervisor can be established. The scenario where the Intruder may disclose the control policy and allow itself to recover the control decision made by the supervisor (and thus being able to obtain a better state-estimate of the system), is addressed by XIE *et al.* [61], who propose a non-deterministic supervisor that provides a set of control decisions at each time instant, where the specific control decision is chosen randomly.

Before presenting opacity enforcement strategies through modifications in the observation of the system, it is worth highlighting that BRYANS *et al.* [46] introduce three different models for the observation of a system: (i) static observation, where the event signals generated by the system that are observable by external agents are fixed; (ii) dynamic observation, where the observability of these event signals depends on predefined rules, for example, the already observed behavior, and; (iii) Orwellian observation, where the observability of event signals depends not only on the past behavior, but also on the future one.

With respect to static observation models for opacity enforcement, CASSEZ *et al.* [62] propose a static mask, which computes a maximum subset of the observable events (sensor selection) that makes the system opaque. FALCONE and MARCHAND [39] propose a strategy to enforce K-step opacity, where, depending on the already observed behavior, it may hold the incoming event signals for an exact amount of steps such that the system becomes opaque. Recently, DULCE-GALINDO *et al.* [63] extend the standard definitions of ISO and IFO to weak versions, and then, propose an opacity enforcement by means of synchronizing automata. BARCELOS and BASILIO [64] propose a CSO enforcement strategy that shuffles the order of event observations with a view for the secret behavior of the system to becoming indistinguishable from the non-secret one.

A different strategy for opacity enforcement is the insertion function, which has been proposed by WU and LAFORTUNE [65] to address CSO enforcement. The idea behind insertion functions is to allow the system to run freely, but misleading the Intruder to never infer that the secret behavior has occurred by inserting fictitious event signals in the output of the system. The procedure to obtain insertion functions has been improved by WU and LAFORTUNE [66], which also approach the problem of quantifying the insertion of events and provides the synthesis of the optimal insertion function. JI *et al.* [67] refer to the insertion function strategy deployed in [65, 66] as private, since the Intruder is unaware of it, and then they address the scenario where the Intruder may also know or discover the structure of the insertion function, therefore providing a strategy that enforces opacity even when it is publicly known, the so-called public-and-private opacity enforcement. In contrast with the previous works concerning insertion functions, KEROGLOU and LAFORTUNE [68] propose a new insertion function which is embedded into the system and acts based on the real location of the system, being a more powerful strategy than the previously related ones. JI *et al.* [69] address the problem of enforcing CSO through insert functions into a system with energy level that can be consumed or increased depending on the event occurrences and insertions. KEROGLOU *et al.* [70] propose CSO enforcement through insertion functions enhanced with finite memory, where instead of immediately releasing the modified behavior, it stores a predefined number of consecutive events to only then output a modified information.

Later, the insertion function has been extended to edit functions by WU *et al.* [71], which allows not only the insertion of event observations but also its deletion. In the sequel, JI and LAFORTUNE [72] improve the edit function so as to enforce opacity even when the strategy is publicly known. JI *et al.* [73] improve the strategies proposed in [67, 72] by considering opacity enforcement using nondeterministic public known edit functions. WINTENBERG *et al.* [74] investigate K-step opacity enforcement through edit functions. Recently, LI *et al.* [75] also extend the insertion function, but, differently from edit functions, the extended insertion function allows fictitious event signals to be inserted not only before genuine event occurrences but also after them.

With respect to opacity enforcement strategies with dynamic observation models, CASSEZ *et al.* [62] extend the notion of CSO to dynamic observation and investigate the problem of synthesizing a dynamic mask that makes the system opaque, since it selects if an event is observable based on the previous behavior of the system. ZHANG *et al.* [76] propose a controller that makes some events unobservable to ensure weak/strong opacity, where the observation deletions are as least as possible in order to release the maximum information. BEHINAEIN *et al.* [77] improve the results obtained by ZHANG *et al.* [76] and also investigate the problem of making the estimation of some states opaque and, at the same time, the estimation of other states non-opaque. YIN and LI [78] address the dynamic masks synthesis problem for infinite-step opacity.

Finally, as far as the author of this work knows, opacity under Orwellian observation has only been investigated by HOU *et al.* [79], where a strategy for the verification of CSO is proposed.

Contribution of this work on opacity enforcement

We consider in this work the problem of CSO enforcement, where a system is said to be current-state opaque if the Intruder is unable to ascertain if the current state of the system is a secret one or not. In this regard, in order to deal with systems which are not inherently current-state opaque, the enforcement strategy we propose here is based on shuffling the order of observations of the events that have occurred in the system, and also observation deletions. The idea behind this strategy is to mislead the Intruder to never be certain that the system is currently in a secret state by making the Intruder always believes that the system is in a non-secret state. This strategy is implemented by means of the so called Opacity-Enforcer, which is placed between the plant and the legitimate receiver, before the point where information is likely to leak, *i.e.*, be observed by the Intruder.

The Opacity-Enforcer works as follows [80]: when it receives a signal associated with an event occurrence, it chooses, based on preset rules, either to release it, release another stored signal, or hold all signals until the arrival of other events, with a view to changing the order of the observation of the events. The Opacity-Enforcer is also allowed to delete the observation of some held event, which is performed in order for the sequence of released signals to be consistent with the behavior of the system. The events released by the Opacity-Enforcer are transmitted to a legitimate receiver through a network, which is assumed to be susceptible to leak information to Intruders. The Opacity-Enforcer proposed here is obtained by adapting the automaton developed by NUNES *et al.* [6] and combining it with the system automaton, in order for the shuffled sequences to remain within the language generated by the system, and with the automaton that models the allowed observation by the intruder. In addition, in order to deal with observation deletion, the dilation operation proposed by CARVALHO *et al.* [3] is used.

Since opacity enforcement strategy presented in this work is likely to also mislead the state estimation of legitimate receivers, we present a protocol that is capable of mitigating this negative effect on the capability of the legitimate receiver to accurately estimate the current state of the system. This protocol leverages the legitimate receiver's knowledge on the actions to be taken by the Opacity-Enforcer in order to refine the estimation of the current state of the system.

We also address the main criticism of opacity enforcement strategies existing in the literature, namely that in order to obfuscate the secret behavior of the system from intruders, some transmitted information is also concealed from the legitimate receiver. In this regard, we introduce a notion of (state-based) utility of the system to refer to some crucial behavior that must always be available to the legitimate receiver, even when opacity is being enforced.

It is worth remarking that the notion of utility within opacity enforcement strategies has been presented firstly in [71], where the authors synthesize obfuscation policies through edit functions that ensure privacy and utility at the same time. However, in their approach, the authors define a value for each pair composed of the current state of the plant and the state estimated by external observers, and then, they impose that the utility of the system is preserved when these values never exceed a predefined maximum value. Differently from the strategy proposed in [71], in this work, the utility of the system is based on the visit of the so-called useful states, and, in addition, we require that the event observations outputted by the Opacity-Enforcer are such that, when some useful state is reached, it must be estimated before a new state is reached, and also that whenever the receiver estimates an useful state, the plant is currently in that state.

1.2 Fault prediction

Predictability, also referred to as prognosis, is a DES property whose widely adopted definition has been presented in GENC and LAFORTUNE [20]. The predictability aims to foresee an event occurrence in a system before its actual occurrence. It differs from diagnosability in the sense that, whereas diagnosability ensures that the occurrence of an unobservable event (usually referred to as fault) is always detected within a finite number of event observations after its occurrence, predictability means that we can always ascertain that such an event will inevitably occur prior to its actual occurrence.

The notion of predictability was introduced by JIANG and KUMAR [21] as pre-diagnosability, and was based on state-traces. The widely adopted definition of predictability was introduced later on by GENC and LAFORTUNE [20], and is related to the possibility of being sure about future occurrences of the fault event based on the observation of sequences that do not contain the fault event, i.e., before its actual occurrence. Differently from JIANG and KUMAR [21], predictability is defined in GENC and LAFORTUNE [20] based on event sequences.

After the aforementioned works, fault prectability has become a research topic of wide interest in the discrete-event community. TAKAI [81] considers predictability in the presence of deadlocks, and introduces the notion of robust predictability, where a single predictor which is able to foresee the fault occurrence for all possible models of observations is synthesized. More recently, BARCELOS *et al.* [82] discus the problem of predictability verification of discrete-event systems (DES) modeled by finite state automata in less restrictive scenarios, where the assumptions regarding language liveness and absence of cycles of states connected by unobservable events only are both dropped. XIAO and LIU [83] have approached the problem of robust fault predictability against intermittent/permanent losses of observations. WATANABE *et al.* [84] propose a verification strategy for safe fault diagnosability and predictability, which, in the case of predictability, requires that every fault is predictable and a predefined illegal behavior is inhibited from occurring by disabling some controllable event after the fault prediction.

As in fault diagnosis, predictability has also been addressed in decentralized architectures, where local agents work in a cooperative way to predict fault occurrences [85–89]. In KUMAR and TAKAI [85], predictability is approached in a decentralized architecture and the definition proposed by GENC and LAFORTUNE [20] is referred to as "uniformly bounded prognosability", and later on, they introduce an weaker version of coprognosability. KUMAR and TAKAI [85] also propose the notion of reaction bound, which calculates the largest number of steps after the prognostic decision in which the fault may occur, and propose a copredictability verification test for regular languages that is based on verifiers and and on the so-called indicator strings. TAKAI and KUMAR [86] consider a distributed architecture assuming that the communication between the system and predictors are limited and subject to delay. In KHOUMSI and CHAKIB [87], the decentralized architecture presented in KUMAR and TAKAI [85] is referred to as disjunctive fault predictability, *i.e.*, the predictor system is sure that the fault will occur if, at least, one of the local predictors is sure. Then, the authors propose a conjunctive and a mixed architecture that embeds both the disjunctive and conjunctive features. In YIN and LI [88], it is evaluated how soon the fault can be predicted and, at the same time, it ensures that the fault will occur in a bounded time after its prediction. YIN and LI [89] address the loss of local predictability decisions in the communication between the local predictors and coordinator.

Recently, YIN and LI [90] propose two new decentralized protocols for fault predictability verification, namely, the positive state-estimate (PSE) based protocol, where each local agent sends a state estimation, which is believed that a fault will certainly occur in the future, and the negative state-estimate (NSE) based protocol, where the local agents provide estate-estimations which they believe that the fault alarm should not be issued; notice that, in both cases, the verdict of the fault occurrence is given by a coordinator that takes into account the decisions of the local agents. ZHOU *et al.* [91] address the problem, in a decentralized architecture, where any fault should be predicted K steps before its occurrence, and, once the fault alarm is issued, the fault must definitely occur within M steps, and introduce three notions of predictability ((M, K)-disjunctive-coprognosability, (M, K)-conjunctive-coprognosability and (M, K)-strongly-coprognosability) and establish algebraic structures to verify these notions.

It is worth noting that predictability has also been addressed in stochastic DES

[92–94], Petri-nets [95–98] and timed systems [99]. We refer the readers to WATANABE *et al.* [100] for an overview on fault prediction of DES.

Contribution of this work on fault predictability

This work addresses the problem of disjunctive predictability verification and online fault prediction of DES in less restrictive scenarios [101], *i.e.*, without making the usual assumptions of language liveness and absence of cycles of states connected by unobservable events. To this end, we adapt the test automaton and the verifier, proposed in VIANA and BASILIO [102] and VIANA *et al.* [103], respectively, which are used to verify codiagnosability, in order to develop two new strategies to verify disjunctive predictability, as follows: (i) the first one, based on the test automaton[102], and; (ii) the second one, obtained by using verifiers [104, 105].

Since the aforementioned methods embrace more general DES classes, the results presented here supersede those presented in GENC and LAFORTUNE [20]. The choice for a diagnoser-based strategy is backed up by [106], where it is claimed that, based on a rigorous statistical analysis, the size of the states of diagnosers and verifiers are, respectively, $\Theta(n^{0.77\log k+0.63})$ and $\Theta(n^2)$, on the average, where k (resp. n) is the number of events (resp. states) of the plant automaton. This means that, for systems with 60 events or less, the number of states of diagnosers is likely to be smaller than that of the corresponding verifier. Therefore, it is worth developing verification algorithms based not only on verifiers but also on diagnoser-based automata.

Another contribution of this work is the development of a strategy to design local fault predictor systems, to be used in online fault prediction. As opposed to the online fault prediction approached in KUMAR and TAKAI [85] and GENC and LAFORTUNE [20], our method is neither based on indicator-string nor on indicator-states but on finding the smallest sequences that allow the local fault predictors to foresee the fault occurrences. Leveraging the knowledge on these sequences, we address the problem of finding the minimum number of (either observable or unobservable) event occurrences after the fault is predicted by some local fault predictor and prior to the actual fault occurrence, and formulate the K-copredictability problem, i.e., if the fault can be predicted at least K event occurrences prior to its actual occurrence.

1.3 Thesis structure

We have structured this work as follows. In Chapter 2, we present a brief review on DES, which includes basic concepts and the properties we discuss in the further chapters. In Chapter 3, we address the problem of current-state opacity enforcement, where we manipulate the state estimation of the Intruder by shuffling event observations and also deleting some of these event observations when strictly necessary. Still in Chapter 3, we show that, by allowing the legitimate receiver to know the operations that are being performed to make the system opaque, it is possible to mitigate the damage caused to the state estimation of the legitimate receiver. In Chapter 4, with a view to approaching the main criticism of opacity enforcement strategies, we introduce the notion of utility, which is a behavior of the system that the legitimate receiver must unambiguously infer from the system. In this regard, we improve the strategy presented in Chapter 3 so as the Opacity-Enforcer has now the task of concealing the secret behavior of the system from intruders and, at the same time, ensure that the useful behavior of the system is available to the legitimate receiver. In Chapter 5, we approach the problem of disjunctive predictability verification and online fault prediction of DES in less restrictive scenarios, where we develop two algorithms to solve disjunctive predictability verification and another one to perform the online fault prediction. In addition, we also introduce the notion of K-copredictability and propose an algorithm for its verification. Finally, the contributions we have made in this work with respect to opacity enforcement, utility ensuring and fault copredictability are presented not only at the end of Chapters 3, 4 and 5, respectively, but also summarized in Chapter 6, where we also present possible extensions of the topics developed in this work.

Chapter 2

Theoretical background

This chapter is intended to review some of the basic concepts to be used in this work, mostly concerning finite-state automaton (FSA), its operations and properties. To this end, we have structured this chapter as follows. In Section 2.1, we present a brief review on discrete-event systems theory, and; in Section 2.2, we present a few properties of DESs.

2.1 Discrete-event systems theory

In contrast to the continuous variable systems, which are characterized by continuous-states and time-driven transitions, discrete-event systems (DES) are characterized by having as state space a discrete set, and the transitions between states driven by the occurrence of events [1]. These events, for example, denote specific actions that the system may take or are associated with occurrences of spontaneous nature. The occurrence of events, generally, causes the system state to change, and are assumed to be instantaneous. Events are typically denoted by lowercase letters. For example, the symbols "a", "b", "σ", "μ" often denote events.

We define a discrete-event system as follows.

Definition 2.1 (Discrete-event system) *A Discrete-event system (DES) is a discrete-state event-driven system, that is, its state space is formed with a discrete set and its state evolution depends entirely on the occurrence of asynchronous discrete events over time.* □

The collection of events is called the set of events (also referred to as "alphabet"), that is finite and is denoted as Σ, *e.g.*, $\Sigma = \{a, b, c\}$. Since the occurrence of successive events, which may or may not be different, composes the behavior of a DES, we define *"sequence"* of events as the concatenation of events (also called *"word"*, *"string"* or *"trace"* in the literature) and is denoted as s. For example,

$s_1 = abc$, $s_2 = ca$ are possible sequences formed with the events of $\Sigma = \{a, b, c\}$. The *"length of a sequence"* is the number of events that form it, denoted as $\|s\|$, *e.g.*, $\|s_1\| = 3$ and $\|s_2\| = 2$. In addition, $|A|$ denotes the cardinality of the set A, which is the number of elements that form set A. The "*empty trace*", which is denoted by ε, is the sequence with no events, and thus, its length is zero, *i.e.*, $\|\varepsilon\| = 0$. Transitions that are labeled with event ε are referred to as "*silent transitions*". Finally, the behavior of a DES can be modeled by using *"languages"*, which is a set formed with finite-length (bounded) sequences and whose formal definition is as follows.

Definition 2.2 *A language L defined over an event set Σ is a set of finite-length sequences formed with events $\sigma \in \Sigma$, including the empty trace ε.* □

The *Kleene-Closure* of an event set Σ is denoted by Σ^* and is a language composed of all possible sequences with finite length formed with the events of Σ, including ε. It can be defined as $\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma\Sigma \cup \ldots$. Note that Σ^* is discrete but infinitely countable.

Before moving on to the operations that can be performed with sequences, we must first clarify some terminologies. Let $s = tuv$, where $s, t, u, v \in \Sigma^*$. We call t a prefix of s, u a substring of s and v a suffix of s.

We now present some useful operations on sequences used throughout the text:

- *Prefix-closure* of a sequence s, defined as $\overline{s} = \{u \in \Sigma^* : (\exists v \in \Sigma^*)[uv = s]\}$.

- *Natural projection* [9], or simply projection, of a larger event set Σ_a over a smaller event set Σ_b, *i.e.*, $\Sigma_b \subseteq \Sigma_a$, is defined as $P_{a,b} : \Sigma_a^* \to \Sigma_b^*$, where $P_{a,b}(\varepsilon) = \varepsilon$, $P_{a,b}(\sigma) = \sigma$ if $\sigma \in \Sigma_b$, $P_{a,b}(\sigma) = \varepsilon$ if $\sigma \in \Sigma_a \setminus \Sigma_b$ and $P_{a,b}(s\sigma) = P_{a,b}(s)P_{a,b}(\sigma)$, where $s \in \Sigma_a^*$, $\sigma \in \Sigma_a$.

- *Inverse projection*, defined as $P_{a,b}^{-1} : \Sigma_b^* \to 2^{\Sigma_a^*}$, where $P_{a,b}^{-1}(t) = \{s \in \Sigma_a^* : P_{a,b}(s) = t\}$ and $\Sigma_b \subseteq \Sigma_a$.

The *prefix-closure*, *projection* and *inverse projection* can be extended over a language L by applying them to each sequence of L. Another useful operation on languages used throughout the text is the *post-language*.

- *Post-language* of L after a sequence s is defined as $L/s = \{t \in \Sigma^* : st \in L\}$. Notice that, by definition, if $s \notin \overline{L}$, then $L/s = \emptyset$.

The formalism used here to model DES is the so-called (deterministic) finite state automaton, which is defined as follows.

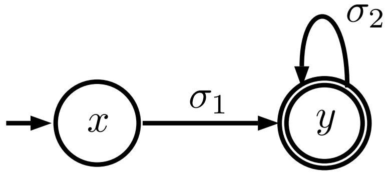


Figure 2.1: State transition diagram.

Definition 2.3 (Automaton [1]) *An automaton, denoted by* G*, is a six-tuple* $G = (X, \Sigma, f, \Gamma, x_0, X_m)$*, where* X *is the finite set of states,* Σ *is the finite set of events,* $f : X \times \Sigma \to X$ *is the partial state transition function,* $\Gamma : X \to 2^{\Sigma}$ *is the active event set function, being defined as* $\Gamma(x) = \{\sigma \in \Sigma : f(x, \sigma)!\}$*, where* $f(x, \sigma)!$ *denotes that* $f(x, \sigma)$ *is defined, i.e.,* $\exists y \in X$ *such that* $f(x, \sigma) = y$*,* $x_0 \in X$ *is the initial state and* $X_m \subseteq X$ *is the set of marked states.* □

It is worth remarking that, whenever there are no marked states ($X_m = \emptyset$), the automaton can be represented as the five-tuple $G = (X, \Sigma, f, \Gamma, x_0)$. In order to formally define the languages generated and marked by an automaton, let us extend the transition function to $f : X \times \Sigma^* \to X$ by the following recursion: $f(x, \varepsilon) = x$; $f(x, \sigma s) = f\big(f(x, \sigma), s\big)$; $\sigma \in \Sigma, s \in \Sigma^*$.

Automata are typically represented by directed graphs, also known as state transition diagrams, whose nodes and labeled arcs represent states and the transitions between them, respectively. In addition, marked states are represented by double circles whereas the initial state is represented by a circle with a single arrow pointing into it. Figure 2.1 shows the state transition diagram of automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, where $X = \{x, y\}$, $\Sigma = \{\sigma_1, \sigma_2\}$, $f(x, \sigma_1) = y$, $f(y, \sigma_2) = y$, $\Gamma(x) = \{\sigma_1\}$, $\Gamma(y), = \{\sigma_2\}$, $x_0 = x$ and $X_m = \{y\}$.

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

It is worth highlighting that the generated and marked languages are denoted as L and L_m, respectively, where the context allows.

Notice that languages formed with finite length sequences can be represented as automata. In this regard, we present the following definition.

Definition 2.5 (Regular Languages) *A language* L *is said to be regular if it can be marked by a finite-state automaton.*

Therefore, the automaton formalism is a practical tool for manipulating regular languages. On the other hand, languages that either have infinite length or are formed with infinite length sequences are called "*non-regular*" languages, and would require infinite states to be represented by automaton formalism.

We will now recall the main operations with automata. We start with the unary operations *accessible part*, *coaccessible part* and *trim*.

The *accessible part* removes all states that cannot be reached from the initial state and their associated transitions, being defined as:

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

where $X_{Ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$, $f_{Ac} : X_{Ac} \times \Sigma \to X_{Ac}$ where $f_{Ac}(x, \sigma) = f(x, \sigma)$ if $f(x, \sigma) \in X_{Ac}$, or undefined, otherwise, and $X_{m,Ac} = X_m \cap X_{Ac}$. States that have been removed when taking the accessible part of an automaton are called "not accessible".

The *coaccessible part* removes all states from which no marked state can be reached and their associated transitions, being defined as:

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

where $X_{CoAc} = \{x \in X : (\exists s, y \in \Sigma^* \times X_m)[f(x, s) = y]\}$, $f_{CoAc} : X_{CoAc} \times \Sigma \to X_{CoAc}$ where $f_{CoAc}(x, \sigma) = f(x, \sigma)$ if $f(x, \sigma) \in X_{CoAc}$ or undefined, otherwise, and $x_{0,CoAc} = x_0$ if $x_0 \in X_{CoAc}$ or empty, otherwise. States that have been removed when the coaccessible part of an automaton is calculated are called "not coaccessible".

The *Trim operation* results in an automaton that is both accessible and coaccessible, *i.e.*,

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

As a consequence, the states of automaton $Trim(G)$ are such that they are reached by an initial state and can also reach some marked state.

Following the unary operations, we now present the composition operations of automata, namely the *parallel composition* and the *product composition*.

The *parallel composition* of two automata results in a new one that synchronizes their common behavior and allows their individual behaviors to happen without constraints, being formally defined as:

$$G_1||G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1||2}, \Gamma_{1||2}, x_{0,1} \times x_{0,2}, X_{m,1} \times X_{m,2}),$$

where $f_{1||2}((x_1, x_2), \sigma) = (f_1(x_1, \sigma), x_2)$, if $\sigma \in \Gamma_1(x_1) \setminus \Sigma_2$, $f_{1||2}((x_1, x_2), \sigma) = (x_1, f_2(x_2, \sigma))$, if $\sigma \in \Gamma_2(x_2) \setminus \Sigma_1$, $f_{1||2}((x_1, x_2), \sigma) = (f_1(x_1, \sigma), f_2(x_2, \sigma))$ if $\sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$, or undefined, otherwise.

On the other hand, the *product* of two automata results in a new one that models their synchronized behavior, being formally defined as:

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, \Gamma_{1\times 2}, x_{0,1} \times x_{0,2}, X_{m,1} \times X_{m,2}),$$

where $f_{1\times 2}((x_1, x_2), \sigma) = (f_1(x_1, \sigma), f_2(x_2, \sigma))$ if $\sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$, or undefined, otherwise.

Finally, an automaton $G_{nd} = (X, \Sigma_{nd}, f_{nd}, \Gamma, X_0, X_m)$ is said to be nondeterministic if at least one of the following conditions holds true: (i) automaton G_{nd} has more than one initial state, *i.e.*, $|X_0| > 1$; (ii) there exist silent transitions defined in G_{nd}, *i.e.*, $\exists y \in X$ such that $f_{nd}(x, \varepsilon) = y$, or; (iii) the transition function is nondeterministic, *i.e.*, $|f_{nd}(x, \sigma)| > 1$, where $(x, \sigma) \in X \times \Sigma$. Notice that, in order for conditions (ii) and (iii) to be satisfied, Σ_{nd} may include event ε, *i.e.*, $\Sigma_{nd} = \Sigma \cup \{\varepsilon\}$, and $f_{nd} : X \times \Sigma \to 2^X$.

The *observer* automaton associated with a nondeterministic automaton G_{nd} is a language-equivalent deterministic automaton $\text{Obs}(G_{nd})$, for which $\mathcal{L}(\text{Obs}(G_{nd})) = P_{\Sigma_{nd},\Sigma}(\mathcal{L}(G_{nd})) = \mathcal{L}(G_{nd})$. In order to build the *observer*, let us present the *ε-reach* of a state $x \in X$, $\varepsilon R(x)$, which is the set composed of all states that can be reached from x by following transitions labeled with ε; by convention, $x \in \varepsilon R(x)$. It can be extended over a state set $Y \in 2^X$ as follows: $\varepsilon R(Y) = \bigcup_{x\in Y} \varepsilon R(x)$. Then, the *observer* of a nondeterministic automaton G_{nd} is defined as:

$$\text{Obs}(G_{nd}) = (X_{obs}, \Sigma, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{m,obs}),$$

where $X_{obs} \subseteq 2^X$, $f_{obs}(x_{obs}, \sigma) = \varepsilon R(\{x' \in X : (\exists x \in x_{obs})[x' \in f_{nd}(x, \sigma)]\})$, $\Gamma_{obs}(x_{obs}) = \bigcup_{x\in x_{obs}} \Gamma(x)$, $x_{0,obs} = \varepsilon R(X_0)$, and $X_{m,obs} = \{x_{obs} \in X_{obs} : x_{obs} \cap X_m \neq \emptyset\}$.

The *observer* automaton can be extended to partially-observed DES modeled by deterministic automata, which have "*unobservable transitions*" instead of silent transitions. In this case, the *observer* automaton associated with a deterministic automaton G and an event set $\Sigma' \subseteq \Sigma$ is denoted as $\text{Obs}(G, \Sigma')$. In order to compute the observer of a partially-observed automaton, we replace all transitions labeled by "unobservable events" $\sigma \in \Sigma \setminus \Sigma'$ with silent transitions (which makes automaton G nondeterministic), and then, we proceed with the aforementioned procedure to obtain $\text{Obs}(G, \Sigma')$. Notice that, the generated and marked languages by $\text{Obs}(G, \Sigma')$ are $\mathcal{L}(\text{Obs}(G, \Sigma')) = P_{\Sigma,\Sigma'}(\mathcal{L}(G))$ and $\mathcal{L}_m(\text{Obs}(G, \Sigma')) = P_{\Sigma,\Sigma'}(\mathcal{L}_m(G))$, respectively, where $P_{\Sigma,\Sigma'} : \Sigma^* \to \Sigma'^*$.

Let us assume that the event set Σ is partitioned into $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} are the sets of observable and unobservable events, respectively. In this case, we define the projection $P_o : \Sigma^* \to \Sigma_o^*$, and denote $G_{obs} = \text{Obs}(G_{nd}, \Sigma_o)$, $L_{obs} = \mathcal{L}(G_{obs}) = P_o(L)$ and $L_{m,obs} = \mathcal{L}_m(G_{obs}) = P_o(L_m)$, where the context allows.

2.2 Opacity and predictability of DES

In the following subsections, we present two properties of interest in this work: opacity and predictability.

2.2.1 Opacity

Opacity is a property which ensures that a given secret behavior of the system is kept hidden from external observers with malicious intentions, usually referred to as intruders, which are assumed to passively observe the information flow.

In this subsection, we present the usual notions of opacity that have been approached with automata formalisms in the literature. These notions are typically divided into language-based opacity and state-based opacity.

We start with the notion of language-based opacity (LBO), first presented by BADOUEL *et al.* [29], and formally defined by DUBREIL *et al.* [30]. In words, language-based opacity states that "*given a secret language $L_s \subseteq L$ and a set of observable events Σ_o, then every secret sequence in the secret language $s \in L_s$ must have the same observation as some non-secret sequence $t \in L \setminus L_s$*". Formally, language-based opacity is defined as follows.

Definition 2.6 (Language-Based Opacity) *Given a DES modeled by $G = (X, \Sigma, f, \Gamma, x_0)$, projection $P_o : \Sigma^* \to \Sigma_o^*$, and a secret language $L_s \subseteq L$, we say that G is language-based opaque with respect to Σ_o and L_s if $\forall s \in L_s$, there exists $t \in L \setminus L_s$ such that $P_o(s) = P_o(t)$, i.e., $P_o(L_s) \subseteq P_o(L \setminus L_s)$.* □

Example 2.1 *Let us consider a system modeled by an automaton G, whose generated language is $L = \overline{(abc + cab)}$, where $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{c\}$. Assume now that the secret language is $L_s = \{abc\}$. As a consequence, $L \setminus L_s = \{\varepsilon, a, ab, c, ca, cab\}$, which implies that G is language-based opaque, since $cab \in L \setminus L_s$ and $P_o(abc) = P_o(cab) = ab$.* □

Alternatively, LIN [23] defines LBO between two sublanguages of the system and introduces the notions of strong and weak opacities, depending on how much the intruder is able to infer from the secret behavior of the system, as follows.

Definition 2.7 (Strong and Weak Opacities) *Given a system modeled by $G = (X, \Sigma, f, \Gamma, x_0)$, whose generated language is L, a general mapping $\Theta : \Sigma_a^* \to \Sigma_b^*$, where $\Sigma_a, \Sigma_b \subseteq \Sigma$, and two languages L_1, $L_2 \subseteq L$, then:*

- *L_1 is strongly opaque with respect to L_2 and Θ if $\Theta(L_1) \subseteq \Theta(L_2)$;*
- *L_1 is weakly opaque with respect to L_2 and Θ if $\Theta(L_1) \cap \Theta(L_2) \neq \emptyset$.* □

It is not difficult to see from Definition 2.7 that, whenever $L_1 \neq \emptyset$ and $L_2 \neq \emptyset$, strong opacity implies weak opacity, but not conversely. We illustrate the definitions of strong and weak opacities with the following example.

Example 2.2 *Let $L = \overline{(abc + cab)}$ and consider the following sublanguages of L: $L_1 = \{abc\}$, $L_2 = \{a, ab, abc\}$, $L_3 = \{cab\}$. In addition, let $\Sigma = \{a, b, c\}$, $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{c\}$, and $\Theta = P_o : \Sigma^* \rightarrow \Sigma_o^*$. Notice that $P_o(L_1) = \{ab\}$, $P_o(L_2) = \{a, ab\}$, and $P_o(L_3) = \{ab\}$. Thus, we can say that L_1 is strongly opaque with respect to L_3 and $\Theta = P_o$, since $P_o(L_1) \subseteq P_o(L_3)$, and that L_2 is weakly opaque with respect to L_3 and $\Theta = P_o$, since $P_o(L_2) \cap P_o(L_3) = \{ab\} \neq \emptyset$. Note, however, that L_2 is not strongly opaque with respect to L_3 and $\Theta = P_o$, since $a \notin P_o(L_3)$.* □

It is worth recalling that LIN [23] defines opacity as a general property between languages with a view to showing that other properties — anonymity, secrecy, observability, diagnosability and detectability — could be seen as special cases of opacity upon properly defining languages L_1 and L_2. As a consequence, Definition 2.7 does not require a secret behavior and the presence of some intruder.

Notice that, when comparing Definitions 2.7 and 2.6, it is not difficult to see that language-based opacity is equivalent to strong opacity for the particular case where $L_1 = L_s$, $L_2 = L \setminus L_s$ and $\Theta = P_o$.

We will now consider the notions of state-based opacity. We start with the definition of current-state opacity (CSO), which has first been introduced by BRYANS *et al.* [22] in the context of Petri Nets and has been presented as a property of finite-state automata by SABOORI and HADJICOSTIS [31]. Broadly speaking, a system is current-state opaque when *"the intruder never knows if the system is actually in a secret state or not"*. We recall the formal definition of CSO provided by WU and LAFORTUNE [24], where the automaton G may have more than one initial state, therefore x_0 is replaced with $X_0 \in 2^X$ and automaton G is nondeterministic.

Definition 2.8 (Current State Opacity) *Given a system modeled by $G = (X, \Sigma, f, \Gamma, X_0)$, projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$ and a set of secret states X_s, then system G is current state opaque if $\forall x_0 \in X_0$ and $\forall s \in L$ satisfying $f(x_0, s) \in X_s$, $\exists \tilde{x}_0 \in X_0$ and $\exists \tilde{s} \in L$ such that $f(\tilde{x}_0, \tilde{s}) \in X \setminus X_s$ and $P_o(s) = P_o(\tilde{s})$.* □

Example 2.3 *Let G be the automaton represented in Figure 2.2, $\Sigma = \{a, b, c\}$, $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{c\}$ and $X_s = \{3\}$. We can say that G is current-state opaque because sequence "acb", which reaches secret state 3, has the same projection as sequence "cab", that reaches state 7, i.e.,$f(0, acb) = 3 = X_s$, $f(0, cab) = 7 \in X \setminus X_s$ and $P_o(acb) = P_o(cab) = ab$. Note that, if we set $\Sigma_o = \{a, c\}$, $\Sigma_{uo} = \{b\}$ and $X_s = \{4\}$, the system is no longer current-state opaque, since there does not exist a sequence $t \in L : f(x_0, t) \in X \setminus X_s$, whose projection is $P_o(t) = P_o(acba) = aca$.* □

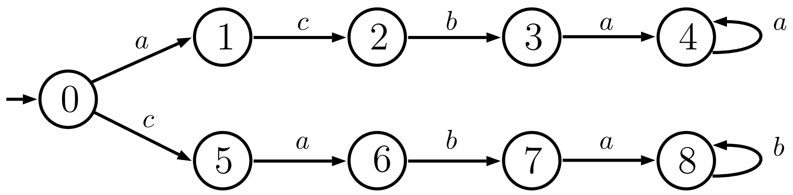


Figure 2.2: System related to Examples 2.3, 2.5 and 2.6.

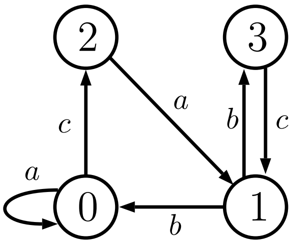


Figure 2.3: System related to Example 2.4, taken from [32].

We present now the notion of initial-state opacity, that was initially proposed in the context of Petri nets by BRYANS *et al.* [22] and brought to the finite-state automaton formalism by SABOORI and HADJICOSTIS [32]. A system is initial-state opaque when *"the intruder is never sure whether the initial state of the system was a secret state or not"*, being formally defined as follows.

Definition 2.9 (Initial-State Opacity) *Given a system modeled by* $G = (X, \Sigma, f, \Gamma, X_0)$*, projection* $P_o : \Sigma^* \to \Sigma_o^*$*, and a set of secret initial states* $X_{0,s} \subseteq X_0$*,* G *is initial-state opaque with respect to* Σ_o *and* $X_{0,s}$ *if* $\forall x_0 \in X_{0,s}$ *and for all* $s \in L$*,* $f(x_0, s)!$*, there exists* $y_0 \in X_0 \setminus X_{0,s}$ *and* $t \in L$*,* $f(y_0, t)!$ *such that* $P_o(s) = P_o(t)$*.* □

Example 2.4 *Let* G *be the automaton represented in Figure 2.3, taken from SABOORI and HADJICOSTIS [32], where* $\Sigma = \{a, b, c\}$*,* $\Sigma_o = \{a, b\}$*,* $\Sigma_{uo} = \{c\}$*,* $X_0 = X$ *and* $X_{0,s} = \{2\}$*. In this example, the intruder wants to discover if, among all the possible initial states, the system has started from a secret state. Indeed the system modeled by automaton* G *is initial-state opaque, since for every sequence* s *starting from state* 2*, there exists another sequence* $t = cs$ *starting from state* 0 *with the same observation, i.e.,* $P_o(s) = P_o(t)$*. On the other hand, if the set of secret initial states is* $X_{0,s} = \{0\}$*, then the system is no longer initial-state opaque, since when the sequence "aa" is observed, the intruder will track down the initial state* 0*.* □

The next notion we recall here is K-step opacity, proposed by SABOORI and HADJICOSTIS [31]. It is a more general property that embeds the secrecy of states

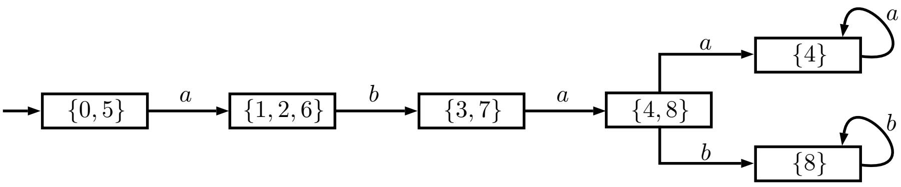


Figure 2.4: Observer of the automaton depicted in Figure 2.2.

from the last K steps executed by the system until the current moment. In words, when *"the intruder is never sure that the system is in a secret state or has visited one in the last K steps"*, the system is said to be K-step opaque. Its formal definition is as follows.

Definition 2.10 (K-Step Opacity) *Given a system modeled by $G = (X, \Sigma, f, \Gamma, x_0)$, projection $P_o : \Sigma^* \to \Sigma_o^*$, a set of secret states $X_s \subseteq X$, and an integer $K \geq 0 \in \mathbb{N}$, we say that G is K-step opaque with respect to Σ_o, X_s and K if for all $s \in L$, and for all $s' \in \overline{s}$ satisfying $|P_o(s)| - |P_o(s')| \leq K$ and $f(x_0, s') \in X_s$, there exists $t \in L$ and $t' \in \overline{t}$ such that $f(x_0, t') \in X \setminus X_s$, $P_o(s) = P_o(t)$ and $P_o(s') = P_o(t')$.* □

Example 2.5 *Assume that G is the automaton represented in Figure 2.2, and let $\Sigma = \{a, b, c\}$, $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{c\}$ and $X_s = \{3\}$. Then, we can say that G is 1-step opaque but not 2. In order to show that fact, let us consider the observer automaton $Obs(G, \Sigma_o)$ depicted in Figure 2.4. Note that, when the intruder observes sequence "ab", it estimates states $\{3, 7\}$. Then, the only possible next observable sequence is "aba", and now the intruder knows that the system is currently in $\{4, 8\}$ and was in $\{3, 7\}$ one step back. Thus, the system is 1-step opaque. However, when the intruder observes sequence "abaa", it becomes sure that G is currently in state 4 and, by considering automaton G shown in Figure 2.2, we can see that the system was also in state 4 one step back and in state 3 two steps back, which is a secret state. Thus the system is not 2-step opaque.* □

Note that CSO can be understood as 0-Step opacity, since the only concern of CSO is about the intruder being sure that the current state is a secret one, *i.e.*, it takes into account zero past steps.

The last state-based opacity notion we revisit in this work is the so-called infinite-step opacity, which has been introduced by SABOORI and HADJICOSTIS [33], being an extension of K-step opacity. In this regard, a system is said to be infinite-step opaque if *"the intruder is never sure whether the system has ever been in a secret state or not"*. Its formal definition is as follows.

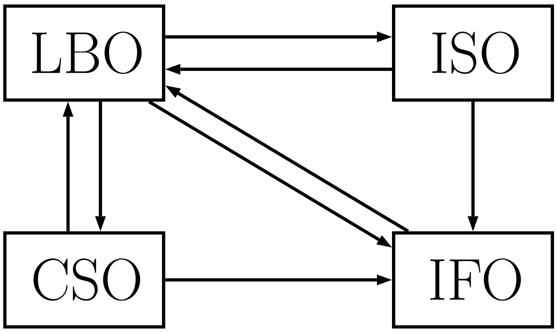


Figure 2.5: Transformations between CSO, IFO, and LBO [24].

Definition 2.11 (Infinite-Step Opacity) *Given a system modeled by* $G = (X, \Sigma, f, \Gamma, x_0)$*, projection* $P_o : \Sigma^* \rightarrow \Sigma_o^*$*, and a set of secret states* $X_s \subseteq X$*,* G *is infinite-step opaque if for all* $s \in L$ *and for all* $s' \in \overline{s}$ *satisfying* $f(x_0, s') \in X_s$*, there exists* $t \in L$ *and* $t' \in \overline{t}$ *such that* $f(x_0, t') \in X \setminus X_s$*,* $P_o(s) = P_o(t)$ *and* $P_o(s') = P_o(t')$*.* □

Example 2.6 *We revisit the automaton* G *depicted in Figure 2.2, where* $\Sigma = \{a, b, c\}$*,* $\Sigma_o = \{a, b\}$*,* $\Sigma_{uo} = \{c\}$ *and* $X_s = \{3\}$*. Since the system is not* 2*-step opaque, as explained in Example 2.5, it cannot be infinite-step opaque. However, if we replace the self-loop in state* 8 *with another self-loop labeled with event "a", then the system becomes infinite-step opaque, since now the intruder can not ascertain if the system has ever been solely in the secret state* 3*.* □

It is worth noting that WU and LAFORTUNE [24] proposed a new opacity notion, namely initial-and-final-state opacity (IFO), which is a generalization of both CSO and ISO, where the secret behavior of the system is modeled as pairs composed of an initial and a marked state. Moreover, WU and LAFORTUNE [24] provided transformations among CSO, IFO, and LBO, and, for prefix-closed languages, also between LBO and ISO. The computational complexity for performing any of these transformations is of polynomial time [24]. Figure 2.5 presents a diagram with these transformations between the opacity notions.

We emphasize that our main objective on this topic is to approach the problem of current-state opacity enforcement. We recall that, in this notion, the secret behavior is modeled as states, which cannot be accurately estimated by the intruder at the current time instant. Therefore, in order for a system to be current-state opaque, all sequences that reach a secret state from the initial state must, from the point of view of the intruder, be indistinguishable from some other sequence that reaches a non-secret state from the initial state.

In this regard, an easy and intuitive way to verify if a system G is CSO is to build its observer automaton G_{obs}, since its states can be understood as current

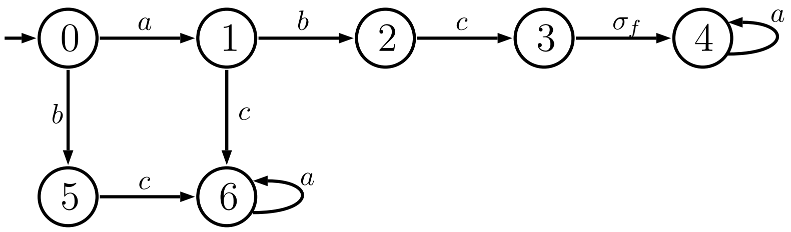


Figure 2.6: Automaton G.

state estimates, and verify if there is no state of G_{obs} formed with secret states solely. Thus, the set of secret states of G_{obs} is computed by $X_{s,obs} = X_{obs} \cap 2^{X_s}$, since these states denote estimations composed of secret states only. The existence of some state in $X_{s,obs}$ means that the intruder is capable of inferring, at some time, that the current state of the system certainly is a secret one. We then say that the system is CSO if and only if $X_{s,obs} = \emptyset$, which means that the intruder is not able to estimate a set of secret states only. In this case, the problem of CSO verification can be rephrased as "the intruder must never estimate a set of secret states only".

2.2.2 Predictability

Predictability, also referred to as prognosability in the literature, is a property of DESs that ensures that some specific events, usually the fault event, can always be foreseen before its actual occurrence.

In order to present the formal definition of predictability, let $\Psi(\sigma)$ denote the set composed of all sequences in $L = \mathcal{L}(G)$ whose last event is σ, being defined as $\Psi(\sigma) = \{s\sigma \in L : s \in \Sigma^*, \sigma \in \Sigma\}$. This function can be extended to an event set $\widetilde{\Sigma} \subseteq \Sigma$ as follows: $\Psi(\widetilde{\Sigma}) = \bigcup_{\widetilde{\sigma} \in \widetilde{\Sigma}} \Psi(\widetilde{\sigma})$. Given a sequence $s \in L$ and a subset $\Sigma_a \subseteq \Sigma$, we denote $\Sigma_a \in s$ to mean that at least one event in Σ_a is necessary to form sequence s, *i.e.*, $\exists \sigma_a \in \Sigma_a, \overline{s} \cap \Psi(\sigma_a) \neq \emptyset$. For example, consider sequence $s = bca \in \mathcal{L}(G)$, where G is the automaton depicted in Figure 2.6. Clearly $\overline{s} = \{\varepsilon, b, bc, bca\}$, and so, for $\Sigma' = \{b, \sigma_f\}$, we have that $\Sigma' \in s$, since, although $\Psi(\sigma_f) = \{abc\sigma_f\}$ and $\overline{s} \cap \Psi(\sigma_f) = \emptyset$, $\Psi(b) = \{b, ab\}$ and $\overline{s} \cap \Psi(b) = b \neq \emptyset$.

The formal definition of predictability is as follows [20].

Definition 2.12 (Predictability [20]) *A prefix-closed and live language L is predictable with respect to projection $P_o : \Sigma^* \to \Sigma_o^*$ and event $\sigma_p \in \Sigma_p$ if:*

$$\big(\exists z \in \mathbb{N}\big)\big(\forall s \in \Psi(\sigma_p)\big)\big(\exists t \in \overline{s}\big)\big[(\sigma_p \notin t) \wedge \boldsymbol{P}\ \big],$$

where the predictability condition $\boldsymbol{P}$ is:

$$\big(\forall u \in L\big)\big(\forall v \in L/u\big)\big((P_o(u) = P_o(t)) \wedge (\sigma_p \notin u) \wedge (\|v\| \geq z) \Rightarrow (\sigma_p \in v)\big). \quad \square$$

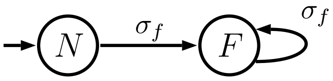


Figure 2.7: Automaton A_ℓ.

Let $\Sigma_f \subseteq \Sigma_{uo}$ be the set composed of fault events, which is assumed to be a singleton for the sake of simplicity and without loss of generality [107], *i.e.*, $\Sigma_f = \{\sigma_f\}$. We will assume here that the fault event σ_f is the event to be predicted, *i.e.*, $\sigma_p = \sigma_f$, and thus, $\Sigma_p = \{\sigma_f\}$. We call s a faulty sequence if $\Sigma_f \in s$; on the other hand, a sequence s such that $\Sigma_f \notin s$ is said to be normal.

Example 2.7 *Consider automaton G depicted in Figure 2.6, whose generated language is L, its event set $\Sigma = \{a, b, c, \sigma_f\}$, and the fault event set $\Sigma_f = \{\sigma_f\}$. Assume that $\Sigma_o = \{a, c\}$ denote the set of observable events. Since, the projection P_o of each prefix of $s_f = abc\sigma_f$ ($P_o(s_f) = ac$) is identical to the projection of some prefix u_1 of the arbitrarily long length normal sequence $u_1v_1 \in aca^*$ ($P_o(u_1v_1) \in aca^*$), language L is not predictable with respect to P_o and Σ_f (we cannot determine an exact moment from which we are certain that the fault will inevitably occur). However, if we consider $\Sigma'_o = \{a, b\}$ as being the set of observable events, it is not difficult to see that L becomes predictable with respect to $P'_o : \Sigma^* \to \Sigma'^*_o$ and Σ_f, since, for prefix $ab \in \overline{s_f}$, there exists no prefix u_2 of the arbitrarily long length normal sequence $u_2v_2 \in (a + b)ca^*$ such that $P'_o(u_2) = P'_o(ab) = ab$, and thus, after sequence ab is observed, any external observer can infer that the fault σ_f will inevitably occur.* □

Fault prediction verification is similar to diagnosability verification [18, 102, 104]. Whereas the latter is usually based on the search, after each fault occurrence, for state estimates where we are certain that the fault has occurred, in the former, we must search for a state estimate prior to the fault behavior such that the fault occurrence is inevitable. Therefore, a system is said to be predictable if, for every fault behavior, there exists a state estimation, prior the fault occurrence, such that we are certain that the fault will inevitably occur.

Inspired by [102], a recent strategy for the verification of fault predictability has been proposed in [82], where the assumptions on language liveness and absence of cycles of states connected by unobservable events have been dropped. In order to build the fault predictor automaton proposed in [82], it is first necessary to build the label automaton $A_\ell = (X_{A_\ell}, \{\sigma_f\}, f_{A_\ell}, \Gamma_{A_\ell}, x_{0,A_\ell}, \emptyset)$, where $X_{A_\ell} = \{N, F\}$, $f_{A_\ell}(N, \sigma_f) = f_{A_\ell}(F, \sigma_f) = F$, $x_{0,A_\ell} = N$. Automaton A_ℓ has been depicted in Figure 2.7. Then, we compute $G_\ell = G \| A_\ell = (X_\ell, \Sigma, f_\ell, \Gamma_\ell, x_{0,\ell}, \emptyset)$ and also its observer observer $G_d = Obs(G_\ell, \Sigma_o) = (X_d, \Sigma_o, f_d, \Gamma_d, x_{0,d}, \emptyset)$, which models the fault predictor automaton.

Notice that the parallel composition $G_\ell = G \| A_\ell$ results in a new automaton where all of its states are labeled either with N, meaning that the fault has not occurred yet, or with F, denoting that the state has been reached after some fault has occurred. According to [18], a diagnoser state is said to be F-certain (resp. normal) if all of its components are F-labeled (resp. N-labeled). If a state has both N and F-labeled components, it is called *uncertain*. We then recall the definition of predictor states, as follows.

Definition 2.13 (Predictor states [82]) *A state $x \in X_d$ of the fault predictor automaton G_d is a predictor state if it is uncertain and either x is the initial state of G_d or there exists $(x', \sigma_o) \in X_d \times \Sigma_o$ such that x' is normal and $f_d(x', \sigma_o) = x$.* □

Notice that predictor states are the first uncertain states of G_d reached from the initial state $x_{0,d}$, which may include $x_{0,d}$ itself, if it is an uncertain state. The set formed with all predictor states of G_d is denoted as X_p. It is not difficult to see that if G has a fault event, then G_d has at least one uncertain state, which implies that $|X_p| \geq 1$.

The next step to perform the fault prediction is to mark all predictor states in G_d, *i.e.*, $X_{m,d} = X_p$, and also all states labeled with N in G_ℓ, *i.e.*, $X_{m,\ell} = \{x \in X_\ell : x \text{ is labeled with } N\}$. Then, we proceed as follows: (i) compute $G_{scc} = G_d \| G_\ell = (X_{scc}, \Sigma, f_{scc}, \Gamma_{scc}, X_{0,scc}, X_{m,scc})$; (ii) set their marked states as being initial states, *i.e.*, $X_{0,scc} = X_{m,scc}$; (iii) unmark all states, *i.e.*, $X_{m,scc} = \emptyset$; (iv) compute $G_{scc,ac} = Ac(G_{scc})$, and; (v) find all non-trivial strongly connected components (SCC[1]) of $G_{scc,ac}$. Notice that the initial states of $G_{scc,ac}$ are such that their first components are predictor states and their second are states labeled with N.

Finally, we recall the following necessary and sufficient conditions for predictability verification based on automaton $G_{scc,ac}$, presented in [82].

Theorem 2.1 (Fault predictability verification [82]) *A language L is predictable with respect to projection P_o and failure event set Σ_f if, and only if, all non-trivial strongly connected components of $G_{scc,ac}$ are composed of states (x_d, x_ℓ) whose second component x_ℓ is F-labeled.* □

According to Theorem 2.1, if all non-trivial SCCs of $G_{scc,ac}$ are formed with states (x_d, x_ℓ) whose second component x_ℓ is F-labeled, then all initial states of $G_{scc,ac}$, which are pairs composed of a predictor state and a normal one, will eventually lead $G_{scc,ac}$ to states where the fault has occurred. This means that all faults will eventually be predicted, and thus, L is predictable. We also recall that no fault

[1] A set of states $B \subseteq 2^X$ forms a nontrivial SCC of G if: (i) for each pair $x, y \in B$, x reaches y and vice versa; (ii) the set B is maximal, i.e., there exists no state z which is not in B but satisfies *(i)*; (iii) if $B = \{x\}$ ($|B| = 1$), then $\exists \sigma \in \Sigma : f(x, \sigma) = x$.

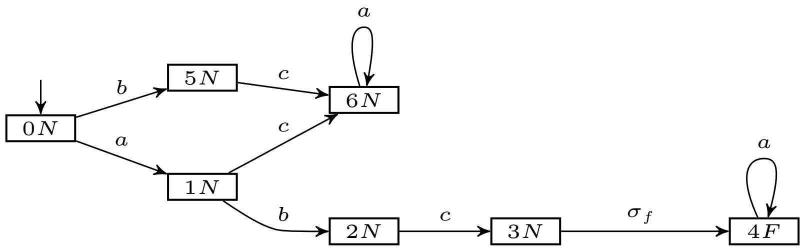


Figure 2.8: Automaton G_ℓ from Example 2.8.

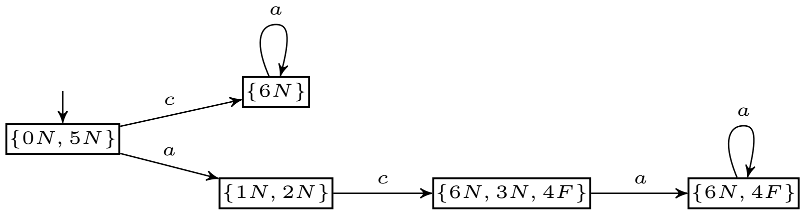


Figure 2.9: Observer automaton $G_d = Obs(G_\ell, \Sigma_o)$ from Example 2.8.

occurrence is missed, since no fault behavior was removed throughout the procedure to compute $G_{scc,ac}$.

Example 2.8 *Let us consider again automaton G depicted in Figure 2.6, where $\Sigma = \{a, b, c, \sigma_f\}$, $\Sigma_f = \{\sigma_f\}$ and $\mathcal{L}(G) = L$. Assume that the set of observable events is $\Sigma_o = \{a, c\}$. We then compute automaton $G_\ell = G \| A_\ell$ and its observer $G_d = Obs(G_\ell, \Sigma_o)$, illustrated in Figures 2.8 and 2.9, respectively. Notice that states $\{0N, 5N\}$, $\{6N\}$ and $\{1N, 2N\}$ of the observer automaton G_d are normal, whereas states $\{3N, 6N, 4F\}$ and $\{6N, 4F\}$ are uncertain. Since $\{3N, 6N, 4F\}$ is the unique uncertain state that is preceded by a normal state, $\{3N, 6N, 4F\}$ is the unique predictor state of G_d. We then mark state $\{3N, 6N, 4F\}$ in G_d and all states labeled with N in G_ℓ, and compute $G_{scc} = G_d \| G_\ell$, which is depicted in Figure 2.10 and whose set of marked states is $X_{m,scc} = \{(\{3N, 6N, 4F\}, 3N), (\{3N, 6N, 4F\}, 6N)\}$. The next steps for the fault predictability verification proposed in [82] are: set $X_{m,scc}$ as initial states of G_{scc}, unmark all of its states and compute $G_{scc,ac} = Ac(G_{scc})$. Notice that automaton $G_{scc,ac}$ has two non-trivial SCCs, both composed of one state only, as follows: $(\{6N, 4F\}, 4F)$ and $(\{6N, 4F\}, 6N)$. The existence of an initial state of $G_{scc,ac}$ (whose first component is a predictor state) that reaches a non-trivial SCC whose last component is labeled with N means that there exists a sequence $s_N = ac$ that reaches the marked state $(\{3N, 6N, 4F\}, 6N)$ in Gscc, with an arbitrarily long-length normal continuation and whose observation is identical to the sequence $s = abc$, which reaches state $(\{3N, 6N, 4F\}, 3N)$, that immediately precedes the fault occurrence, i.e., $P_o(ac) = P_o(abc) = ac$, meaning that the fault cannot be predicted. This result concurs with that of Example 2.7 and is also in accordance with Theorem 2.1, since the existence of a non-trivial SCC whose second component is labeled with N implies that the language L is not predictable with respect to P_o and Σ_f. If the reader carries*

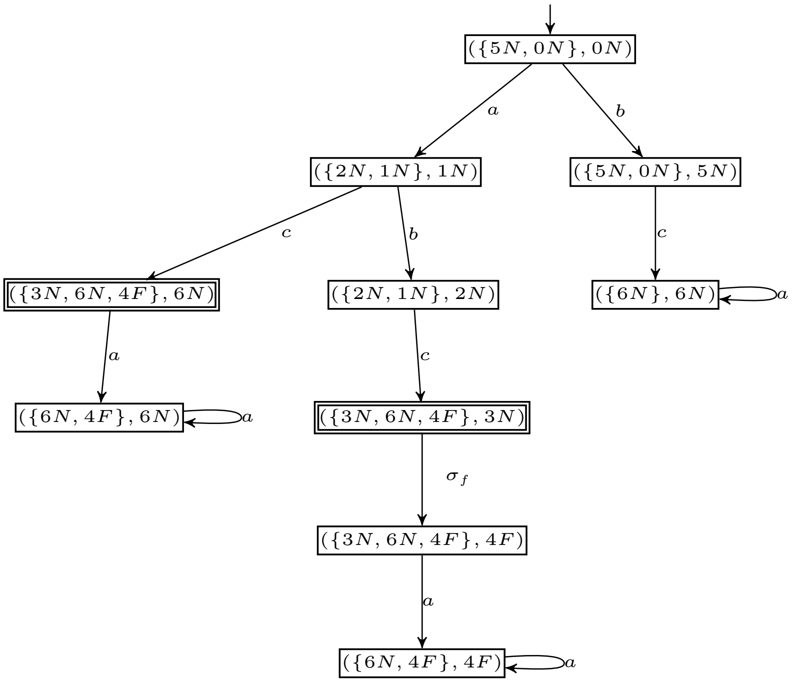


Figure 2.10: Automaton $G_{scc} = G_d || G_\ell$ from Example 2.8.

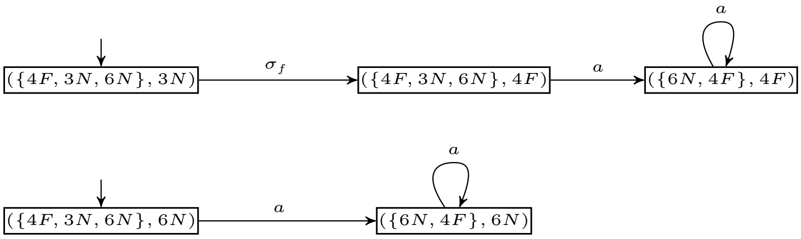


Figure 2.11: Automaton $G_{scc,ac} = Ac(G_{scc})$ from Example 2.8.

out the fault predictability verification for $\Sigma'_o = \{a, b\}$, then the resulting automaton $G_{scc,ac}$ will have only one non-trivial SCC, namely state $(\{4F\}, 4F)$, which implies, according to Theorem 2.1, that L is predictable with respect to $P'_o : \Sigma^* \rightarrow \Sigma'^*_o$ and Σ_f. □

In Chapter 4, we address two new strategies for fault copredictability verification [101]; one of them that is also based on test automaton G_{scc} and improves the results presented in [82].

Chapter 3

Opacity enforcement

In this chapter, we approach the problem of enforcing current-state opacity by shuffling event observations and also deleting some of them when strictly necessary. The idea behind this chapter is that, by manipulating the order of released observation of events, we ensure that the intruder is never capable of estimating a set of secret states only, whereas the harm caused by the opacity enforcement strategy to the estimates of the legitimate receiver can be mitigated by allowing it to know about the Opacity-Enforcer policies.

Throughout this chapter, Section 3.1 characterizes the problem to be solved, the assumptions made on the model of the system, the capacity of the intruder over the system and how the proposed Opacity-Enforcer works; Section 3.2 enlighten us with the concepts behind the strategy for enforcing opacity through shuffle and deletions in event observations whereas Section 3.3 formally presents such a strategy; Section 3.4 presents the algorithms developed to achieve the opacity enforcement strategy; Section 3.5 show us a didactic example where the strategy proposed in this chapter is applied. Section 3.6 presents a procedure to improve the accuracy of the state estimation by the legitimate receiver, when it exists, by assuming that the receiver knows, a priori, the operations that will be performed by the system. Finally, Section 3.7 summarizes all of the contributions made in this chapter. A preliminary version of the results obtained in this chapter are presented in [64] whereas its full version has been presented in [80].

3.1 Problem formulation

The architecture considered in this chapter is shown in Figure 3.1, and is composed of a plant, an Opacity-Enforcer and two players, a legitimate receiver and an intruder. After an event occurrence being read by a sensor, it is transmitted to the Opacity-Enforcer in the same order and immediately after their occurrences in the plant. The observable events, that are released by the Opacity-Enforcer, are transmitted

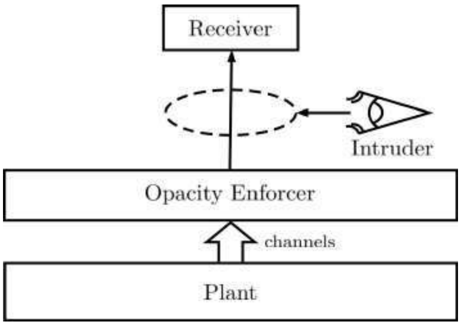


Figure 3.1: The opacity enforcement architecture.

to the legitimate receiver through a non-secure network, being susceptible to leak information to intruders. Although both legitimate receiver and intruder have the same power of observation and have full knowledge of the model of the system, only the legitimate receiver knows how the Opacity-Enforcer behaves to enforce the desired opacity. The secrets of the system are represented as secret states, which the intruder must never discover.

For security reasons, the intruder must never estimate that the system is in a secret state, even when it is not; for example, if the intruder estimates that the system is in a secret state but the system is not, the intruder can still take malicious actions on the system. For this reason we introduce the following design specification.

S1 The intruder can never estimate that the current state of the system is a secret state regardless the current state of the system.

The Opacity-Enforcer we propose here works as follows. When it receives a signal associated with an event occurrence, it makes one of the following decisions: (*i*) it immediately releases the event; (*ii*) it holds the event until one or more events occur; (*iii*) it holds the event and releases a previously held event, not necessarily in the order of its occurrence, which may change the order of event observation; (*iv*) or it deletes the event. In other words, the Opacity-Enforcer either delays the event release by a certain number of steps, where step is understood here as any new arrival of events at the Opacity-Enforcer, or deletes the event forever.

We make the following assumptions on the intruder's capacity:

I1. The intruder has a copy of the automaton that models the system, including events, states, transition functions, and its initial state(s).

I2. The intruder has full access to the events transmitted from the Opacity-Enforcer to the legitimate receiver, *i.e.*, it observes all of the observable events.

I3. The intruder does not know about the existence of the Opacity-Enforcer, and so, it does not know that the information might have been changed.

I4. The intruder always expects to estimate some state inside the model whenever it observes an event.

Notice that, the current-state opacity enforcement strategy proposed in this work is capable of misleading the Intruder to never estimate that the current state of the system is a secret state even when the strategy is publicly known, *i.e.*, assumption **I3** is dropped, as discussed later in Subsection 3.6.

3.2 Opacity enforcement through shuffle and deletions in event observations

In this section, we propose an opacity enforcement strategy that leverages the possibility of either delaying an event release or deleting its observation in order to cause changes in the observed sequences, so as to create new sequences whose projections satisfy the CSO requirement of Definition 2.8. In this regard, let $G_p = (X_p, \Sigma_p, f_p, \Gamma_p, X_{0,p})$ be the automaton that models the behavior of the plant, Σ its observable event set and $X_{s,p}$ the secret states of the plant. Notice that, when we build its observer $G = Obs(G_p, \Sigma) = (X, \Sigma, f, \Gamma, x_0)$, the estimation of some secret states in $X_{s,p}$ may become indistinguishable from the estimation of states in $X_p \backslash X_{s,p}$, and therefore, the opacity enforcement strategy must manipulates the observation of events so as the intruder becomes unable to estimate those states that leak the secret, *i.e.*, the secret states of the observer $X \cap 2^{X_{s,p}}$ (to be denoted as X_s for simplicity).

It is worth remarking that since the strategy for opacity enforcement presented here modifies the sequence of observations outputted by the plant, so as to mislead external agents, it is only necessary to know the observed behavior of the plant, and not its actual behavior, which includes both observable and unobservable event occurrences. Since we are concerned with manipulating the order of event observations to obfuscate the estimation of secret states of the observer X_s, all of the operations and functions hereafter defined will be performed over the language of the observer $L = \mathcal{L}(G)$.

As consequence, the CSO enforcement strategy modifies the order of observed events to create new sequences that mislead the intruder's estimation of the current state of the system so as it becomes unable to estimate any secret state of the observer, which corresponds to the secret states of the plant that are not inherently opaque. To this end, in order to build automaton R_{OE} that models the behavior of the Opacity-Enforcer, we must check if, for every sequence that reaches a secret state and its continuations, it is possible, by shuffling event observations and/or deleting some of their observations, to create a new sequence that can be seen as another possible one whose prefixes and continuations lead only to non-secret states. It is important to take into account which continuations the modified sequence of released events have when performing the opacity enforcement strategy, otherwise the Opacity-Enforcer may release a sequence of events such that all of its continuations lead to secret states and they cannot be modified to sequences that lead the system to a secret-free path of estimation.

The problem of opacity enforcement can be posed as follows [65]: given a system modeled by an automaton $G = (X, \Sigma, f, \Gamma, x_0)$ that is not CSO, obtain another automaton R_{OE} that satisfies the following conditions:

OE1. R_{OE} is CSO;
OE2. $L(G) = L(Obs(R_{OE}, \Sigma))$.

Thus, the CSO enforcement strategy developed in this chapter is obtained by building an automaton R_{OE} from G that satisfies conditions **OE1** and **OE2**. However, we must first verify if a system is current-state enforceable.

To this end, let us define the following operation over the sequences of L.

Definition 3.1 (Sequence permutation) *Let $s = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*$ and $T(s) = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ denote the n-tuple formed from it. The sequence permutation is a mapping $S_p : \Sigma^* \to 2^{\Sigma^*}$ where for each $s = \sigma_1\sigma_2 \ldots \sigma_n$, it associates a set $S_p := \{s_p \in \Sigma^* : s_p$ is a sequence that corresponds to a permutation of $T(s)\}$.* □

Example 3.1 *Let $s = aba$. The corresponding tuple of s is $T(s) = (a, b, a)$. The permutations defined from $T(s)$ form the set $\mathcal{P} = \{(a, b, a), (a, a, b), (b, a, a)\}$ and thus, the sequence permutation of s is $S_p(s) = \{aba, aab, baa\}$.*

In order to model event deletion, we need to augment the language generated by automaton G to include unobservable events. To this end, the first step is to dilate the language L [3]. Let $\Sigma = \Sigma_D \dot{\cup} \Sigma_{\neg D}$ be a partition of Σ, where Σ_D denotes the set of observable events whose observation are allowed to be deleted and $\Sigma_{\neg D}$ denotes the set of observable events whose observation cannot be deleted; Σ_D and $\Sigma_{\neg D}$ will be referred to as deletable and undeletable event sets. We emphasize that the choice of which events are deletable is a design variable. In addition, let $\Sigma_d = \{\sigma_d : \sigma \in \Sigma_D\}$ denote the set of deleted observations, and define $\Sigma_{dil} = \Sigma \cup \Sigma_d$.

Definition 3.2 (Dilation) *The dilation operation is the mapping $Dil : \Sigma^* \to 2^{\Sigma_{dil}^*}$, defined as: $Dil(\varepsilon) = \{\varepsilon\}$; $Dil(\sigma) = \{\sigma\}$, if $\sigma \in \Sigma \setminus \Sigma_D$; $Dil(\sigma) = \{\sigma, \sigma_d\}$, if $\sigma \in \Sigma_D$; and, $Dil(s\sigma) = Dil(s)Dil(\sigma)$, where $s \in \Sigma^*$ and $\sigma \in \Sigma$. Its extension over a language L is performed by applying it to each sequence in L, i.e., $Dil(L) = \bigcup_{s \in L} Dil(s)$.* □

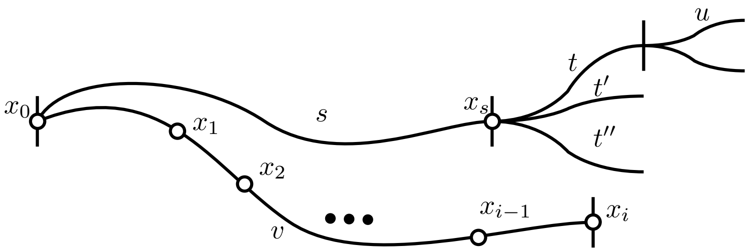


Figure 3.2: Opacity-enforceability.

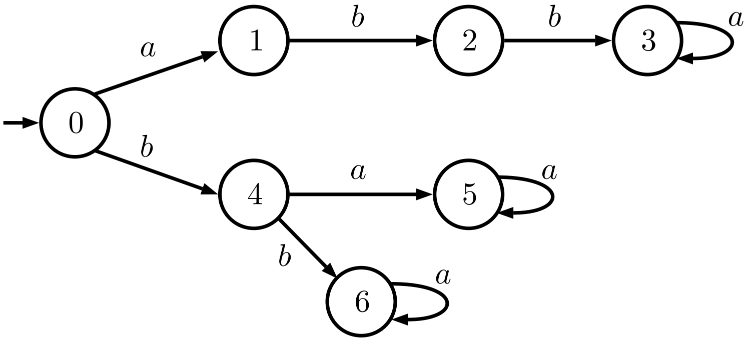


Figure 3.3: Example for opacity-enforceability.

We say that a system is current-state opaque enforceable (CSOE) if, for all sequences s that reach a secret state, and for any of their continuations t, there always exists a sequence u after st, so that there is a shuffling v of stu, with possible event deletions, such that all prefixes w of v lead to non-secret states only, as illustrated in Figure 3.2. Formally, opacity-enforceability is defined as follows.

Definition 3.3 (CSO Enforceability) *A system, whose observable behavior is modeled by an automaton $G = (X, \Sigma, f, \Gamma, x_0)$, is CSOE with respect to Σ_D, X_s and $P_{dil} : \Sigma_{dil}^* \to \Sigma^*$ through changes and deletions in the order of event observations if $(\forall s : f(x_0, s) \in X_s)(\forall t \in L/s)(\exists u \in L/st \wedge \exists v \in P_{dil}(Dil(S_p(stu))) \cap L)[f(x_0, w) \in X \setminus X_s, \forall w \in \overline{v}]$.* □

According to Definition 3.3, given an automaton G that is not CSO but is CSOE, it is possible to build an automaton R_{OE} such that for all sequences $s \in L(G)$ that lead to secret states, we can obtain, by changing the order of event observations together with possible event deletion, at least one sequence whose projection with respect to Σ is in the language generated by G (condition **OE2**), its prefixes never visit secret states, and it satisfies condition **OE1**.

Example 3.2 *In order to illustrate the CSO enforceability concept, let us consider a system whose observed behavior is modeled by the automaton shown in Figure 3.3.*

Assume, initially, that the set of deletable events is $\Sigma_D = \emptyset$, and that the set of secret states is $X_s = \{3\}$. Notice that, $L_s = abba^*$ is the set of all sequences *that reach the secret state, and that for all of the continuations $t \in a^*$ of $s \in L_s$, we may set $u = \varepsilon$ (a finite sequence). Notice that since $P_{dil}(Dil(S_p(stu))) \cap L = P_{dil}(Dil(S_p(abba^*))) \cap L = abba^* + bba^*$, it is clear that $v \in bbaa^*$, and thus, for all $w \in \overline{v}$, $f(0, w) \in \{0, 4, 6\} \subset X \backslash X_s$. Therefore, the system is CSOE through shuffles of events in Σ and deletions of events in Σ_D.*

Assume, now, that $\Sigma_D = \{b\}$ and $X_s = \{3, 6\}$. In this case, $L_s = abba^* + bba^*$. Let us first consider sequences $abba^*$. In this case, we cannot shuffle sequence $abba^*$ so that it becomes like $bbaa^*$, since $f(0, bb) = \{6\} \in X_s$. Notice that $P_{dil}(Dil(S_p(abba^*))) \cap L = abba^* + bba^* + ba^*$, and since the observation of event b can be deleted, we can see that for $s \in abba^*$ and for all $t \in a^*$, there exists a finite $u = \varepsilon$, and so, we may define $v \in baa^*$ such that for all $w \in \overline{v}$, $f(0, w) \in \{0, 4, 5\} \subset X \backslash X_s$. Let us now consider sequences $s' \in bba^*$, which reach secret state 6. Notice that $P_{dil}(Dil(S_p(bba^*))) \cap L = abba^* + bba^* + ba^*$. It is clear that $t' \in a^*$ represent all possible continuations of s', and so, there exists $u' = \varepsilon$ that allow us to define $v' \in ba^*$ such that for all $w' \in \overline{v'}$, $f(0, w') \in \{0, 4, 5\} \subset X \backslash X_s$. Thus, the system is, in this case also, CSOE through shuffles and deletions of events. □

Remark 3.1 *In Example 3.2, when $\Sigma_D = \{b\}$, $X_s = \{3, 6\}$ and assuming sequence $s = bbaa$ is generated by the system, state 6 is reached. in this case, an immediate solution would be to delete two occurrences of event b after bba is generated (leading to the estimation of state 1). Thus, when the last event a is generated, sequence aa will be transmitted to all external observers. However, such a sequence is not in the behavior of the system, and so, would reveal to the intruder that some manipulation of event transmissions has been carried out; therefore, thus violating Assumptions **I3** and **I4**. Thus, the challenge here is that the Opacity-Enforcer modifies the observation of events wisely, by taking into account not only past event occurrences but also future ones, in some way that these modifications never compromise future estimations.* □

In order to define an opacity enforcement strategy, the occurrence of an event in the system must be distinguished from its observation by the intruder. To this end, let Σ_r be a copy of Σ with all of its events labeled by a subscript "r", *i.e.*, $\Sigma_r = \{\sigma_r : \sigma \in \Sigma\}$. We recall that Σ_d, the set of deleted observations, is itself a copy of Σ_D, the set of deletable events. We define the following functions.

Definition 3.4 (Release and Deletion Functions)

- *The release function is the mapping $\varphi_r : \Sigma^* \rightarrow \Sigma_r^*$, where $\varphi_r(\varepsilon) = \varepsilon$, $\varphi_r(\sigma) = \sigma_r$ and $\varphi_r(s\sigma) = \varphi_r(s)\varphi_r(\sigma)$, where $\sigma \in \Sigma$ and $s \in \Sigma^*$.*
- *The inverse release function is the mapping $\varphi_r^{-1} : \Sigma_r^* \rightarrow \Sigma^*$ where $\varphi_r^{-1}(\varepsilon) = \varepsilon$, $\varphi_r^{-1}(\sigma_r) = \sigma$ and $\varphi_r^{-1}(s_r\sigma_r) = \varphi_r^{-1}(s_r)\varphi_r^{-1}(\sigma_r)$;*

- *The deletion function is the mapping $\varphi_d : \Sigma_D^* \to \Sigma_d^*$, where $\varphi_d(\varepsilon) = \varepsilon$, $\varphi_d(\sigma) = \sigma_d$, and $\varphi_d(s\sigma) = \varphi_d(s)\varphi_d(\sigma)$.* □

3.3 Opacity-enforcement strategy

According to Section 3.1, for every sequence that has occurred in the plant, the Opacity-Enforcer must take one of the following actions after each event arrival:

OEA1. Release immediately its observation, or hold the event but releasing the observation of another event that has been held (not necessarily in the same order it occurred);

OEA2. Delete its observation (if the event is deletable), or hold it and delete the observation of another deletable event that had been held;

OEA3. Take no action, *i.e.*, hold the event that has arrived without releasing/deleting any observation at all and then wait for the arrival of a new event.

In order to implement this policy, we propose the following procedure: all sequences s generated by the system are sent to an augmentation function, which enlarges those sequences with information of possible event releases and deletions, such that, when the released information reaches the intruder, it will be misled to never estimate a set of secret states only. With that in mind, we make the following definitions:

- Augmented event set: $\Sigma_a = \Sigma \cup \Sigma_r \cup \Sigma_d$.
- Function $\mathcal{N} : \Sigma_a^* \times \Sigma_a \to \mathbb{N}$, which, for a pair (s, σ), $\mathcal{N}(s, \sigma)$ returns the number of occurrences of event σ in sequence s.

Notice that Σ_a denotes the event set of the automaton that models the behavior of the Opacity-Enforcer and allows the CSO enforcement strategy through event observation shuffling/deletions, since the event set Σ_a is composed of: (i) the events generated by the plants Σ; (ii) the release of their observations Σ_r, and; (iii) the deletions of their observations Σ_d, if it is an deletable event. For example, sequence $s_a = abb_ra_rcc_da \in \Sigma_a$ denotes that event a and b have been generated by the plant in this order, but event b had its observation released before that of event a (b_ra_r), then event c has been generated by the plant, its observation has been deleted (c_d) and another event a has been generated by the plant in the sequel. Since event a occurred two times in s_a, we have that $\mathcal{N}\big(s_a, a\big) = 2$.

Since both release and deletion of observations are executed after their occurrences, the following constraint must be imposed:

$$\mathcal{N}\big(s_a,\sigma\big) \geq \mathcal{N}\big(s_a,\varphi_r(\sigma)\big) + \mathcal{N}\big(s_a,\varphi_d(\sigma)\big). \tag{3.1}$$

where $s_a \in \Sigma_a^*$ and $\sigma \in \Sigma$. According to Inequality (3.1), the number of occurrences of an event $\sigma \in \Sigma$ in a sequence $s_a \in \Sigma_a^*$ must be greater than or equal to the sum of the observation releases and deletions of σ. Thus, for $s_a \in \Sigma_a^*$, the set of allowed events to be released or deleted with respect to an augmented sequence can be defined as:

$$\Sigma_{R\vee D}(s_a) = \big\{\sigma \in s_a : \big(\sigma \in \Sigma\big) \wedge \big(\mathcal{N}(s_a,\sigma) > \mathcal{N}(s_a,\varphi_r(\sigma)) + \mathcal{N}(s_a,\varphi_d(\sigma))\big)\big\}. \tag{3.2}$$

In order to make the reading easier, we will define the following projections to be used throughout the text:

1) $P_{dil}: \ \Sigma_{dil}^* \rightarrow \Sigma^*$;
2) $P_a: \ \Sigma_a^* \rightarrow \Sigma^*$;
3) $P_r: \ \Sigma_a^* \rightarrow \Sigma_r^*$;
4) $P_d: \ \Sigma_a^* \rightarrow \Sigma_d^*$.

We now formally define the language augmentation function.

Definition 3.5 (Language augmentation function) *The language augmentation function $\mathcal{A} : \Sigma_a^* \rightarrow 2^{\Sigma_r \cup \Sigma_d}$ of a system modeled by automaton G with respect to secret states X_s, and event sets Σ, Σ_r and Σ_d is defined as:*

$$\mathcal{A}(s_a) = \mathcal{A}_r(s_a) \cup \mathcal{A}_d(s_a) \tag{3.3}$$

where:

- $\mathcal{A}_r(s_a) = \Big\{\sigma_r \in \varphi_r\big(\Sigma_{R\vee D}(s_a)\big) : f\Big(x_0, \varphi_r^{-1}\big(P_r(s_a\sigma_r)\big)\Big) \in X \setminus X_s\Big\}$;
- $\mathcal{A}_d(s_a) = \varphi_d\big(\Sigma_{R\vee D}(s_a) \cap \Sigma_D\big)$. □

The language augmentation function works as follows: for a given augmented sequence $s_a \in \Sigma_a^*$, $\mathcal{A}(s_a)$ returns a set of events composed of all allowed event releases σ_r and event deletions σ_d. Notice that, $\mathcal{A}_r$ ensures that all released observations make the intruder estimate only non-secret states of the observer, and, according to the definition of $\mathcal{A}_d$, every event of $\Sigma_{R\vee D}$ that is allowed to be deleted can be deleted at any time. If there is no allowed observation releases or deletions for a given augmented sequence s_a, then $\mathcal{A}(s_a) = \emptyset$, meaning that no action is to be taken by the Opacity-Enforcer; so, it waits for the arrival of a new event generated by the system.

Example 3.3 *Let us consider a system whose observed behavior is modeled by automaton G shown in Figure 3.3, and assume that $\Sigma = \{a, b\}$, $\Sigma_D = \{b\}$ and $X_s = \{3, 6\}$. Let us consider the sequence $s = abb_rb \in \Sigma_a^*$, which means that ab has happened in the system, b_r was released and event b has just occurred. According to Definition 3.5, we have $\mathcal{A}(abb_rb) = \{a_r,\ b_d\}$, since: (i) $\Sigma_{R \vee D}(abb_rb) = \{a, b\}$, (ii) $f\big(x_0, \varphi_r^{-1}(b_ra_r)\big) = 5$ and $5 \in X \setminus X_s$, and (iii) $b_d \in \varphi_d\big(\Sigma_{R \vee D}(abb_rb) \cap \Sigma_D\big)$. Notice that $b_r \notin A(abb_rb)$, since the release of event b_r would lead to the estimation of the secret state* 6. □

In order to build a language L_a such that $P_a(L_a) = L$ and $\forall s_a \in L_a$, $f(x_0, \varphi_r^{-1}(P_r(s_a))) \in X \backslash X_s$, we will apply the language augmentation function to the language L generated by the system, so as whenever an external observer receives these released events, it always estimates non secret states of the model.

Language L_a is formally defined recursively as follows.

Definition 3.6 (Augmented language) *Given language $L \in \Sigma^*$ and an augmentation function $\mathcal{A} : \Sigma_a^* \to 2^{\Sigma_r \cup \Sigma_d}$, the augmented language $L_a \in \Sigma_a^*$ is recursively defined as follows.*

1) $\varepsilon \in L_a$;

2) $(s_a \in L_a) \wedge \big[\big((\sigma \in \Sigma) \wedge (P_a(s_a\sigma) \in L)\big) \vee \big(\sigma \in \mathcal{A}(s_a)\big)\big] \leftrightarrow s_a\sigma \in L_a$. □

It is clear, according to Definition 3.6, that at least one new sequence $(s_a\sigma)$ is added to the augmented language every time an observable event σ occurs in G. In addition, in order for the Opacity-Enforcer to release (resp. delete) an event observation $\sigma_r \in \Sigma_r$ (resp. $\sigma_d \in \Sigma_d$) after s_a, then $s_a\sigma_r$ (resp. $s_a\sigma_d$) must also be added to L_a. It is worth noting that, by construction, $P_a(L_a) = L$. In addition, if at a certain point the Opacity-Enforcer is no longer able to release any new observation, then only sequences $s_a\sigma$ must be added to L_a, after σ occurs in the system.

However, as the next example shows, for a given regular language L, the corresponding augmented language L_a may have the following problems: *(i)* it can be non regular, which prevents the implementation of the Opacity-Enforcer by using a finite state automaton; *(ii)* it may happen that for some sequence $s_a \in L_a$ and all of its continuations $s'_a \in L_a$ with events $\sigma \in \Sigma$, both $\mathcal{A}(s_a) = \emptyset$ and $\mathcal{A}(s'_a) = \emptyset$.

Example 3.4 *Let us revisit Example 3.3. According to Definition 3.6, we set, initially, $L_a = \{\varepsilon\}$.*

- *For $s_a = \varepsilon$, we have that $\{a, b\} \subset L_a$, since $\{P_a(a), P_a(b)\} \subset L$. Thus, sequences a and b are added to L_a, which becomes $L_a = \{\varepsilon, a, b\}$.*
- *For $s'_a = a$, we have that $s'_{a_1} = ab \in L_a$, since $P_a(s'_{a_1}) \in L$, and $s'_{a_2} = aa_r \in L_a$, since $a_r \in \mathcal{A}(s'_a)$. Thus, sequences s'_{a_1} and s'_{a_2} are added to L_a, becoming $L_a = \{\varepsilon, a, b, ab, aa_r\}$.*

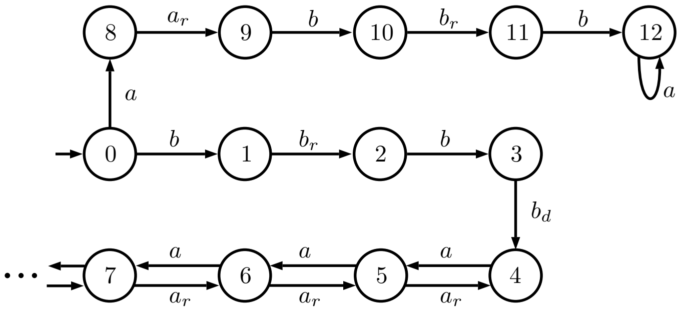


Figure 3.4: Part of the automaton that generates non regular language L_a.

- *For $s''_a = b$, we have that both $s''_{a_1} = bb$ and $s''_{a_2} = ba$ are in L_a, since $P_a(s''_{a_1})$, $P_a(s''_{a_2}) \in L$, and also that both sequences $s''_{a_3} = bb_r$ and $s''_{a_4} = bb_d$ must be in L_a, since $b_r, b_d \in \mathcal{A}(s''_a)$. Therefore, sequences s''_{a_1}, s''_{a_2}, s''_{a_3} and s''_{a_4} are added to L_a, which becomes $L_a = \{\varepsilon, a, b, ab, aa_r, bb, ba, bb_r, bb_d\}$.*

Notice that, if we continue this process, the resulting augmented language L_a will be non regular, since, as we can see in Figure 3.4, after sequence $s_a = bb_rbb_da \in L_a$, either event a occurs again or the observation of a is released (event a_r), and this pattern continues indefinitely for all sequences $bb_rbb_daa^m$, $m \in \mathbb{N}$, which shows that the automaton that generates L_a requires an infinite number of states. In addition, for sequence $s_a = aa_rbb_rb$, we have that $\mathcal{A}(s_a) = \emptyset$, since b_r cannot be released, otherwise, it would make the intruder estimate a secret state. However, sequence s_a can still be augmented with event occurrences in the plant (event a, in this case), since sequences $s_a = aa_rbb_rba^m$, $m \in \mathbb{N}$, where $\mathcal{A}(s_a) = \emptyset$, are such that $P_a(s_aa) \in L$, and so, $s_a \in L_a$, as depicted in Figure 3.4. Notice that, in this case, events a and b are being held indefinitely after sequence aa_rbb_rb is executed. □

In order to avoid the problems arising from the definition of L_a, we will introduce a restriction on the maximum number of occurrences of future events in the plant for which an event observation can be held. To this end, let us define the following step delay bound set:

$$SD(k) = \{(\sigma_1, k_{\sigma_1}), (\sigma_2, k_{\sigma_2}), \ldots, (\sigma_n, k_{\sigma_n})\}. \tag{3.4}$$

where $k = [k_{\sigma_1}, k_{\sigma_2}, \ldots, k_{\sigma_n}]$, $k_{\sigma_i} \in \mathbb{N}$, $i = 1, 2, \ldots, n$, where $n = |\Sigma|$, is a vector whose i-th component represents the maximum number of steps the observation release of event σ_i can be delayed. Since event observation releases/deletions do not delay the observation of other events, the step delays are accounted only when plant events $\sigma \in \Sigma$ occur. It is worth mentioning that the choice of the event step delay bound is a project variable, where the greater each k_{σ_i} is, the easier is to enforce CSO, but, on the other hand, the more inaccurate the estimates of the legitimate receiver become.

Let $pre(s_a, \sigma^{i_\sigma})$, $i_\sigma = 1, \ldots, \mathcal{N}(s_a, \sigma)$, denote the prefix of an augmented sequence $s_a \in \Sigma_a^*$ whose last event is the i_σ-th occurrence of $\sigma \in \Sigma$, and let us define $pre(s_a, \sigma_\ell^{i_\sigma})$, where $\ell \in \{r, d\}$, as the prefix of s_a whose last event is the i_σ-th action (observation release, σ_r, or deletion, σ_d) of the Opacity Enforcer over the observation of event σ when $i_\sigma \leq \mathcal{N}(s_a, \sigma_r) + \mathcal{N}(s_a, \sigma_d)$, or; $pre(s_a, \sigma_\ell^{i_\sigma}) = s_a$, otherwise. We state the following result.

Fact 3.1 *An augmented sequence $s_a \in \Sigma_a^*$ satisfies a step delay bound $SD(k)$, i.e., $s_a \vDash SD(k)$*[1]*, if, and only if, $\forall \sigma \in \Sigma$, $\|P_a(pre(s_a, \sigma_\ell^{i_\sigma}))\| - \|P_a(pre(s_a, \sigma^{i_\sigma}))\| \leq k_\sigma$, $i_\sigma = 1, \ \ldots, \ \mathcal{N}(s_a, \sigma)$, and $\ell \in \{r, d\}$.* □

In words, an augmented sequence satisfies the step delay bound if, and only if, for all all events $\sigma \in \Sigma$, the number of plant events between the i-th occurrence of event σ and its observation release/deletion is less than its step delay bound. For example, let $\Sigma = \{a, b\}$ and $SD(k) = \{(a, 1), (b, 0)\}$. Let us consider the following sequences:

(i) $s_{a,1} = abb_r a$. In this case, $i_a = 1, \ldots, \mathcal{N}(s_a, a) = 1, 2$. Notice that $s_{a,1} \nvDash SD(k)$, since $pre(s_{a,1}, a_\ell^1) = s_{a,1} = abb_r a$ and $pre(s_{a,1}, a^1) = a$, which implies that $\|P_a(pre(s_{a,1}, a_\ell^1))\| - \|P_a(pre(s_{a,1}, a^1))\| = \|P_a(abb_r a)\| - \|P_a(a)\| = \|aba\| - \|a\| = 2 > k_a = 1$.

(ii) $s_{a,2} = aba_r aaba_r b_r$. In this case, $i_a = 1, 2, 3$ and $\|P_a(pre(s_{a,2}, a_\ell^1))\| - \|P_a(pre(s_{a,2}, a^1))\| = \|P_a(aba_r)\| - \|P_a(a)\| = \|ab\| - \|a\| = 1 \leq k_a = 1$, but $\|P_a(pre(s_{a,2}, a_\ell^2))\| - \|P_a(pre(s_{a,2}, a^2))\| = \|P_a(aba_r aaba_r)\| - \|P_a(aba_r a)\| = \|abaab\| - \|aba\| = 2 > k_a = 1$; therefore, $s_{a,2} \nvDash SD(k)$.

It is worth remarking that if some sequence $s_a \in L_a$ does not satisfy $SD(k)$, then all of its continuations in L_a do not satisfy $SD(k)$ either, *i.e.*, $s_a \nvDash SD(k) \rightarrow (s_a t_a \nvDash SD(k), \forall t_a \in L_a/s_a))$. This implies that, given an augmented language L_a and a step delay bound $SD(k)$, the bounded delay augmented language $L_a^{SD} \subseteq L_a$ is such that $L_a^{SD} = \{s_a \in L_a : s_a \vDash SD(k)\}$. With that in mind, we recursively define L_a^{SD} as follows:

Definition 3.7 (Bounded delay augmented language) *Given language $L \in \Sigma^*$, an augmentation function $\mathcal{A} : \Sigma_a^* \rightarrow 2^{\Sigma_r \cup \Sigma_d}$ and a step delay bound $SD(k)$, the bounded delay augmented language $L_a^{SD} \in \Sigma_a^*$ is recursively defined as follows:*

(i) $\varepsilon \in L_a^{SD}$;

(ii) $(s_a \in L_a^{SD}) \wedge \big\{\big[(\sigma \in \Sigma) \wedge (P_a(s_a\sigma) \in L) \wedge (s_a\sigma \vDash SD(k))\big] \vee \big[\sigma \in \mathcal{A}(s_a)\big]\big\} \leftrightarrow s_a\sigma \in L_a^{SD}$.

[1] We use the symbol $\vDash$ as an abuse of notation, where $a \vDash b$ means that clause "a" literally satisfies condition "b".

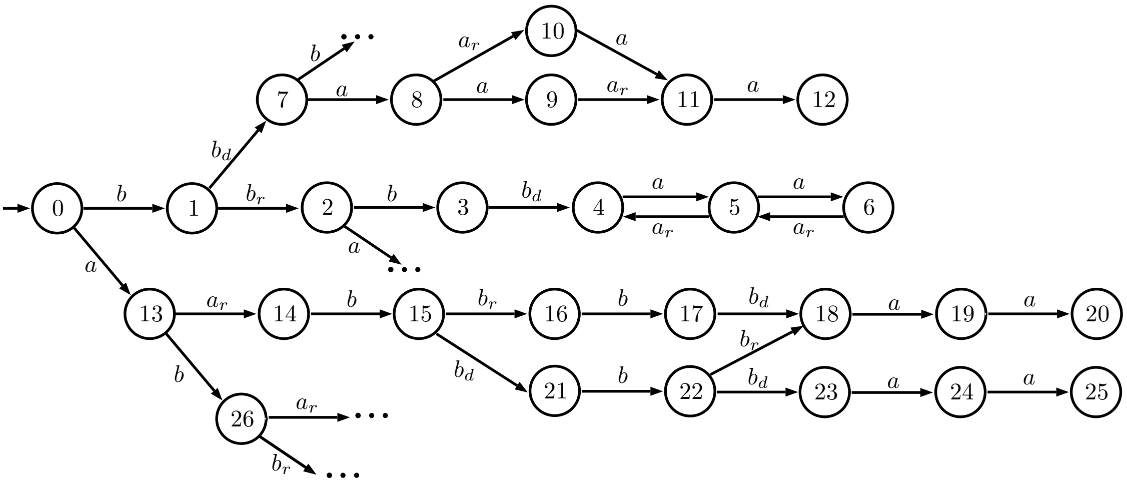


Figure 3.5: Part of the automaton that generates a regular language L_a^{SD} bounded by $SD(k)$.

The construction of the bounded delay augmented language L_a^{SD} is similar to that of the augmented language L_a. We start with the empty sequence ε and then, for each sequence s_a that is already in L_a^{SD}, we add $s_a\sigma$ into L_a^{SD} if one of the following two conditions holds true: (i) σ is an event generated by the plant ($\sigma \in \Sigma$), it can occur in the plant after $P_a(s_a)$, *i.e.*, $P_a(s_a\sigma) \in L$, and its occurrence, considering the events being hold in s_a, does not violate the step delay bound ($s_a\sigma \vDash SD(k)$), or; (ii) event σ denotes an observation release or deletion according to the language augmentation function, *i.e.*, $\sigma \in \mathcal{A}(s_a)$.

Example 3.5 *Consider again the automaton shown in Figure 3.3, where $\Sigma = \{a, b\}$, $\Sigma_D = \{b\}$ and $X_s = \{3,\ 6\}$, and, assume that $SD(k) = \{(a,\ 1), (b,\ 0)\}$. Thus, whenever event b occurs, its observation must be either released or deleted immediately, whereas event a can have its observation released either immediately or in one subsequent event occurrence, but not after that. We set, initially, $L_a^{SD} = \{\varepsilon\}$.*

- *For $s_a = \varepsilon$, we have that $\{a, b\} \subset L_a^{SD}$, since $\{P_a(a), P_a(b)\} \subset L$ and $a, b \vDash SD(k)$. Thus, sequences a and b are added to L_a^{SD}, which becomes $L_a^{SD} = \{\varepsilon, a, b\}$.*

- *For $s'_a = a$, we have that $s'_{a_1} = ab \in L_a^{SD}$, since $P_a(s'_{a_1}) \in L$ and $s'_{a_1} \vDash SD(k)$, and $s'_{a_2} = aa_r \in L_a^{SD}$, since $a_r \in \mathcal{A}(s'_a)$ and $s'_{a_2} \vDash SD(k)$. Thus, sequences s'_{a_1} and s'_{a_2} are added to L_a^{SD}, which becomes $L_a^{SD} = \{\varepsilon, a, b, ab, aa_r\}$.*

- *For $s''_a = b$, we have that both $s''_{a_1} = bb_r$ and $s''_{a_2} = bb_d$ are in L_a^{SD}, since $b_r, b_d \in \mathcal{A}(s''_a)$. Note that, sequences $s''_{a_3} = bb$ and $s''_{a_4} = ba$ cannot be in L_a^{SD}, since, even though $P_a(s''_{a_3}), P_a(s''_{a_4}) \in L$, both sequences, s''_{a_3} and s''_{a_4}, imply that the observation of event b is being held for one step, which violates $SD(k)$*

($k_b = 0$), i.e., $s''_{a3}, s''_{a4} \not\models SD(k)$. Therefore, only sequences s''_{a1} and s''_{a2} are added to L_a, which becomes $L_a^{SD} = \{\varepsilon, a, b, ab, aa_r, bb_r, bb_d\}$.

The bounded delay augmented language L_a^{SD} is obtained by carrying out this procedure until there exists no new sequence that can be added to L_a^{SD}. Notice that, sequence $s_a = ba \in L_a$ but $s_a \notin L_a^{SD}$, even though $P_a(s_a) \in L$, since it models the case when the first occurrence of event b is being held for one step and such a behavior is not allowed by $SD(k)$, i.e., $\|P_a(pre(s_a, b_\ell^1))\| - \|P_a(pre(s_a, b^1))\| = \|P_a(ba)\| - \|P_a(b)\| = \|ba\| - \|b\| = 1 > k_b = 0$, hence, $s_a \not\models SD(k)$ which implies that $s_a\Sigma_a^* \cap L_a^{SD} = \emptyset$.

Figure 3.5 depicts part of the automaton that generates L_a^{SD}. It is worth noticing that the L_a^{SD} is regular whereas L_a is not. □

It is worth remarking that the bounded delay augmented language L_a^{SD} may still be undesirable, in the sense that, even though all sequences $s_a \in L_a^{SD}$ satisfy the step delay bound SD, one of the following undesirable behaviors may occur:

UB1. The Opacity-Enforcer halts after a sequence $s_a \in L_a^{SD}$, where for some event $\sigma \in s_a$, Inequality (3.1) holds with strict inequality relation ($>$). In this case, the Opacity Enforcer holds indefinitely the observation of non deletable events, since, from the definition of L_a^{SD}, every new observation release would either lead to the estimation of a secret state or to a sequence that is outside the language generated by the system. This is illustrated in Figure 3.5, where, after reaching state 12, no action can be taken by the Opacity Enforcer, since neither another observation of event a can be held nor previous observations of a can be released or deleted;

UB2. The immediate continuations of a sequence $s_a \in L_a^{SD}$ are conflicting, in the sense that the Opacity-Enforcer is allowed not only to take some action (release or deletion) but can also wait for another event occurrence, *i.e.*, the Opacity Enforcer action is not unique: it can execute either **OEA1**/**OEA2** or **OEA3**. This is illustrated in Figure 3.5, where, after reaching state 8, the Opacity-Enforcer cannot decide if it releases the observation a_r or waits for another occurrence of a;

UB3. The immediate continuations of a sequence $s_a \in L_a^{SD}$ suggests that the Opacity-Enforcer may take different actions: observation release (**OEA1**) and deletion (**OEA2**). This is illustrated in Figure 3.5, where, after reaching state 1, the Opacity-Enforcer has to decide between releasing and deleting events observation, b_r and b_d, respectively.

Thus, it is necessary to prune L_a^{SD} to obtain a prefix-closed augmented language L_a^{OE}, which does not have the aforementioned undesired behaviors. This can be achieved by removing from L_a^{SD} all sequences s_a and $s_a t_a$, for every $t_a \in L_a^{SD}/s_a$, in such a way that $P_a(L_a^{OE}) = L$ and all of the remaining sequences $s_a \in L_a^{OE}$ satisfy simultaneously the following Opacity-Enforcer conditions:

OEC1. $(\forall s_a \in L_a^{OE})\big((L_a^{OE}/s_a = \emptyset) \rightarrow (\forall \sigma \in \Sigma, \mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma)) = \mathcal{N}(s_a, \sigma))\big)$;

OEC2. $(\forall s_a \in L_a^{OE})\big((\exists \sigma \in \Sigma : P_a(s_a)\sigma \in L) \rightarrow (\exists t_a \in L_a^{OE}/s_a : P_a(s_a t_a) = P_a(s_a)\sigma)\big)$.

Opacity-Enforcer Condition **OEC1** states that all augmented sequences $s_a \in L_a^{OE}$ that have no continuation $t_a \in \Sigma_a^*$ must have all event observations either released or deleted by the Opacity-Enforcer. Condition **OEC2** states that if an event σ can occur in the system after $P_a(s_a)$, then there must exist at least one continuation t_a of sequence s_a that models this behavior.

It is worth noticing that Condition **OEC1** ensures that no sequence in L_a^{OE} satisfies undesirable behavior **UB1** and Condition **OEC2** ensures that $P_a(L_a^{OE}) = L$. Together, Conditions **OEC1** and **OEC2** ensure that the language L_a^{OE} is CSO with respect to X_s and SD, in the sense that the intruder is unable to estimate secret states when observing the events released by the Opacity-Enforcer, *i.e.*, events $\sigma_r \in \Sigma_r$.

In order to ensure that the Opacity-Enforcer has unique actions over the augmented sequences $s_a \in L_a^{OE}$, we still need to continue the pruning process. With a view to keeping the step delay bound of each event as low as possible, which makes the legitimate receiver estimation more accurate, we proceed according to the following rules: (i) the release (resp. deletion) of a event observation (**OEA1** (resp. **OEA2**)) has the highest (resp. lowest) priority among other actions the Opacity-Enforcer may take; (ii) if there is no observation to be released and no held observation is being delayed for its maximum step delay bound, then the Opacity-Enforcer must take no action (**OEA3**) and wait for an event occurrence, and; (iii) the Opacity-Enforcer must either release exactly one observation σ_r, or delete exactly one observation σ_d, or still perform no action (in this case, it waits for an event σ to occur in the system). Such rules are formally described by the following conditions:

UC1. $(\forall s_a \in L_a^{OE})\big((\exists \sigma_r \in \Sigma_r : s_a\sigma_r \in L_a^{OE}) \rightarrow s_a(\Sigma \cup \Sigma_d)\Sigma_a^* \cap L_a^{OE} = \emptyset\big)$;

UC2. $(\forall s_a \in L_a^{OE})\big((\exists \sigma \in \Sigma : s_a\sigma \in L_a^{OE}) \rightarrow s_a\Sigma_d\Sigma_a^* \cap L_a^{OE} = \emptyset\big)$;

UC3. $(\forall s_a \in L_a^{OE})\big((|L_a^{OE} \cap s_a\Sigma_r| \leq 1) \wedge (|L_a^{OE} \cap s_a\Sigma_d| \leq 1)\big)$.

Notice that, Conditions **UC1** and **UC2** ensure that **UB2** is never satisfied, whereas Condition **UC3** avoids the undesired behavior stated in **UB3**.

Example 3.6 *Let us consider the bounded delay augmented language generated by the automaton depicted in Figure 3.5, where it was assumed that* $SD(k) = \{(a,1),(b,0)\}$. *In order to obtain the Opacity-Enforcer language, we first set* $L_a^{OE} = L_a^{SD}$*, and then, we remove sequences from* L_a^{OE} *which either represent an undesirable behavior or leave the Opacity-Enforcer with non unique actions.*

Let us consider initially the pruning of L_a^{OE} *so as the remaining language satisfy Conditions* ***OEC1*** *and* ***OEC2****. The used sequences are obtained from the automaton in Figure 3.5.*

- *Sequence* $s_{a,1} = aa_rbb_rbb_daa$*, which reaches state* 20*, violates Condition* ***OEC1****, and must be removed, which implies that the transition from state* 19 *to* 20 *must be deleted. It follows that sequence* $s_{a,2} = aa_rbb_rbb_da$ *no longer satisfies Condition* ***OEC1****, and so must also be removed from* L_a^{OE}.

- *Sequence* $s_{a,3} = aa_rbb_rbb_d$*, which reaches state* 18*, violates Condition* ***OEC2*** *and should be excluded, since* $P_a(s_{a,3}) = abb$ *can be continued with* a *in* L*, but there is no continuation* $t_a \in L_a^{OE}/s_a$ *such that* $P_a(s_at_a) = abba$*. Now, the remaining sequence* $s_{a,4} = aa_rbb_rb$ *(state* 17*) does not satisfy Condition* ***OEC1*** *and must be deleted. By carrying out this process, we have to remove all sequences* $(aa_r + aba_r)\Sigma_a^*$ *from* L_a^{OE}.

Once L_a^{OE} *has only sequences that satisfy Conditions* ***OEC1*** *and* ***OEC2****, we continue the pruning process with a view to satisfying Conditions* ***UC1*** *–* ***UC3****. We present two examples of this pruning procedure.*

- *Sequence* $s_{a,5} = bb_rbb_da$*, which reaches state 5, can be continued with both* a *and* a_r*, as illustrated in Figure 3.5, which means that the Opacity-Enforcer has two options: to release an observation or wait for an event occurrence. Such a lack of unicity violates Condition* ***UC1****, and, thus, since observation release has the highest priority, we must remove all sequences* $bb_rbb_da(a_ra)^*a$ *from* L_a^{OE}*, i.e., state 6 is removed.*

- *Sequence* $s_{a,6} = b$ *also requires the Opacity-Enforcer to choose between releasing its observation (event* b_r*) or deleting it (event* b_d*), which violates Condition* ***UC2****. Thus, since observation deletions have the least priority, we must remove all sequences* $bb_d\Sigma_a^*$ *from* L_a^{OE}.

Figure 3.6 shows an automaton whose generated language is one of the possible feasible augmented languages $L_a^{OE} \subseteq L_a$ *of the language generated by automaton depicted in Figure 3.3.* □

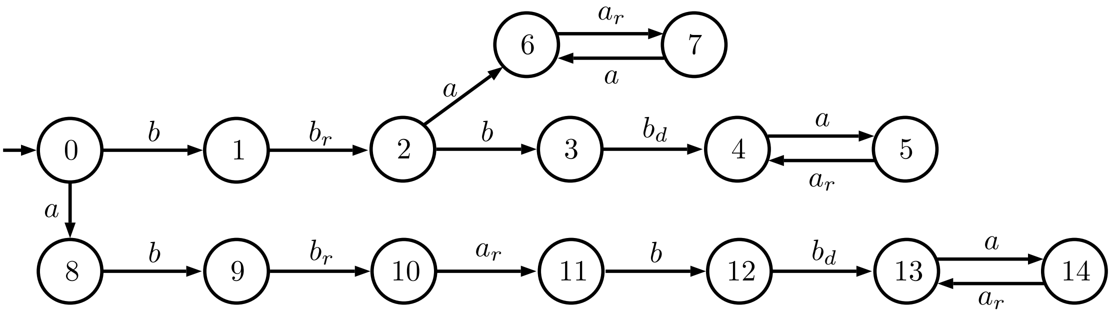


Figure 3.6: Automaton that generates a regular feasible language L_a^{OE}.

Remark 3.2 *Notice that Condition **UC3** is not violated in Example 3.6. However, when it is violated, i.e., two or more observations are allowed to be released (resp. deleted), a wise solution is to keep in L_a^{OE} only the continuation that corresponds to the action over the event occurrence that has first happened, and removing all other continuations. For example, if $s_a \in L_a^{OE}$ can be continued with both $\sigma_{1,r}$ and $\sigma_{2,r}$ but σ_1 has started being held before σ_2, i.e., $\|pre(s_a, \sigma_1^{k_1})\| < \|pre(s_a, \sigma_2^{k_2})\|$, $k_i = \mathcal{N}(s_a, \sigma_{i,r}) + \mathcal{N}(s_a, \sigma_{i,d}) + 1$ for $i = 1, 2$, then we should let only sequence $s_a\sigma_{1,r}\Sigma_a^*$ be in L_a^{OE}.* □

Remark 3.3 *As opposed to the works where edit functions are used to delete event observations, we are also interested in distinguishing if the event observation has been deleted or the observation release has been delayed, and so, dilation plays a key role in the strategy proposed here. For example, assume that an event σ has occurred in the plant but its observation has been deleted. Thus, if we remove event σ_d (that accounts for the deletion of σ) in the corresponding augmented sequence s_a, we will not be able to know, when the number of event occurrences of σ in the plant is greater than the number of occurrences of σ_r (that accounts for observation releases of σ), if the exceeding occurrences of σ have either been deleted or no action regarding event release has yet been taken by the Opacity-Enforcer.* □

3.4 Algorithms

We present two algorithms in this section. Algorithm 3.1 is an adaptation of the algorithm proposed by [6], and generates all possible shuffles of the event occurrences and observation releases/deletions. Algorithm 3.2 checks if the system is CSOE and returns an automaton that realizes the Opacity-Enforcement strategy.

3.4.1 Algorithm for shuffling event occurrences and observation releases/deletions

We propose an algorithm (Algorithm 3.1) to build an automaton (denoted as D), whose states, as in [6], represent queues of events that were generated by the system but have not been released/deleted yet. The symbol ν, that denotes "*blank space*", is also used in the labeling of the states of D to account for the observation release/deletion of an event that is not the first of the queue. For example, assume that we are building automaton D from the event set $\Sigma = \{a, b\}$ and that only event b is deletable. Thus, transitions labeled by events b_r and b_d will represent the change from the state labeled with aba to state $a\nu a$, and a transition labeled with event a_r will move automaton D from state aba to state ba. The blank space ν plays a crucial role in the modeling process, since it allows the count of the number of steps that event observations are being delayed. As a consequence, state ν is the initial state of D, since no event has occurred yet.

The algorithm presented here can be considered a special case of that proposed by [6] in the sense that there are no unobservable events to be considered here. The structure of automaton D represents the evolution of a buffer determined by $SD(k)$ regarding the occurrence of the events in Σ. The states of D represent the current stored event queue and the transitions represent either event occurrences or actions over event observations (release or deletion).

In order to label each state of automaton D, some operations over sequences $s \in (\Sigma \cup \{\nu\})^*$ are required, as follows.

Definition 3.8 *Let $\Sigma_\nu = \Sigma \cup \{\nu\}$ and let $Q \subseteq \Sigma_\nu^*$. We define the following functions:*

- *Replacement function. It is the mapping $rep : Q \times (\mathbb{N} \setminus \{0\}) \rightarrow Q$, such that $\forall q = q_1 q_2 \ldots q_l \in Q$,*

$$rep(q, i) = \begin{cases} q_1 q_2 \ldots q_{i-1} \nu q_{i+1} \ldots q_l, & \text{if } i \leq l \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

- *Cut function. It is the mapping $cut : Q \rightarrow Q$, where $\forall q = q_1 q_2 \ldots q_l \in Q$,*

$$cut(q) = \begin{cases} q_i q_{i+1} \ldots q_l, & \text{if } (\exists i \leq l)[(q_i \neq \nu) \wedge \\ & (q_k = \nu, \forall k \in \{1, 2, \ldots, i-1\})] \\ \nu, & \text{if } q_k = \nu, \ \forall k \in \{1, 2, \ldots, l\}. \end{cases} \qquad \square$$

In order to illustrate functions *rep* and *cut*, let us consider a state $q = ab \in Q$. We have that $cut(rep(q, 1)) = cut(\nu b) = b$, and $cut(rep(q, 2)) = cut(a\nu) = a\nu$.

Algorithm 3.1 computes automaton D, that generates all allowed shuffles of the event occurrences and observation releases/deletions. The idea behind this algorithm is to start from the initial state ν (empty queue), and, for a state $x_D = s$, we add to the right of s events σ that are allowed to occur according to $SD(k)$, and define transitions labeled by σ connecting x_D to a new state $x'_D = s\sigma$. Regarding the actions over the observation, if it is related to the first element of some state $x''_D = \sigma s$, both events σ_r and σ_d will be active in x''_D, defining transitions to state $x_D = s$; notice that event σ has been removed from queue σ_s. On the other hand, if some event in the middle of the queue that forms state $x'''_D = s_1\sigma^1 s_2$, where σ^1 is the first time event σ has happened in the queue that labels x'''_D, both events σ_r and σ_d will lead the automaton to state $x''''_D = s_1\nu s_2$; notice that, in this case, event σ is replaced with ν.

Algorithm 3.1 *Computation of automaton D*

```
Input: $\Sigma$, $\Sigma_D$, $k^{max} = [k^{max}_{\sigma_1}, \ldots, k^{max}_{\sigma_n}]$.
Output: $D = (X_D, \Sigma_a, f_D, \Gamma_D, x_{0,D})$.
  1. $x_{0,D} \leftarrow \nu$ and $X_D \leftarrow \emptyset$.
  2. Set $\Sigma_r = \varphi_r(\Sigma)$, $\Sigma_d = \varphi_d(\Sigma_D)$, and $\Sigma_a = \Sigma \cup \Sigma_r \cup \Sigma_d$.
  3. $F \leftarrow x_{0,D}$, where $F$ denotes a FIFO queue.
  4. While $F \neq \emptyset$, do:
     4.1. $u \leftarrow head[F]$
     4.2. If $u = x_{0,D}$, then:
          4.2.1. For each $\sigma \in \Sigma$:
                    Set $f_D(u, \sigma) = \sigma$.
                    $\Gamma_D(u) \leftarrow \Gamma_D(u) \cup \{\sigma\}$.
                    $Enqueue(F, \sigma)^2$.
          4.2.2. $X_D \leftarrow X_D \cup \{u\}$.
          4.2.3. $Dequeue(F)$.
     4.3. Else:
          4.3.1. Set $\ell = \|u\|$ and build $I_\ell = \{1, 2, \ldots, \ell\}$.
          4.3.2. Denote $u = \sigma^1\sigma^2 \ldots \sigma^\ell$ and create the set $I_\nu = \{y \in I_\ell : (\exists \sigma^y \in$
                 $u)[\sigma^y = \nu]\}$.
          4.3.3. $I_{\ell\setminus\nu} \leftarrow I_\ell \setminus I_\nu$.
          4.3.4. If $\|\sigma^y\sigma^{y+1} \ldots \sigma^\ell\| \leq k^{max}_{\sigma^y}$, for all $y \in I_{\ell\setminus\nu}$, then:
                    For each $\sigma \in \Sigma$:
                         Set $f_D(u, \sigma) = u\sigma$.
                         $\Gamma_D(u) \leftarrow \Gamma_D(u) \cup \{\sigma\}$.
                         $Enqueue(F, u\sigma)$.
          4.3.5. Set $\Sigma_{temp} = \Sigma$.
```

[2] For those who are not familiar with queue manipulation, Enqueue (resp. Dequeue) is the operation that inserts (resp. removes) an element in the last (resp. first) position of a queue.

```
4.3.6. For each $y \in I_{\ell \setminus \nu}$:
       If $\sigma^y \in \Sigma_{temp}$.
           $\Sigma_{temp} \leftarrow \Sigma_{temp} \setminus \{\sigma^y\}$.
           $\sigma_r \leftarrow \varphi_r(\sigma^y)$.
           Set $\widetilde{u} = cut(rep(u, y))$ and $f_D(u, \sigma_r) = \widetilde{u}$.
           $\Gamma_D(u) \leftarrow \Gamma_D(u) \cup \{\sigma_r\}$.
       If $(\widetilde{u} \notin X_D) \wedge (\widetilde{u} \notin F)$:
           $Enqueue(F, \widetilde{u})$.
       If $\sigma^y \in \Sigma_D$:
           $\sigma_d \leftarrow \varphi_d(\sigma^y)$.
           Set $f_D(u, \sigma_d) = \widetilde{u}$.
           $\Gamma_D(u) \leftarrow \Gamma_D(u) \cup \{\sigma_d\}$.
4.3.7. $X_D \leftarrow X_D \cup \{u\}$.
4.3.8. $Dequeue(F)$.
```

Algorithm 3.1 works as follows. It starts by creating the initial state $x_{0,D}$ in Step 1, labeling it by ν, since no event has occurred yet and the set of states X_D is set as empty. Step 2 creates sets Σ_r (resp. Σ_d) by labeling each event of Σ (resp. Σ_D) with subscript r (resp. d) by means of the release (resp. deletion) function $\varphi_r(\sigma)$ (resp. $\varphi_d(\sigma)$), and then, builds the augmented event set Σ_a. Step 3 creates a FIFO queue F and sets its first element as the initial state of D. Step 4 starts a loop where all possible event occurrences and observation releases/deletions are analyzed, which may require that new transitions and states be created in D. Notice that Step 4 is repeated until F becomes empty, meaning that no new state needs to be added. Step 4.1 chooses the first state u of queue F, and then, Step 4.2 checks if it is the initial state. If so, then Algorithm 3.1 proceeds to Step 4.2.1, where the transition function $f_D(\nu, \sigma) = \sigma$ is defined for each event $\sigma \in \Sigma$, and the active event set $\Gamma(x_{0,D})$ of the initial state is updated; then, these states labeled with σ are added to the queue F. After all states corresponding to the events of Σ have been created, Step 4.2.2 adds state u to the set of states X_D, and in the sequel, Step 4.2.3 removes u from F. On the other hand, if the current state u is not the initial state, Algorithm 3.1 skips to Step 4.3, where the length of u is measured, *i.e.*, $\ell = \|u\|$, and then, it creates a set whose elements are the positive integers smaller or equal to ℓ, *i.e.*, $I_\ell = \{1, 2, \ldots, \ell\}$. Step 4.3.2 searches for the positions of the blank spaces in the label of state u and then stores them in the integer set I_ν. Step 4.3.3 creates set $I_{\ell \setminus \nu}$, which has the positions of all events that were not released/deleted yet. Step 4.3.4 checks if every non released/deleted observation can still be delayed, and, if so, it creates new transition functions $f_D(u, \sigma) = u\sigma$ for every event $\sigma \in \Sigma$, updates the active event set $\Gamma(u)$, and insert the created states to queue F. In order to take into account observation releases, Step 4.3.5 creates event set Σ_{temp} that stores the events that can still be released from a given queue. Step 4.3.6 works in the ascending order of $y \in I_{\ell \backslash \nu}$, where for each $\sigma^y \in \Sigma_{temp}$, it removes σ^y from Σ_{temp}, so that each event σ is analyzed only once, and then, it sets the transition function by means of replacement and cut functions, updates the active event set, and adds every new state created in this step to queue F; in addition, if σ^y is a deletable event, Step 4.3.6 also adds a transition labeled with $\sigma_d = \varphi_d(\sigma^y)$ representing the observation deletion of event σ^y. Step 4.3.7 adds u to the set of states X_D and, finally, Step 4.3.8 removes u from the queue F.

We will now obtain the language generated by D, and to this end, let us first introduce the following function .

Definition 3.9 (Rearrangement function) *Given an event set* $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$*, the rearrangement function* $\chi : \Sigma^* \to 2^{\Sigma_a^*}$ *is a mapping with respect to* $SD(k) = \{(\sigma_1, k_{\sigma_1}), (\sigma_2, k_{\sigma_2}), \ldots, (\sigma_n, k_{\sigma_n})\}$ *and is defined as follows:*

$$\chi(s) = \{s_a \in P_a^{-1}(s) : s_a \vDash (i) \wedge (ii)\}$$

where, $\forall \sigma \in \Sigma$*:*

(i) $\mathcal{N}\big(s_a, \sigma\big) \geq \mathcal{N}\big(s_a, \varphi_r(\sigma)\big) + \mathcal{N}\big(s_a, \varphi_d(\sigma)\big)$*;*

(ii) $\|P_a(pre(s_a, \sigma_\ell^{i_\sigma}))\| - \|P_a(pre(s_a, \sigma^{i_\sigma}))\| \leq k_\sigma$*, where* $i_\sigma = 1, \ldots, \mathcal{N}(s_a, \sigma)$ *and* $\ell \in \{r, d\}$*.* □

Condition (i) ensures that $\chi(s)$ always has at least the same number of event occurrences as the sum of observation releases and deletions, and condition (ii) ensures that $SD(k)$ is not violated. Thus, rearrangement function χ returns all of the possible order rearrangement of observation releases and deletions regarding the step delay bound $SD(k)$. The rearrangement function can be extended to languages by applying it to each one of the sequences in the language, *i.e.*, $\chi(L) = \bigcup_{s \in L} \chi(s)$.

For example, given $\Sigma = \{a, b\}$, $SD(k) = \{(a, 1), (b, 0)\}$ and $\Sigma_D = \emptyset$, for a sequence $s = bab$, we have that $\chi(s) = \{bb_r a a_r b b_r, bb_r a b a_r b_r, bb_r a b b_r, bb_r a b b_r a_r\}$.

With the help of $\chi(s)$, we can state the following result.

Lemma 3.1 *The language generated by automaton D with respect to the event set Σ and the step delay bound $SD(k)$ is $\mathcal{L}(D) = \chi(\Sigma^*)$* □

Proof. The proof is omitted, since the algorithm for obtaining automaton D is a special case of Algorithm 1 by [6]. ■

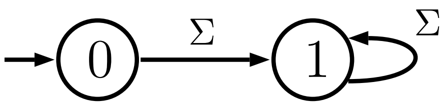


Figure 3.7: Model of a not CSOE system.

Let us now build automaton $G_{shf} = (X_{shf}, \Sigma_a, f_{shf}, \Gamma_{shf}, x_{0,shf}) = G \| D$, which models the shuffle of event occurrences subject to observation releases and deletions determined by $SD(k)$. Notice that, with the help of Lemma 3.1, it is not difficult to see that $\mathcal{L}(G_{shf}) = P_a^{-1}(L) \cap \chi(\Sigma^*) = \chi(L)$.

It is not difficult to see that the events released in a shuffled sequence of G_{shf} may lead the intruder to estimate a secret state or even conclude that some event occurrence is not allowed, thus violating Specification **S1** and Assumption **I4**, respectively. This problem can be solved if automaton G_{shf} is synchronized with an automaton G_{int} that models the allowed state estimations for the intruder, determined by Specification **S1** and Assumption **I4**. Automaton G_{int} is formed by removing all secret states from G and then adding subscript r to all events that label its transitions, *i.e.*, $G_{int} = Ac(X_{int}, \Sigma_r, f_{int}, \Gamma_{int}, x_{0,int})$, where $X_{int} = X \setminus X_s$, $f_{int}(x, \varphi_r(\sigma)) = f(x, \sigma)$, if $f(x,\sigma) \in X \setminus X_s$ or undefined, otherwise, $\Gamma_{int}(x) = \{\sigma_r \in \Sigma_r : f_{int}(x, \sigma_r)!\}$, and $x_{0,int} = \{x_0\} \cap X_{int}$. Notice that $L_{int} = \{\varphi_r(s) : (s \in L) \wedge (f(x_0, t) \in X \setminus X_s, \forall t \in \overline{s})\}$.

Finally, let us construct automaton $G_a^{SD} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}) = G_{shf} \| G_{int}$, which gives us all shuffled sequences so as the released observations neither lead the intruder to estimate secret states nor lie outside the modeled behavior. Thus, we may conclude that $\mathcal{L}(G_a^{SD}) = L_a^{SD} = \chi(L) \cap P_r^{-1}(L_{int})$.

We now present a necessary and sufficient condition for a regular language L to be CSOE with respect to Σ, Σ_D, $SD(k)$ and X_s, expressed in terms of L_a^{SD}.

Lemma 3.2 *A system, whose generated language and bounded delay augmented language are L and L_a^{SD}, respectively, is CSOE with respect to $SD(k)$, X_s and Σ_D if, and only if, $P_a(L_a^{SD}) = L$.*

Proof. Since $\chi(L) = P_a^{-1}(L) \cap \chi(\Sigma^*)$ and $L_a^{SD} = \chi(L) \cap P_r^{-1}(L_{int})$, we may conclude that $L_a^{SD} \subseteq P_a^{-1}(L)$, and thus $P_a(L_a^{SD}) \subseteq P_a(P_a^{-1}(L)) = L$. Let us now prove that the system is CSOE if, and only if, $L \subseteq P_a(L_a^{SD})$.

($\Leftarrow$) Assume that the system is not CSOE but that $L \subseteq P_a(L_a^{SD})$. Consider the system depicted in Figure 3.7, where $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ and whose generated language is $L = \Sigma^*$, and assume that $X_s = \{1\}$ (thus $L_{int} = \{\varepsilon\}$) and $\Sigma_D = \emptyset$, *i.e.*, no event observation can be deleted. If we set $SD(k) = \{(\sigma_1, 0), \ldots, (\sigma_n, 0)\}$, then it is not difficult to see that the system is not CSOE w.r.t. $SD(k)$, X_s and Σ_D. However, since $P_r^{-1}(L_{int}) = (\Sigma_a \setminus \Sigma_r)^* = \Sigma^*$ and $\chi(L) = \left(\bigcup_{i=1}^{n} \sigma_i \varphi_r(\sigma_i)\right)^*$, we may conclude that $L_a^{SD} = \{\varepsilon\} \cup \Sigma$. Thus, $P_a(L_a^{SD}) = L_a^{SD}$, which contradicts the assumption that $L \subseteq P_a(L_a^{SD})$.

($\Rightarrow$) Assume that the system is CSOE but that $L \nsubseteq P_a(L_a^{SD})$. Thus, there exists $s \in L$ such that $s \notin P_a(L_a^{SD})$, *i.e.*, for all $s_a \in P_a^{-1}(s)$, we have that $s_a \notin L_a^{SD}$. In addition, since $L_a^{SD} = \chi(L) \cap P_r^{-1}(L_{int})$, we may conclude that $\exists s \in L$ such that $(s_a \notin \chi(L)) \wedge (s_a \notin P_r^{-1}(L_{int})), \forall s_a \in P_a^{-1}(s)$, which means that there exists some sequence $s \in L$ that can not be shuffled and, at the same time, lies outside the allowed state estimation for the intruder, which means that s reaches a secret state, and also, there is no allowed shuffling that makes s looks like a sequence in G_{int}, implying that the system is not CSOE w.r.t. $SD(k)$, X_s and Σ_D, which contradicts the initial assumption. ■

Remark 3.4 *It is immediate from Lemma 3.2 that in order to verify if a given system modeled by an automaton G is CSOE with respect to $SD(k)$, X_s, Σ and Σ_D, we first compute $Obs(G_a^{SD}, \Sigma)$, mark all states of both $Obs(G_a^{SD}, \Sigma)$ and G, and then check if both $Obs(G_a^{SD}, \Sigma) \cap Comp(G)$ and $Comp(Obs(G_a^{SD}, \Sigma)) \cap G$ have empty marked language, where the $Comp(\cdot)$ is an operation that generates the complement language regarding a marked language (the readers are referred to [1]).* □

3.4.2 Algorithm for realization of the CSO enforcer

Before we proceed to the computation of the automaton that implements the CSO enforcer, we present another necessary and sufficient condition that guarantees the CSO enforceability of a system through changes and deletions in the order of event observations, this time, in terms of L_a^{OE}.

Lemma 3.3 *A system modeled by automaton $G = (X, \Sigma, f, \Gamma, x_0)$, is CSOE with respect to $\Sigma_D \subseteq \Sigma$, $X_s \subseteq X$ and $SD(k) = \{(\sigma_1, k_{\sigma_1}), (\sigma_2, k_{\sigma_2}), \ldots, (\sigma_n, k_{\sigma_n})\}$ through changes and deletions in the order of event observations if, and only if, there exists a non empty prefix closed augmented language L_a^{OE} obtained from L_a^{SD} whose sequences satisfy the Opacity-Enforcer Conditions, i.e., $\exists L_a^{OE} \subseteq L_a^{SD} :\ L_a^{OE} \neq \emptyset$.* □

Proof.
($\Rightarrow$) If the system is CSOE w.r.t Σ_D, X_s and $SD(k)$, *i.e.* $\forall s : f(x_0, s) \in X_s$, $\forall t \in L/s$, $\exists u \in L/st$ and $\exists v \in P_{dil}(Dil(S_p(stu))) \cap L$ such that $\forall w \in \overline{v}$, $f(x_0, w) \in X \backslash X_s$.

The proof is constructive and, for that, we will analyze two cases: (i) sequences s that reach a secret state, *i.e.*, $s \in L : f(x_0, s) \in X_s$, and; (ii) sequences s whose prefixes never lead to a secret state and whose continuations will never lead to a secret state either, *i.e.*, $s \in L : (f(x_0, s) \in X \setminus X_s) \wedge (f(x_0, st) \in X \setminus X_s, \forall t \in L/s)$. Notice that, we do not consider sequences $s \in L$ that have a continuation that leads to a secret state, since, in this case, the CSOE requires the knowledge about future event occurrences after s.

For case (i), according to Definition 3.3, for all $stu \in L$, there exists an augmented sequence $s_a \in \Sigma_a^* : (P_a(s_a) = stu) \wedge (s_a \vDash SD(k)) \wedge (\mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma)) = \mathcal{N}(s_a, \sigma), \forall \sigma \in \Sigma) \wedge (f(x_0, \varphi_r^{-1}(w_r)) \in X \setminus X_s, \forall w_r \in P_r(\overline{s_a}))$. consequently, $s_a \in L_a^{SD}$, $s_a \vDash$ **OEC1** (for the case where s_a has no continuation in L_a^{SD}, the number of event occurrences is equal to the sum of released and deleted observations), and, $s_a \vDash$ **OEC2** (since, if $\exists \sigma \in \Sigma : P_a(s_a)\sigma \in L$, we can set $t' = tu\sigma$, and, according to Definition 3.3, there must exists $u' \in L/st'$ such that there exists $s'_a \in L_a^{SD} : s'_a \vDash SD(k) \wedge s_a \in \overline{s'_a}$, and thus, there exists $s''_a \in \overline{s'_a} : P_a(s''_a) = stu\sigma = P_a(s_a)\sigma$). In case (ii), for sequences $s \in L : (f(x_0, s) \in X \setminus X_s) \wedge (f(x_0, st) \in X \setminus X_s, \forall t \in L/s)$, there exists $s_a \in [\bigcup_{\sigma \in \Sigma} \sigma(\varphi_r(\sigma) + \varphi_d(\sigma))]^* \cap L_a^{SD}$ such that $P_a(S_a) = s_a$, $s_a \vDash$ **OEC1** (since the observation of each event σ in s_a is immediately released/deleted after its occurrence) and $s_a \vDash$ **OEC2** (since, if $\exists \sigma \in \Sigma : P_a(s_a)\sigma \in L$, we can set $s' = s\sigma$ and find $s'_a : s_a \in \overline{s'_a}$ and $P_a(s'_a) = s'$). Therefore we can form a set L'_a with sequences of case (i) and (ii), and their prefixes, where every sequence in L'_a satisfies simultaneously Conditions **OEC1** and **OEC2**, and so, by setting $L_a^{OE} = L'_a$, we have that $\exists L_a^{OE} \subseteq L_a : \ L_a^{OE} \neq \emptyset$.

($\Leftarrow$) Assume that the system is not CSOE w.r.t. Σ_D, X_s and $SD(k)$. According to Definition 3.3, $\exists s : f(x_0, s) \in X_s$, and $\exists t \in L/s$, $\forall u \in L/st$, and $\forall v \in P_{dil}(Dil(S_p(stu))) \cap L$ is such that $\exists w \in \overline{v}$, $f(x_0, w) \in X_s$. Consider all sequences $s_a \in \Sigma_a^*$ such that $P_a(s_a) = stu$ and $\varphi_r^{-1}(P_r(s_a)) = v$. Since $\exists w \in \overline{v}$, $f(x_0, w) \in X_s$, there exists $w_a \in \overline{s_a}$ such that $\varphi_r^{-1}(P_r(w_a)) = w$, and therefore $w_a \notin L_a^{SD}$. Let w_{a_n}, $\|w_{a_n}\| = n \in \mathbb{N}$ be the longest prefix of w_a such that $w_{a_n} \in L_a^{SD}$. Assume that $w_{a_n} \in L_a^{OE}$. Thus, w_{a_n} cannot have a continuation in L_a^{OE} since it does not have one in L_a^{SD}, *i.e.*, $L_a^{OE}/w_{a_n} = \{\varepsilon\}$. We will now prove that $L_a^{OE} = \emptyset$ by showing that $w_a^{pre} \notin L_a^{OE}, \forall w_a^{pre} \in \overline{w_{a_n}}$. If there exists $\sigma \in \Sigma$ such that $\mathcal{N}\big(w_{a_n}, \sigma\big) > \mathcal{N}\big(w_{a_n}, \varphi_r(\sigma)\big) + \mathcal{N}\big(w_{a_n}, \varphi_d(\sigma)\big)$, then Condition **OEC1** would be violated, and thus, $w'_{a_n} \notin L_a^{OE}$. Assume now that Condition **OEC1** holds true. Since $w_{a_n} \in \overline{s_a}$, we may conclude that $P_a(w_{a_n}) \in \overline{P_a(s_a)} = \overline{stu}$. If $P_a(w_{a_n}) \neq stu$, then $\exists \sigma \in \Sigma : P_a(w_{a_n})\sigma \in L$. On the other hand, if $P_a(w_{a_n}) = stu$ and since Condition **OEC1** holds true, all event observations of sequence stu were either released or deleted, which is a contradiction, since in this case there would exist $w_a^{pre} \in \overline{w_{a_n}} : f(x_0, \varphi_r^{-1}(P_r(w_a^{pre}))) \in X_s$, which implies that $w_{a_n} \notin L_a^{SD}$. Since $w_{a_n} \in L_a^{OE}$ and $\exists \sigma \in \Sigma : P_a(w_{a_n})\sigma \in L$, according to **OEC2**, we have that $w'_{a_n} \in L_a^{OE} : \big(w_{a_n} \in \overline{w'_{a_n}}\big) \wedge \big(P_a(w'_{a_n}) = P_a(w_{a_n})\sigma\big)$, which is a contradiction, since we assumed that $L_a^{OE}/w_{a_n} = \{\varepsilon\}$, and thus, $w_{a_n} \notin L_a^{OE}$. Consider now sequence $w_{a_{n-1}} = w_{a_n}\sigma_a$ whose length is $n-1$ and $\sigma_a \in \Sigma_a^*$. If $\sigma_a \in \Sigma_r \cup \Sigma_d$, then $P_a(w_{a_n}) = P_a(w_{a_{n-1}})$, and since $\exists \sigma \in \Sigma : P_a(w_{a_n})\sigma \in L$, we may conclude that $\exists \sigma \in \Sigma : P_a(w_{a_{n-1}})\sigma \in L$. On the other hand, if $\sigma_a \in \Sigma$, then $P_a(w_{a_n}) = P_a(w_{a_{n-1}})\sigma_a$, which implies that $\exists \sigma_a \in \Sigma : P_a(w_{a_{n-1}})\sigma_a \in L$. In both cases, $w_{a_{n-1}}$ have a continuation according to **OEC2**, which is a contradiction unless $w_{a_{n-1}} \notin L_a^{OE}$. This procedure is carried out for all $w_a^{pre} \in \overline{w_{a_n}}$. It is worth remarking that $\forall s_a \in L_a^{OE} : \|s_a\| = 1$, we have that $s_a \in \Sigma$, since the first event of the augmented sequences must always be an occurrence in the system. In addition, for $w_{a_0} = \varepsilon$, there exists $w_{a_1} = \sigma_i \in \Sigma : P_a(w_{a_0})\sigma_i = \sigma_i \in L$ but there does not exist $s'_a \in L_a^{OE}$ such that $P_a(s'_a) = \sigma_i$, since $w_{a_1} = \sigma_i$ and its continuations are not in L_a^{OE}. Therefore, $\varepsilon \notin L_a^{OE}$, and since this language is prefix closed, we may conclude that $L_a^{OE} = \emptyset$. ■

We now propose Algorithm 3.2, which computes the realization of the Opacity-Enforcer automaton, to be denoted as $R_{OE} = (X_{OE}, \Sigma_a, f_{OE}, \Gamma_{OE}, x_{0,OE})$, whose generated language is L_a^{OE}. The idea behind Algorithm 3.2 is to prune the bounded delay shuffled language L_a^{SD} of G_a^{SD} so as to simultaneously satisfy Conditions **OEC1**, **OEC2**, **UC1**, **UC2** and **UC3**. To this end, all sequences that do not satisfy those conditions are recursively removed from the original augmented language L_a^{SD}. This procedure is repeated until the remaining language satisfy the desired behavior of the Opacity-Enforcer. We will assume with no harm that the input automaton G_a^{SD} is CSOE.

In addition, in order for the desired model of the Opacity-Enforcer to both never inhibit plant occurrences and ensure **OEC2**, we must verify, for each state $x_{OE} = \{x, x_D, x_{int}\}$ of automaton R_{OE}, the existence of some sequence s_a that satisfies $f_{OE}(x_{0,OE}, s_a) = x_{OE}$ and $f(x_0, P_a(s_a)\sigma)!$, for some $\sigma \in \Sigma$. If such a sequence s_a exists, then there must exist some $t_a \in (\Sigma_r \cup \Sigma_d)^*\sigma$ such that $f_{OE}(x_{0,OE}, s_a t_a)!$. To this end, we first define the Opacity-Enforcer action reach.

Definition 3.10 (Opacity-Enforcer action reach) *Given an automaton $G_a = (X_a, \Sigma_a, f_a, \Gamma_a, x_{0,a})$, the Opacity-Enforcer action reach is a mapping $OEAR : X_a \to 2^{X_a}$ that outputs all states x'_a reached from x_a through a sequence formed with events associated with actions taken by the Opacity-Enforcer. It is formally defined as follows.*

$$OEAR(x_a) = \{x'_a \in X_a : (\exists s_a \in (\Sigma_r \cup \Sigma_d)^*)[f_a(x_a, s_a) = x'_a]\}. \qquad \square$$

Having calculated all the states $x'_a \in X_{OE}$ reached from $x_a \in X_{OE}$ through actions taken by the Opacity-Enforcer, the next step is to compute among the active event sets of the states in $OEAR(x_a)$ those events that are plant events. To this end, we define the following function.

Definition 3.11 (Next) *Given an automaton $G_a = (X_a, \Sigma_a, f_a, \Gamma_a, x_{0,a})$, the function next is a mapping $NX : X_a \to 2^{\Sigma}$ as follows.*

$$NX(x_a) = \Big(\bigcup_{x'_a \in OEAR(x_a)} \Gamma_a(x'_a)\Big) \cap \Sigma. \qquad \square$$

Finally, the recursion of Algorithm 3.2 is performed as follows: we start with $L_{a_0}^{OE} = L_a^{SD}$, and, at each iteration, we create language $L_{a_{i+1}}^{OE} = L_{a_i}^{OE} \setminus L_{a_i}^{UB}$, where $L_{a_i}^{UB} = \{s \in L_{a_i}^{OE} : s \vDash UB1 \vee UB2 \vee UB3\}$. It stops when $L_{a_i}^{UB} = \emptyset$.

Algorithm 3.2 *Computation of automaton R_{OE}*

```
Input: $G_a^{SD} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD})$.
Output: $R_{OE} = (X_{OE}, \Sigma_a, f_{OE}, \Gamma_{OE}, x_{0,OE})$.
  1. Set $V = G_a^{SD} = (X_V, \Sigma_a, f_V, \Gamma_V, x_{0,V})$.
  2. flag ← true.
  3. While flag = true, do:
     3.1. flag ← false.
     3.2. For each $x_V = (x, x_D, x_{int}) \in X_V$:
          3.2.1. If $(x_D \neq \nu$ and $\Gamma_V(x_V) = \emptyset) \vee ((\Gamma(x) \cap \Gamma_D(x_D)) \setminus NX(x_V) \neq \emptyset)$,
                 then $X_V \leftarrow X_V \setminus \{x_V\}$, $V \leftarrow Ac(V)$, flag ← true.
  4. For each $x_V = (x, x_D, x_{int}) \in X_V$:
     4.1. If $\Gamma_V(x_V) \cap \Sigma_r \neq \emptyset$, then set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \Sigma_r$ and $V = Ac(V)$.
     4.2. If $\Gamma_V(x_V) \cap \Sigma \neq \emptyset$, then set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \Sigma$ and $V = Ac(V)$.
     4.3. If $|\Gamma_V(x_V) \cap (\Sigma_r \cup \Sigma_d)| > 1$, then:
          4.3.1. Denote $x_D = q_1 \ldots q_{\|x_D\|}$ and set $i = 1$.
          4.3.2. While $(\{\varphi_r(q_i), \varphi_d(q_i)\} \cap \Gamma_V(x_V) = \emptyset) \wedge (i \leq \|x_D\|)$, do $i = i + 1$.
          4.3.3. Set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \{\varphi_r(q_i), \varphi_d(q_i)\}$ and $V = Ac(V)$.
  5. Return $R_{OE} = V = (X_{OE}, \Sigma_a, f_{OE}, \Gamma_{OE}, x_{0,OE})$.
```

We now explain Algorithm 3.2 in details. STEP 1 creates a copy V of automaton G_a^{SD}. STEP 2 sets a flag equal to true, that will be used in the recursion of STEP 3, which removes all states x_V from automaton V that satisfies at least one of the following conditions: (i) $\Gamma_V((x, x_D, x_{int})) = \emptyset$ and $x_D \neq \nu$, which means that some event observation is being held forever; (ii) there is some event $\sigma \in \Sigma$ allowed to occur in x $(\sigma \in \Gamma(x))$ that does not violate $SD(k)$ $(\sigma \in \Gamma_D(x_D))$ but not allowed to occur in any of the states reached through actions taken by the Opacity-Enforcer $NX(OEAR(x_V))$, meaning that some event σ that can occur in G is being inhibited, which is an action that the Opacity-Enforcer can not perform. STEP 3 is repeated until the flag becomes false, which is only possible if no state in X_V needs to be deleted. Thus, STEP 3 of Algorithm 3.2 enforce Conditions **OEC1** and **OEC2**.

The strategy of shuffling event occurrences with their observations is likely to leave V with non-unique decision, *i.e.*, a state of V may have events of Σ, Σ_r and

Σ_d simultaneously in its active event set. In this case, the Opacity-Enforcer must either wait for the arrival of an event occurrence, release an event observation or even delete it. Such non-unique decisions are removed in Step 4, where we impose the following priority order for the Opacity-Enforcer actions. Firstly, release an observation event $\sigma_r \in \Sigma_r$ whenever it is possible (Step 4.1); secondly, wait for some event occurrence in the plant $\sigma \in \Sigma$ (Step 4.2), and; finally, delete an event observation $\sigma_d \in \Sigma_d$ only as last resource, hence, the least prioritized decision. In addition, Step 4.3 guarantees that the Opacity-Enforcer will either release or delete exactly one observation at each state; thus, if there is a state that has two or more allowed events to be released, Steps 4.3.1 and 4.3.2 find the event among them that occurred first and removes the others, which is performed in Step 4.3.3. Steps 4.1, 4.2 and 4.3 implement Unicity Conditions **UC1**, **UC2** and **UC3**, respectively. Finally, Step 5 computes the realization of Opacity-Enforcer automaton $R_{OE} = V$.

Remark 3.5 (Computational complexity of Algorithm 3.2) *All of the steps of Algorithm 3.2 are either constant or linear with respect to the number of states of V, except* Step 3*, where at least one state of automaton V is removed each time* Step 3 *is repeated; therefore, the fist time* Step 3 *is executed, $|X_V|$ states are analyzed, the second time, $|X_V| - 1$ states are analyzed, and as consequence, in the worst case $(1 + |X_V|) \times |X_V|/2$ states will be checked until V becomes empty, which implies that the computational complexity of Algorithm 3.2 is $O(|X_V|^2)$. In addition, since (i) $V = G_a^{SD} = G \| D \| G_{int}$, (ii) $|X_{int}| \leq |X|$ and (iii) $X_D = 1 + \left[\sum_{j=0}^{k}(|\Sigma| + 1)^j\right] \times |\Sigma| \leq c \times |\Sigma|^{k+1}$ [6], where $k = max(|k_{\sigma_1}|, \ldots, |k_{\sigma_n}|)$ and c is a big enough natural number, we may conclude that V has at most $c \times |X|^2 \times |\Sigma|^{k+1}$ states, and then, we may conclude that Algorithm 3.2 is $O(|X|^4 \times |\Sigma|^{2k+2})$. Notice that, starting with the observed model requires a priori computation of the observer, and so, $|X|$ is $O(2^{|X_p|})$, where $|X_p|$ is the cardinality of the set of states of the plant. Thus, with respect to the plant model, the worst case computational complexity of our method becomes $O(2^{4|X_p|} \times |\Sigma|^{2k+2})$.*

Theorem 3.1 *A system whose observer automaton $G = (X, \Sigma, f, \Gamma, x_0)$ is CSOE through changes and deletions in the order of event observations with respect to $\Sigma_D \subseteq \Sigma$, $X_s \subseteq X$ and $SD(k) = \{(\sigma_1, k_1), (\sigma_2, k_2), \ldots, (\sigma_n, k_n)\}$ if, and only if, automaton R_{OE} is such that $\mathcal{L}(Obs(R_{OE}, \Sigma)) = L$, i.e., $P_a(L_a^{OE}) = L$.* □

Proof. ($\Rightarrow$) Assume that the system is CSOE w.r.t. Σ_D, X_s and $SD(k)$. By construction, $L_a^{SD} = P_a^{-1}(L) \cap \chi(L) \cap P_r^{-1}(L_{int})$ and $L_a^{OE} \subseteq L_a^{SD}$, which implies that $L_a^{OE} \subseteq L_a^{SD} \subseteq P_a^{-1}(L)$, and therefore, $P_a(L_a^{OE}) \subseteq L$.

We will now prove that $L \subseteq P_a(L_a^{OE})$ by induction on the length of sequences.

(i) $s = \varepsilon \in L$. According to Lemma 3.3, $L_a^{OE} \neq \emptyset$, which implies that $\varepsilon \in P_a(L_a^{OE})$.

(ii) Assume that for all sequence $s = s_k \in L$ such that $\|s_k\| \leq k$, there exists at least one sequence $s_{a,k} \in L_a^{OE}$ such that $P_a(s_{a,k}) = s_k$.

(iii) Now, consider sequences $s_{k+1} = s_k\sigma \in L$. According to Lemma 3.2, $L = P_a(L_a^{SD})$, and thus, there exists sequences $s_{a,k+1} \in L_a^{SD}$ such that $P_a(s_{a,k+1}) = s_{k+1}$. Notice that automaton R_{OE} is obtained by pruning automaton V, which is a copy of G_a^{SD}, whose generate language is L_a^{SD}. Thus, if all sequences $s_{a,k+1}$ are removed in Step 3 of Algorithm 3.2, then all sequences whose projections have length $k+1$ violate either Condition **OEC1** or **OEC2**, which means that L_a^{SD} is not CSOE, contradicting the initial assumption. Thus, at least one sequence $s_{a,k+1} : P_a(s_{a,k+1}) = s_{k+1}$ remains in V after Step 3, and hence, there exists at least one sequence $s_{a,k+1} \in L_a^{OE}$ such that $P_a(s_{a,k+1}) = s_{k+1}$. Notice that, if a sequence $s'_{a,k+1}$ is removed from L_a^{SD} in Step 4 of Algorithm 3.2, then there must exist another sequence $s''_{a,k+1}$ such that $P_a(s''_{a,k+1}) = s_{k+1}$.

Thus, $L \subseteq P_a(L_a^{OE})$, which implies that $L = P_a(L_a^{OE})$.

($\Leftarrow$) Assume that $L = P_a(L_a^{OE})$. Since $\mathcal{L}(R_{OE}) \subseteq \mathcal{L}(G_a^{SD})$, we have that $\mathcal{L}(Obs(R_{OE}, \Sigma)) \subseteq \mathcal{L}(Obs(G_a^{SD}, \Sigma))$, therefore, $L \subseteq P_a(L_a^{SD})$. By construction, $L_a^{SD} = P_a^{-1}(L) \cap \chi(L) \cap P_r^{-1}(L_{int})$, which implies that $L_a^{SD} \subseteq P_a^{-1}(L)$, and therefore, $P_a(L_a^{SD}) \subseteq L$. Therefore, $L \subseteq P_a(L_a^{SD}) \subseteq L$, which implies that $L = P_a(L_a^{SD})$, and so, according to Lemma 3.2, the system is CSOE w.r.t. Σ_D, X_s and $SD(k)$. ■

Remark 3.6 *Given a system that is CSOE w.r.t. Σ_D, X_s and $SD(k)$, it is not difficult to see that, after the computation of automaton G_a^{SD} and the execution of Algorithm 3.2, the language L_a^{OE} generated by the resulting automaton R_{OE} is such that all of its sequences are in accordance with Conditions **OEC1-OEC2** and **UC1-UC3**. In addition, R_{OE} is the realization of the Opacity-Enforcer, keeping track not only of the events executed by the system but also of the release and deletion of their observations. Notice that, if we input an automaton G_a^{SD} whose generated language is not CSOE, then* Step 3 *of Algorithm 3.2 will remove all sequences from V, and thus, the obtained automaton R_{OE} will be empty. This result is in accordance with Lemma 3.3, since when a system is not CSOE, either $L_a^{OE} = \emptyset$ or there exists some sequence $s_a \in L_a^{OE}$ that violates Conditions **OEC1** or **OEC2**. The latter cannot happen since* Step 3 *enforces Conditions **OEC1** and **OEC2**.* □

According to Theorem 3.1, automaton R_{OE} obtained according to Algorithm 3.2 satisfies condition **OE2**. In order to show that the realization of automaton R_{OE} solves the problem of opacity enforcement, we now must prove that R_{OE} also satisfies condition **OE1**. This result is presented in Proposition 3.1 as follows.

Proposition 3.1 *Assume that the system modeled by G is not CSO with respect to X_s and Σ and define $X_{s,OE} = \{x_{OE} = (x, x_D, x_{int}) \in X_{OE} : x \in X_s\}$ as the set of secret states of automaton R_{OE}. Then, automaton R_{OE} obtained from G according to Algorithm 3.2 is CSO with respect to $X_{s,OE}$ and Σ_r.*

Proof. All sequences $s_a \in L_a^{OE}$ that reaches secret states of $X_{s,OE}$, *i.e.*, $f_{OE}(x_{0,OE}, s_a) \in X_{s,OE}$, are such that $f(x_0, P_a(s_a)) \in X_s$, since its first component is a secret state. In addition, according to Definition 3.5 and 3.7, $f\Big(x_0, \varphi_r^{-1}\big(P_r(s_a)\big)\Big) \in X \setminus X_s$, since its third component is not a secret state. Since sequence $\varphi_r^{-1}\big(P_r(s_a)\big)$ reaches a non secret state, it will be, according to Definition 3.5 and 3.7, augmented so as its projections over Σ and Σ_r correspond to the same sequence, and thus, there must exist a sequence $s'_a \in L_a^{OE}$ such that $P_r(s'_a) = P_r(s_a)$ and that $P_a(s'_a) = \varphi_r^{-1}\big(P_r(s_a)\big)$, which implies that there exists a state $x' \in X \setminus X_s$ such that $f(x_0, P_a(s'_a)) = x'$, and, as a consequence, there exists a state $x'_{OE} = (x', x'_D, x'_{int}) \in X_{OE} \setminus X_{s,OE}$ such that $f_{OE}(x_{0,OE}, s'_a) = x'_{OE}$. Consequently, automaton R_{OE} is CSO with respect to $X_{s,OE}$ and Σ_r. ■

Remark 3.7 *According to Algorithms 3.1 and 3.2, there are two trivial ways to enforce CSO w.r.t. Σ_D, X_s and $SD(k)$. The first one is by setting k_{σ_i} infinite, $i = 1, \ldots, |\Sigma|$, and $\Sigma_D = \emptyset$, i.e., all event observations can be held indefinitely and none of them is deletable. In this case, the resulting enforcer automaton will be $R_{OE} = G$, which implies that $L_a^{OE} = L$. Notice that, $P_r(L_a^{OE}) = \varepsilon$, meaning that the Opacity-Enforcer releases no observation. The second trivial way to enforce CSO is by setting each $k_{\sigma_i} = 0$, $i = 1, \ldots, |\Sigma|$ and $\Sigma_D = \Sigma$, i.e., no event observation can be delayed but all of them are deletable. Thus, the language generated by the resulting enforcer automaton R_{OE} will be such that $L_a^{OE} = \{s_a \in \big(\bigcup_{\sigma \in \Sigma} \sigma\varphi_d(\sigma)\big)^* : P_a(s_a) \in L\}$, meaning that every event observation is deleted immediately after its occurrence. The corresponding Opacity-Enforcer automaton can be constructed as follows. For each transition $f(x, \sigma) = y$, we first remove it, and then add state x' and transitions $f(x, \sigma) = x'$ and $f(x', \sigma_d) = y$. Notice that, in this case we also have that $P_r(L_a^{OE}) = \varepsilon$. In both cases, the estimation of the intruder will cease at the initial state, which we assumed to be a non secret state.* □

3.5 Example

In order to illustrate the CSO enforcing strategy proposed here, consider automaton G shown in Figure 3.8, where the initial state is $x_0 = 0$, the observable event set is $\Sigma = \{a, b, c, d\}$ and the set of secret states is $X_s = \{5\}$. Notice that the model for state estimation of both, legitimate receiver and intruder, is given by G, since all of its events are observable.

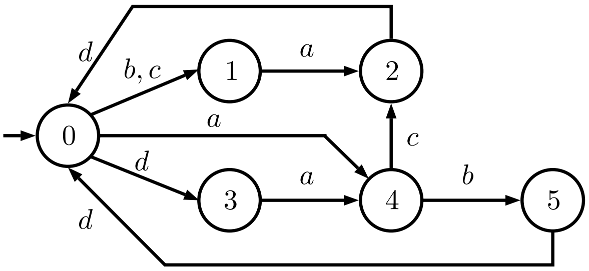


Figure 3.8: Automaton G used to illustrate the CSO enforcement strategy.

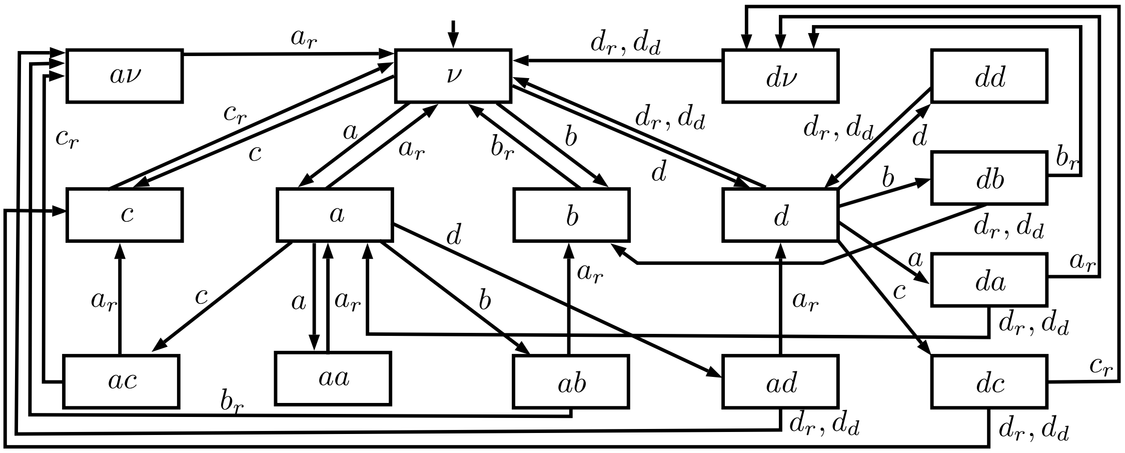


Figure 3.9: Automaton D

Let us consider the following setup for CSO enforcement: observation release of events a and d can be held for at most one step, whereas events b and c must be released immediately after their occurrences. In addition, only event d is deletable. Thus, we set $SD(k) = \{(a, 1), (b, 0), (c, 0), (d, 1)\}$ and $\Sigma_D = \{d\}$. According to Theorem 3.2, in order to verify if the language generated by the system is CSOE with respect to the current setup, we compute G_a^{SD} and verify if $P_a(L_a^{SD}) = L$. To this end, we build automaton D according to Algorithm 3.1, shown in Figure 3.9. Then, we create automaton G_{int} as a copy of G, remove all secret states (in the example, state 5 only), take its accessible part and add subscript r to all of its events. Finally, we build automaton $G_a^{SD} = G \| D \| G_{int}$. Proceeding according to Remark 3.4, we may conclude that the system is CSOE w.r.t. Σ_D, X_s and $SD(k)$. Due to its size, the state transition diagram of G_a^{SD} has not been depicted.

Therefore, we can proceed to the construction of the Opacity-Enforcer automaton R_{OE}. Algorithm 3.2 starts by setting $V = G_a^{SD}$ and flag = true in Steps 1 and 2, respectively. Step 3 will keep deleting undesirable states of V until there is no state $x_V = (x, x_D, x_{int})$ that violates Conditions **OEC1** or **OEC2**, which is performed in Step 3.2.1. Figure 3.10 shows part of automaton V when Step 3.2.1 is executed for the first time. Notice that, the red colored states violate Condition **OEC1**, $x_D \neq \nu \wedge \Gamma_V(x_V) = \emptyset$, which means that these states have some observation being

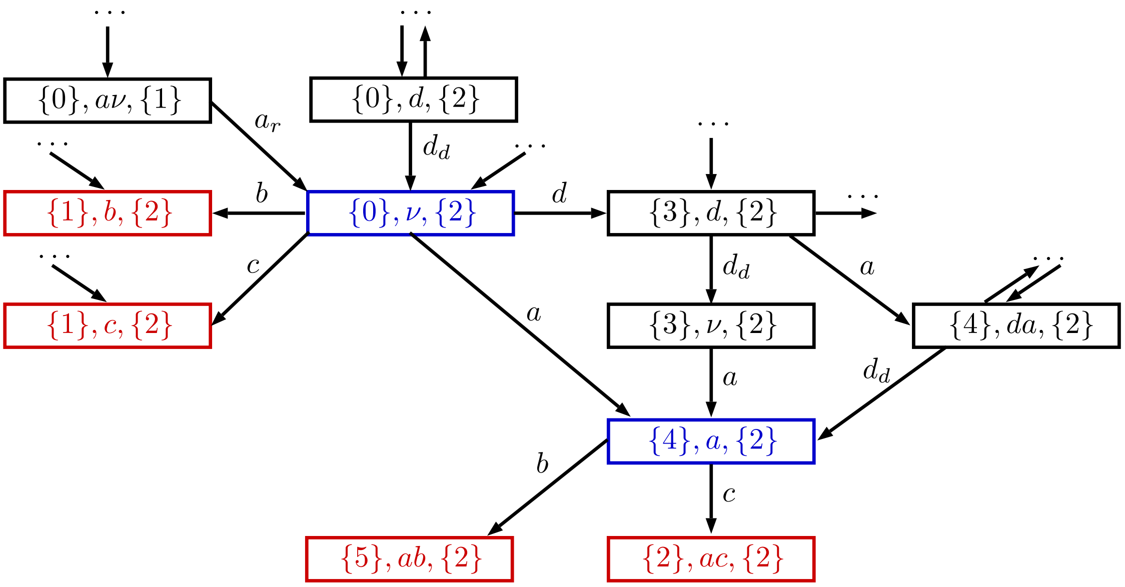


Figure 3.10: Part of V that shows undesirable states.

held indefinitely, and, thus, they must be removed. When those states are removed, the blue colored states become undesirable the second time Step 3.2.1 is executed, since they violate Condition **OEC2**, $(\Gamma(x) \cap \Gamma_D(x_D)) \setminus \Gamma_V(x_V) \neq \emptyset$, meaning that some events $\sigma \in \Sigma$ should be inhibited, which is an action that cannot be performed by the Opacity-Enforcer, and so, these states must also be removed. It is worth remarking that Algorithm 3.2 always sets $V = Ac(V)$ at the end of Step 3.2.1.

When all the remaining sequences satisfy simultaneously Conditions **OEC1** and **OEC2**, Algorithm 3.2 skips to Step 4, where all transitions from automaton V that violate Conditions **UC1**–**UC3** are removed. Figure 3.11 shows red colored transitions and states that must be removed due to the existence of possible non unique actions. Notice that, state $(\{2\}, a, \{1\})$ has transitions labeled with a_r and d, and so, Step 4.1 will remove the transition labeled with d, in order for the Opacity-Enforcer to release an observation as soon as possible (**UC1**). In state $(\{3\}, d, \{0\})$, the Opacity-Enforcer may either wait for the occurrence of a or delete the observation of d; therefore, Step 4.2 will remove the transition labeled with d_d, since observation deletions must be performed only as the last resource (**UC2**). Finally, state $(\{2\}, ac, \{0\})$ has two observation releases defined, and so, Step 4.3 will remove the transition labeled with c_r, since event a happened before c (**UC3**). Finally, Step 5 computes $R_{OE} = V$, which is shown in Figure 3.12.

Notice that the components in state $x_{OE} = (x, x_d, x_{int})$ of automaton R_{OE}, shown in Figure 3.12, provide the following information: component x shows the current state of the system, x_D shows the events that are being held by the Opacity-Enforcer, and, x_{int} is the intruder's state estimation. In addition, automaton R_{OE} keeps track of both the system dynamics and the observation released to the intruder. For

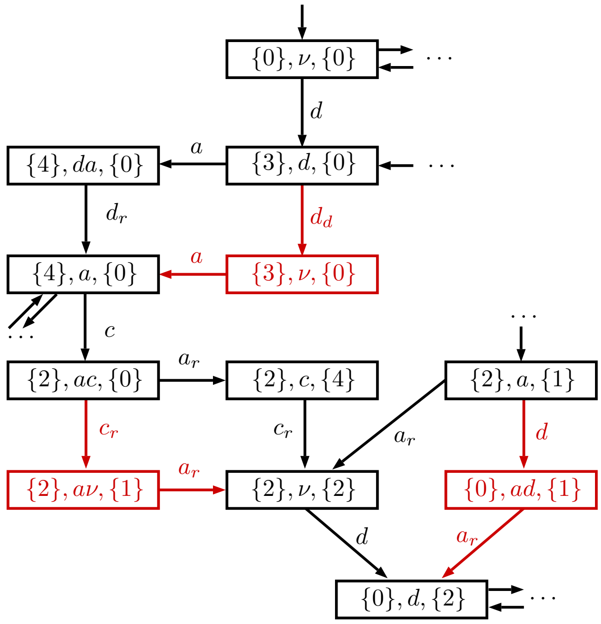


Figure 3.11: Part of V that shows decision conflicts.

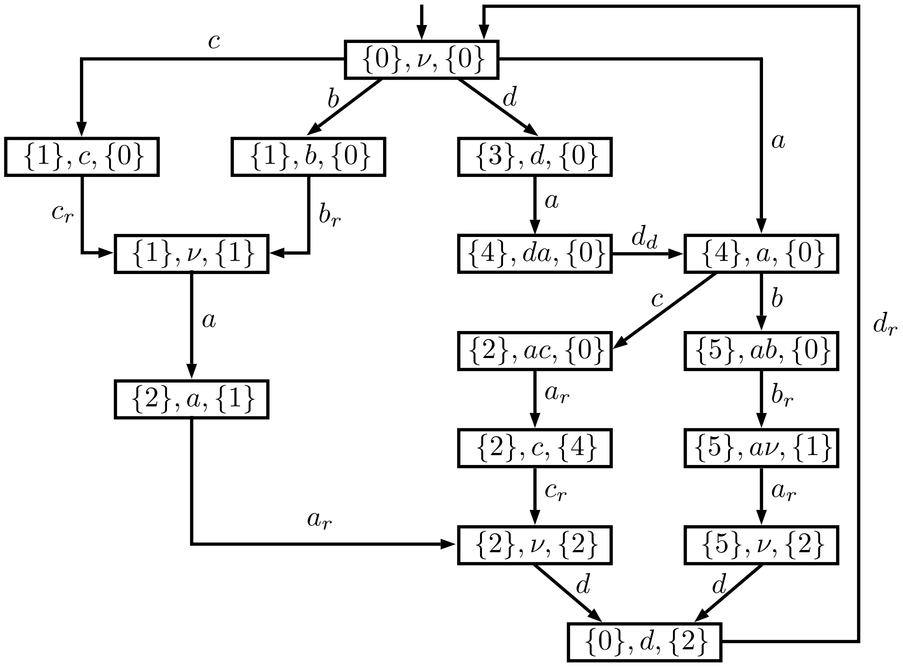


Figure 3.12: Automaton R_{OE} that realizes the opacity enforcement strategy.

example, in automaton R_{OE} depicted in Figure 3.12, when G generates sequence $s = dab$, *i.e.*, the system reaches the secret state 5, the corresponding augmented sequence in R_{OE} will be $s_a = dad_d bb_r a_r$, which determine the actions to be taken by the Opacity-Enforcer and also their order. In this case, sequence s_a denotes that the Opacity-Enforcer will initially hold events d and a, and then delete the observation of d (d_d) and wait for event b to occur before releasing the observations of b and a ($b_r a_r$). Since the sequence of released events is $b_r a_r$, the intruder estimates the system as being in state 2. Finally, notice that all information is represented in the state of automaton R_{OE} reached by sequence s_a, namely, state $x_{OE} = (\{5\}, \nu, \{2\})$, which denotes that the current state of G is $x = 5$, there is no event being held ($x_D = \nu$), and the intruder estimates state $x_{int} = 2$.

3.6 Mitigating the negative effect of opacity enforcement on the legitimate receiver's state estimate capability

One of the criticisms to opacity theory is that, when the information is also intended to be sent to some receiver that needs to be aware of the system evolution, both the intruder and the legitimate receiver are misled when opacity is enforced. In this section, we present a protocol that, when applied to the strategy proposed here, is capable of mitigating the negative effect of opacity enforcement on the capability of the legitimate receiver to accurately estimate the current state of the system. This protocol leverages the legitimate receiver's knowledge on the actions to be taken by the Opacity-Enforcer in order to refine the estimation of the current state of the system.

We recall that the state estimation of the legitimate receiver and the intruder may differ since the legitimate receiver has complete knowledge on the opacity enforcement strategy applied to the system. The estimates of the legitimate receiver are based on the first component of each state of automaton R_{OE}, whereas the estimate of the intruder is based on its third component. In this regard, the model of the state estimates of the legitimate receiver is obtained by computing the observer automaton $G_{est,R} = Obs(R_{OE}, \Sigma_r, 1)$, where the index 1 is used to indicate that the state estimations are computed with respect to the first component of the states of R_{OE}. On the other hand, the real state estimations of the intruder are given by the observer automaton $G_{real,int} = Obs(R_{OE}, \Sigma_r, 3)$, which can be also obtained by computing the accessible part of automaton G after all secret states have been removed and its transitions labeled with subscript r.

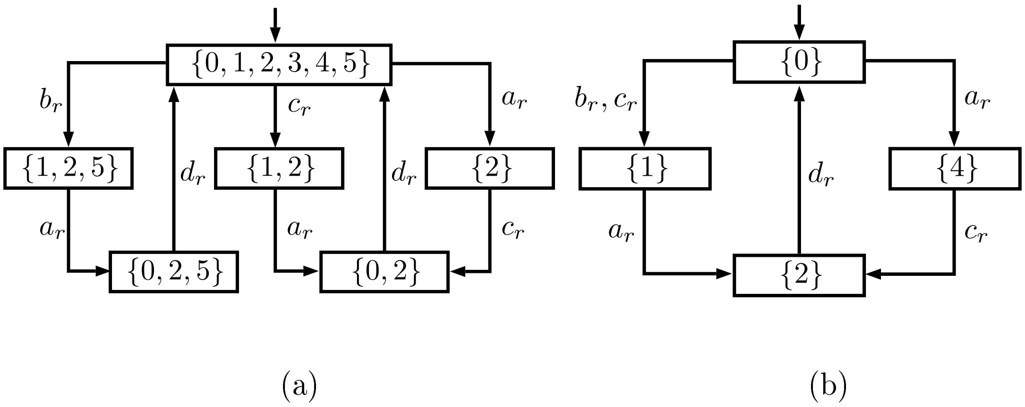


Figure 3.13: Automata $G_{est,R}$ (a) and $G_{real,int}$ (b) with the states capable of being estimated by the legitimate receiver and the intruder, respectively.

Example 3.7 *Consider the plant modeled by automaton G, shown in Figure 3.8, and whose Opacity-Enforcer automaton R_{OE} is depicted in Figure 3.12. The legitimate receiver's state estimates of the current state of the system after each event release is given by automaton $G_{est,R} = Obs(R_{OE}, \Sigma_r, 1)$ shown in Figure 3.13(a), whereas the states the intruder estimates correspond to automaton $G_{real,int} = Obs(R_{OE}, \Sigma_r, 3)$, depicted in Figure 3.13(b). Notice that, when event a is released (occurrence of a_r) the intruder estimates that the system is in state 4, whereas the legitimate receiver is sure that the systems is in state 2, which is actually correct, since, according to Figure 3.12, event a is only released after the occurrences of either sequences ac or dac in the plant. In addition, when the Opacity-Enforcer releases events b_r and a_r, in this order, the intruder believes that the plant is in state 2, whereas the legitimate receiver has a more accurate state estimate ($\{0, 2, 5\}$), which means that the plant may have visited the secret state.* □

It is worth remarking that, if the existence of the Opacity-Enforcer becomes public (Assumption **I3**, regarding the intruder being unaware of the Opacity-Enforcer, is dropped), then the state estimation model used by the intruder becomes identical to that of the legitimate receiver, since the intruder is now aware of the opacity-enforcement strategy. Notice that, in this case, the intruder is still unable to accurately infer secret states when the current state of the system corresponds to them, and thus, the system is still current-state opaque from the point of view of the intruder when the strategy for CSO enforcement is publicly known.

By dropping Assumption **I1**, which states that the intruder has complete knowledge of the system, and then by assuming that the model of the system that is publicly known may differ from its real model, the legitimate receiver's estimation can be further improved if the information regarding the real sequence that will be executed by the plant is known a priori. To this end, let G_e be the automaton whose generated language is formed with the real sequences that will be executed by the

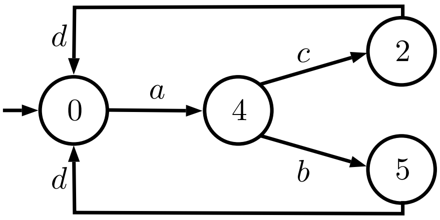


Figure 3.14: Automaton G_e whose generated language corresponds to the actual sequences to be executed by the plant.

plant. Algorithm 3.3 presents the construction of automaton $G^e_{est,R}$ with the state estimation of the legitimate receiver, given that it is aware of G_e.

Algorithm 3.3 *Computation of automaton $G^e_{est,R}$*

```
Input: $R_{OE}$ and $G_e$.
Output: $G^e_{est,R}$.
  1. Compute $R^e_{OE} = G_e \| R_{OE}$.
  2. Compute $G^e_{est,R} = Obs(R^e_{OE}, \Sigma_r, 1)$.
  3. Return $G^e_{est,R}$.
```

Notice that, in Step 1 of Algorithm 3.3, we remove from R_{OE} all sequences that will not be executed by the plant, which are the ones that make the legitimate receiver's current state estimation less accurate. Next, in Step 2, we compute automaton $G^e_{est,R}$ as the observer of R^e_{OE} with respect to its first component, thus obtaining a refined state estimation.

Example 3.8 *Let G_e, depicted in Figure 3.14, be the automaton which corresponds to the real sequences to be executed by the plant modeled in Figure 3.8, whose Opacity-Enforcer automaton R_{OE} is shown in Figure 3.12. In addition, assume that the legitimate receiver knows the model of G_e in advance. According to Algorithm 3.3, we obtain automaton $G^e_{est,R}$, which gives all of the legitimate receiver's estimations, as shown in Figure 3.15. Notice that the legitimate receiver has now a much more accurate estimation of the current state of the plant. In particular, according to Figure 3.15, whenever event b_r is released, the legitimate receiver is sure that the system is in the secret state (5), whereas, according to Figure 3.13(b), the intruder always estimates that the plant is in state 1.* □

The above discussion shows that the strategy proposed here has the potential to be used to address opacity enforcement in order to avoid misleading the receiver as well. In this regard, an Opacity-Enforcer could be designed to establish some compromise between achieving opacity with respect to a malicious intruder and accuracy of receiver's estimation.

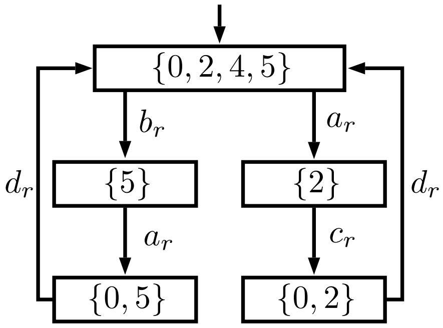


Figure 3.15: Automaton $G^e_{est,R}$ with the state estimations of the legitimate receiver.

3.7 Concluding remarks

In order for the intruder to be always misled to wrongly estimate non-secret states, the CSO enforcement strategy proposed in this chapter leverages the possibility of delaying and deleting some event observations.

The Opacity-Enforcer proposed in this chapter keeps track not only of the events executed by the system but also release and deletion of their observation signals, and has shown to have the potential to be used to enforce opacity and, at the same time, not mislead the legitimate receiver as much as the intruder.

In addition, we have presented a protocol that, when applied to the strategy proposed in this chapter, is capable of mitigating the negative effect of opacity enforcement on the capability of the legitimate receiver to accurately estimate the current state of the system. This protocol leverages the legitimate receiver's knowledge on the actions to be taken by the Opacity-Enforcer in order to refine the estimation of the current state of the system.

Chapter 4

Ensuring utility while enforcing opacity

In this chapter, we approach the main criticism of opacity enforcement techniques presented in the literature, namely the fact that in order to make the system behavior opaque to the intruder, the observation of the legitimate receiver becomes obfuscated as well. In most cases, the crucial information the legitimate receiver requires is not necessarily the secret behavior of the system (which must be concealed from the intruder in order for the system to be opaque). In this regard, we introduce the notion of (state-based) utility to refer to such a crucial behavior, that is required to always be available to the legitimate receiver, even when opacity is being enforced. To this end, we propose an algorithm that, whenever possible, ensures the utility of the system and, at the same time, enforces current-state opacity. Although the algorithm we propose in this chapter relies on the same strategy as that presented in Chapter 3, it is simpler.

This chapter is structured as follows. Section 4.1 presents the architecture of the system considered in this chapter, the problem to be solved and introduces the notion of utility; Section 4.2 recalls the strategy to augment languages through shuffles and deletions of event observations presented earlier in Chapter 3. In addition, the conditions required for the augmented language to model not only the behavior of the system, but also the event observation release/deletion policy, so as to ensure utility while enforcing opacity are also given in Section 4.2; Section 4.3 presents the algorithm developed to achieve both utility and current-state opacity enforcement; Section 4.4 illustrates the strategy proposed in the chapter with a toy example. Finally, Section 4.5 summarizes all of the contributions of the chapter.

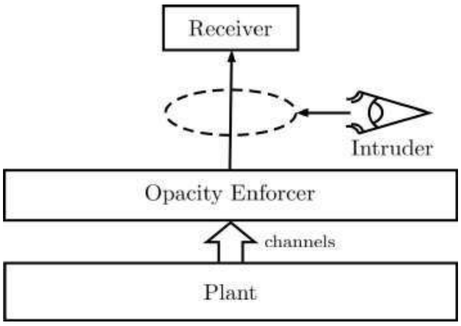


Figure 4.1: The opacity enforcement architecture.

4.1 Problem formulation

We consider here the same architecture as that assumed in Section 3.1 and depicted again in Figure 4.1, which is composed of a plant, an Opacity-Enforcer and two players, a legitimate receiver and an intruder. The observable events generated by the plant are transmitted in the same order as their actual occurrences to the Opacity-Enforcer, which is now responsible for manipulating the order of event observation releases/deletions in order for the following two tasks be simultaneously achieved: (i) the intruder is misled to never infer that the secret behavior has been executed (opacity enforcement), and; (ii) the legitimate receiver must always be aware of the system useful behavior when it occurs (preserve utility).

Since the notion of utility approached in this chapter is state-based, we define the set of useful states X_u, which, when visited, characterizes the useful behavior of the system. It is worth mentioning that, since we require that all useful states are unambiguously observed by the legitimate receiver, any secret state is obfuscated to the intruder and both legitimate receiver and intruder have the same access to the events released by the Opacity-Enforcer, as a consequence, in order for the Opacity-Enforcer to ensure the utility of the system and enforce CSO, a secret state must never be useful, and vice-versa. We say that, if the visit to useful states is unambiguously observed by external agents, *i.e.*, when all external agents are sure that the system is in those states, then the system is said to have ensured-utility (EU). The formal definition is as follows.

Definition 4.1 (Ensured-Utility Systems) *Let a system be modeled by automaton* $G = (X, \Sigma, f, \Gamma, x_0)$*, projection* $P_o : \Sigma^* \to \Sigma_o^*$ *and let* X_u *be the set of useful states. Then, system* G *has ensured-utility with respect to* P_o *and* X_u *if*

$$(\forall x_u \in X_u)(\forall s \in L, f(x_0, s) = x_u)(\forall x' \in X \setminus \{x_u\})$$
$$\big(\forall s' \in L, (f(x_0, s') = x') \to (P_o(s) \neq P_o(s'))\big) \quad \square$$

According to Definition 4.1, a system has ensured-utility with respect to P_o and

X_u when it is possible, under the projection P_o, to distinguish every useful state x_u in X_u from any other state $X \setminus \{x_u\}$ of the system, which includes the fact that x_u must be distinguished even from other useful states $x'_u \in X_u, x'_u \neq x_u$. On the other hand, if there exist states $x_u \in X_u$ and $x' \in X \setminus \{x_u\}$ and sequences $s, s' \in L$ such that $f(x_0, s) = x_u$, $f(x_0, s') = x'$ and whose observations are identical, *i.e.*, $P_o(s) = P_o(s')$, then the legitimate receiver cannot distinguish the visit to the useful state x_u from another state x' when sequence $P_o(s)$ is observed, therefore, according to Definition 4.1, the system does not have ensured-utility with respect to P_o and X_u.

Notice that the notions of utility and (weak) detectability[1] [25, 26] may be alike, in the sense that both of them require that the current state must be determined in finite time. However, they differ in two aspects: (i) utility requires that we are capable of determining the current state of the system only if it is a useful state in X_u, as opposed to detectability, which is concerned with determining any current state of X and also its subsequent states; (ii) every useful state must always be distinguished from any other states, whereas detectability requires that every state eventually becomes distinguishable from other states.

It is worth highlighting that both the design specification **S1** and all of the assumptions made on the intruder's capacity **I1–I4** are also assumed here (see Section 3.1).

4.2 Ensuring utility while enforcing opacity

One of the drawbacks of enforcing opacity is that the legitimate receiver also becomes unable to distinguish some behaviors of the plant, where one of them may even provide crucial information required by the legitimate receiver. With that in mind, we introduce the notion of utility, which is the useful behavior of the system that we require to always be available to the legitimate receiver. In this regard, besides enforcing opacity, any Opacity-Enforcer to be designed has an additional task, namely preserving the utility of the system, which is accomplished whenever the following (utility) specifications hold true simultaneously.

US1. Whenever the plant visits an useful state, the legitimate receiver must estimate it as soon as possible;

US2. Whenever the legitimate receiver estimates an useful state, the plant must currently be in that state.

[1] A discrete event system is (weakly) detectable if we can determine the current state and the subsequent states of the system after a finite number of observations for some trajectories of the system [25].

In order to enforce current-state opacity, the Opacity-Enforcer must mislead the intruder's estimation of the current state of the system if it is currently in a secret state and, to this end, it leverages the possibility of either delaying/deleting the observation of events, as detailed in Section 3.3.

We recall that the strategy for ensuring utility while enforcing opacity presented here modifies the sequence of observations outputted by the plant but not its actual behavior. Thus, given that $G_p = (X_p, \Sigma_p, f_p, \Gamma_p, x_{0,p})$ is the automaton that models the behavior of the plant and Σ denotes its observable event set, we define its observer as $G = Obs(G_p, \Sigma) = (X, \Sigma, f, \Gamma, x_0)$, and so, all of the operations and functions hereafter defined will be performed over the language $L = \mathcal{L}(G)$. We also recall that X_s denotes the set of secret states of the observer.

We assume, without loss of generality, that the plant modeled by automaton G_p has ensured-utility with respect to projection $P : \Sigma_p^* \to \Sigma^*$ and the set of useful X_u; otherwise the task of ensuring the utility of a system that has useful states that are not inherently indistinguishable is not possible. As a consequence of the plant being an ensured-utility system, when we compute its observer automaton $G = Obs(G_p, \Sigma)$, the states $x \in X$ of G are either composed of not useful states $x \in 2^{X_p \setminus X_u}$ or are composed of a unique useful state $x \in X_u$. Therefore, the useful states of the observer are the same as the useful states of the plant, and will be denoted as X_u as well.

Notice that the task of the Opacity-Enforcer can be viewed as an event observations release/deletion manipulation, and thus, the event set Σ_a of the automaton that models the Opacity-Enforcer must be composed of: (i) the events generated by the plant Σ; (ii) the release of their observations Σ_r, and; (iii) the deletion of their observations Σ_d (if the event is deletable). Thus, we define the augmented event set as $\Sigma_a = \Sigma \dot{\cup} \Sigma_r \dot{\cup} \Sigma_d$. We also recall the following projections.

1) $P_a : \Sigma_a^* \to \Sigma^*$;
2) $P_r : \Sigma_a^* \to \Sigma_r^*$.

Moreover, in order to model the policy of the Opacity-Enforcer regarding event observation release/deletions (**OEA1–OEA3**) restricted to step delay bound $SD(k)$, we can augment the language L generated by G to the bounded delay augmented language L_a^{SD} according to Definition 3.7, which augments the sequences of L by adding to them event observation releases and deletions (not necessarily in the same order as they occurred) according to the restrictions of the step delay bound $SD(k)$ (which prevents L_a^{SD} from being a non regular language and from events being indefinitely held by the Opacity-Enforcer).

Example 4.1 *Let $G = (X, \Sigma, f, \Gamma, x_0)$, depicted in Figure 4.2, be the automaton that models the observed behavior of a plant, whose secret behavior is defined by the*

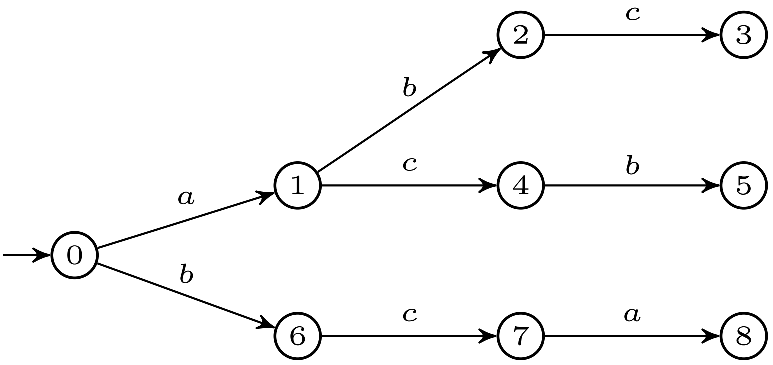


Figure 4.2: Automaton G.

*visit to state 3, i.e., $X_s = \{3\}$. Notice that, by setting $SD(k) = \{(a,0),(b,1),(c,0)\}$ and $\Sigma_D = \emptyset$, it is possible for the sequence $s = abc$, which reaches the secret state 3, to be augmented to sequence $s_a = aa_rbcc_rb_r$, which means that the observation of event b has been delayed for one step, and now, whenever sequence s occurs in the system, all external agents observe $\varphi_r^{-1}(P_r(s_a)) = \varphi_r^{-1}(a_rc_rb_r) = acb$, and hence, they estimate state 5 instead of the secret state 3. However, if we define state 4 as an useful state ($X_u = \{4\}$), it is not possible to ensure utility while enforcing opacity in the system, since as mentioned previously, when $s = abc$ is generated, $\varphi_r^{-1}(P_r(s_a)) = acb$ is observed by external agents, whose prefix ac lead them to estimate the useful state 4 whereas the current state of the system does not correspond to it, which violates specification **US2**. On the other hand, if we now set $SD(k') = \{(a,2),(b,0),(c,0)\}$, then sequence $s = abc$ can now be augmented to sequence $s'_a = abb_rcc_ra_r$, which is observed by external agents as $\varphi_r^{-1}(P_r(s_a)) = bca$. In this case, utility has been ensured while opacity was being enforced, since none of the prefixes of $\varphi_r^{-1}(P_r(s_a)) = bca$ mislead the legitimate receiver to wrongly estimate the useful state 4 and the intruder estimates state 8 whereas the system is currently in the secret state 3.* □

We recall from Section 3.3 (see page 38) that the Opacity-Enforcer must not hold indefinitely the observation of non deletable events (**UB1**), and thus, it is necessary to prune L_a^{SD} to obtain a prefix-closed augmented language L_a^{OE}. To this end, we remove from L_a^{SD} sequences s_a and s_at_a, for every $t_a \in L_a^{SD}/s_a$, so that $P_a(L_a^{OE}) = L$ and all of the remaining sequences $s_a \in L_a^{OE}$ satisfy simultaneously the following Opacity-Enforcer conditions:

OEC1. $(\forall s_a \in L_a^{OE})\big((L_a^{OE}/s_a = \emptyset) \rightarrow (\forall \sigma \in \Sigma, \mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma)) = \mathcal{N}(s_a, \sigma))\big)$;

OEC2. $(\forall s_a \in L_a^{OE})\big((\exists \sigma \in \Sigma : P_a(s_a)\sigma \in L) \rightarrow (\exists t_a \in L_a^{OE}/s_a : P_a(s_at_a) = P_a(s_a)\sigma)\big)$.

We also recall that Condition **OEC1** ensures that no sequence in L_a^{OE} satisfies undesirable behavior **UB1** and Condition **OEC2** ensures that $P_a(L_a^{OE}) = L$, which guarantees that the language L_a^{OE} is CSO with respect to X_s and Σ_D, in the sense that the intruder is unable to estimate secret states when observing the events $\sigma_r \in \Sigma_r$ released by the Opacity-Enforcer.

In order for the Opacity-Enforcer to also ensure utility, the policy of event observation release/deletion must satisfy both specifications **US1** and **US2**. To this end, we must remove from L_a^{OE} all sequences s_a and $s_a t_a$, for every $t_a \in L_a^{OE}/s_a$, that violates either **US1** or **US2**, resulting in a new prefix-closed augmented language L_a^{UOE} such that: (i) $P_a(L_a^{UOE}) = L$, and; (ii) all of sequences $s_a \in L_a^{UOE}$ not only satisfy **OEC1** and **OEC2** simultaneously, but also the following additional Opacity-Enforcer conditions:

OEC3. $(\forall s_a \in L_a^{UOE})\big((\exists x_u \in X_u)[f(x_0, P_a(s_a)) = x_u \wedge f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u] \rightarrow (s_a \Sigma \Sigma_a^* \cap L_a^{UOE} = \emptyset)\big)$;

OEC4. $(\forall s_a \in L_a^{UOE})\big((\exists x_u \in X_u)[f(x_0, P_a(s_a)) \neq x_u \wedge f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u] \rightarrow (s_a \cap \Sigma_a^* \Sigma_r = \emptyset)\big)$.

Notice that Condition **OEC3** ensures that when the system is currently in some useful state $x_u \in X_u$ but the legitimate receiver has not estimated it yet, then the Opacity-Enforcer must release/delete event observations before the system evolves (occurrence of some event $\sigma \in \Sigma$) with a view for the legitimate receiver to estimating state x_u as soon as possible. Condition **OEC4** ensures that the Opacity-Enforcer must not release any event observation $\sigma_r \in \Sigma_r$ that makes the legitimate receiver estimate some useful state $x_u \in X_u$ when the current state of the system is not x_u. Therefore, Conditions **OEC3** and **OEC4** guarantee that language L_a^{UOE}, which models the behavior of the Opacity-Enforcer, satisfy specifications **US1** and **US2** simultaneously. It is worth noticing that, if the resulting language L_a^{UOE} is not empty and satisfies both Conditions **OEC1** and **OEC2**, then, with the help of Lemma 3.3, the system modeled by automaton G is CSOE. Finally, we conclude that L_a^{UOE} is capable of ensuring utility while opacity is being enforced.

Example 4.2 *Consider again automaton $G = (X, \Sigma, f, \Gamma, x_0)$ shown in Figure 4.2, where $X_s = \{3\}$, $X_u = \{4\}$ and $\Sigma_D = \emptyset$. In order to illustrate the Opacity-Enforcer Conditions **OEC3** and **OEC4**, we set $SD(k) = \{(a, 2), (b, 0), (c, 1)\}$. Let H, depicted in Figure 4.3, be an automaton whose generated language is $\mathcal{L}(H) = L_a^{OE}$, which satisfies Conditions **OEC1** and **OEC2** simultaneously. Notice that each state of automaton H has three components: (i) the first component denotes the current state of the plant; (ii) the second component denotes the event observations that are being held by the Opacity-Enforcer (which must release/delete them eventually),*

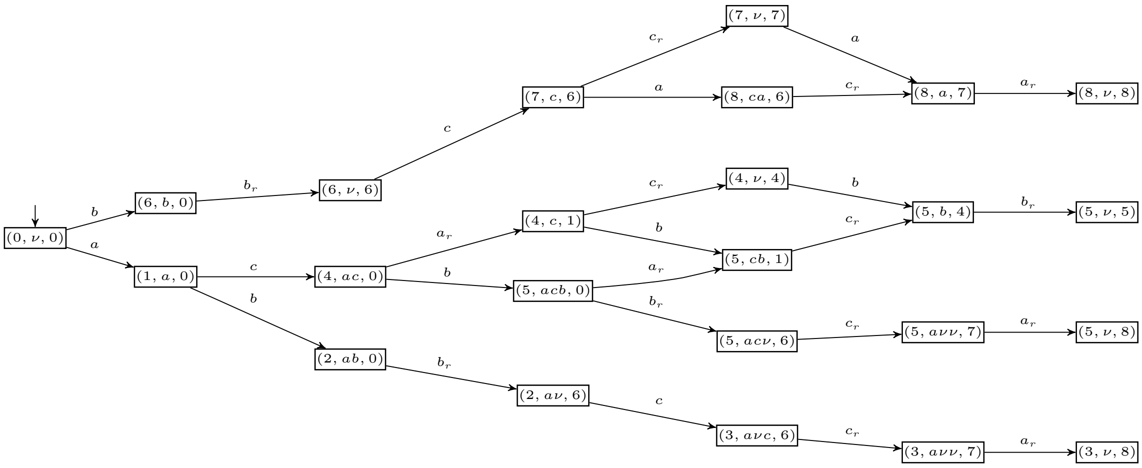


Figure 4.3: Automaton H.

and; (iii) the third component denotes the state estimated by external observers (legitimate receiver and intruder). It is not difficult to see that some of the sequences in $\mathcal{L}(H)$ violate either ***OEC3*** *or* ***OEC4****, for example:*

- *Sequence $s_a = ac$ reaches state $(4, ac, 0)$, which means that the plant is currently in useful state 4 but external observes are estimating state 0 ($f(x_0, P_a(ac)) = f(x_0, ac) = 4$ and $f(x_0, \varphi_r^{-1}(P_r(ac))) = f(x_0, \varepsilon) = 0$). However, some of its continuations are such that $s_a \Sigma \Sigma_a^* \cap \mathcal{L}(H) = acb(\overline{b_r c_r a_r} + \overline{a_r c_r b_r}) \neq \emptyset$, which means that the plant can evolve to state 5 through the occurrence of event b without the legitimate receiver estimating the useful state 4, and thus, these continuations must be removed so as the resulting language satisfies Condition* ***OEC3****.*

- *Sequence $s_a = aca_r$, which reaches state $(4, c, 1)$, also violates Condition* ***OEC3****, since it can be continued with event b, and thus, in order to satisfy Condition* ***OEC3****, sequence aca_rb and all of their continuations must be removed.*

- *Sequences $acba_rc_r$ and aca_rbc_r, which reach state $(5, b, 4)$, violate Condition* ***OEC4****, since they model the case when the system is currently at state 5 but the legitimate receiver has just estimated useful state 4, since the last event of those sequences represent an observation release, meaning that the Opacity-Enforcer has just released an event observation that misled the legitimate receiver to estimate an useful state while the plant is not currently at it. Then, we must remove from $\mathcal{L}(H)$ those sequences and also their continuations, so as the resulting language satisfies Condition* ***OEC4****.*

Notice that, after the removal of those sequences that violate Conditions ***OEC3*** *and* ***OEC4****, it may be the case that the resulting language has sequences that now* *violate Condition **OEC1**, which should also be removed. Therefore, the procedure for obtaining an augmented language L_a^{UOE} that ensures utility while opacity is being enforced is performed recursively.* □

Remark 4.1 *To the best of our knowledge, the notion of utility withing opacity enforcement strategies has been presented firstly in [71], where the authors synthesize obfuscation policies through edit functions that ensure privacy and utility at the same time. However, in their approach, the utility of the plant is ensured when the "distance" between the current and the estimated state never exceeds a predefined maximum value. Differently from the strategy proposed in [71], we require that the event observations outputted by the Opacity-Enforcer are such that, whenever the system visits an useful state, it must be estimated by the legitimate receiver before a new state is reached, and also that whenever the legitimate receiver estimates an useful state, the plant must currently be in that state.* □

It is not difficult to see that, when the language generated by some plant is augmented with event observation releases/deletions, the current state of the plant may differ from the state estimated by external observers, since for a given $s_a \in L_a^{SD}$, it is possible that $f(x_0, P_a(s_a)) \neq f(x_0, \varphi_r^{-1}(P_r(s_a)))$. Therefore, we extend the notion of EU systems (see Definition 4.1) to augmented systems, as follows.

Definition 4.2 (Ensured-Utility Augmented Systems) *Let a system, whose observable behavior L is modeled by an automaton $G = (X, \Sigma, f, \Gamma, x_0)$, be augmented to automaton G_a, where $\mathcal{L}(G_a) \subseteq L_a^{SD}$, with respect to the set of deletable events Σ_D, secret states X_s and step delay bounds $SD(k)$. Let X_u denote the set of useful states. The augmented system modeled by automaton G_a is said to have ensured-utility if the following two conditions hold true simultaneously:*

(i) $\forall s_a \in \Sigma_a^* \Sigma \cap \mathcal{L}(G_a)$ *if* $f(x_0, P_a(s_a)) = x_u \in X_u$, *then* $\forall t_a \in \mathcal{L}(G_a)/s_a$, $\exists t'_a \in (\Sigma_r \cup \Sigma_d)^*$ *such that* $f(x_0, \varphi_r^{-1}(P_r(s_a t'_a))) = x_u$ *and either* $t_a \in \overline{t'_a}$ *or* $t'_a \in \overline{t_a}$.

(ii) $\forall s_a \in \Sigma_a^* \Sigma_r \cap \mathcal{L}(G_a)$, *if* $f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u \in X_u$, *then* $f(x_0, P_a(s_a)) = x_u$.

□

According to Definition 4.2, an augmented system has ensured-utility when it simultaneously satisfies two conditions: (i) if an augmented sequence $s_a \in \mathcal{L}(G_a)$ denotes that the system has just reached an useful state $x_u \in X_u$, *i.e.*, $s_a \in \Sigma_a^* \Sigma$ and $f(x_0, P_a(s_a)) = x_u$, then each one of its continuations $t_a \in \mathcal{L}(G_a)/s_a$ must be such that either the useful state x_u will certainly be estimated before the current state of the system changes, *i.e.*, $\exists t'_a \in (\Sigma_r \cup \Sigma_d)^*$ such that $f(x_0, \varphi_r^{-1}(P_r(s_a t'_a))) = x_u$ and $t_a \in \overline{t'_a}$, or the useful state x_u was estimated before the current state of the

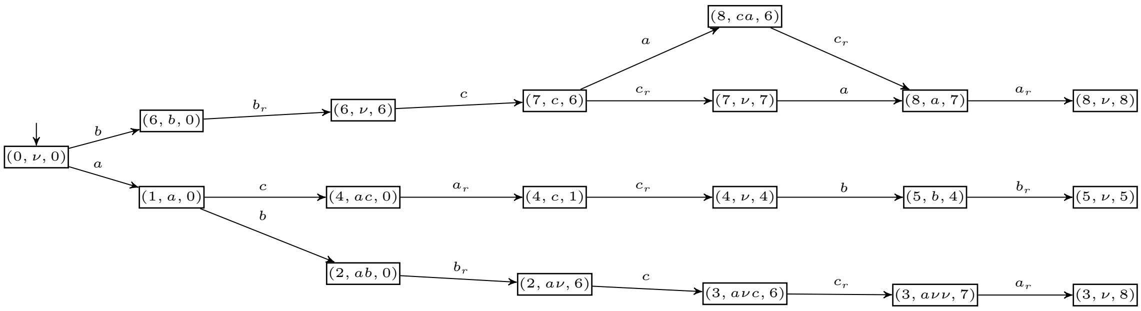


Figure 4.4: Automaton H'.

system had changed, *i.e.*, $\exists t'_a \in (\Sigma_r \cup \Sigma_d)^*$ such that $f(x_0, \varphi_r^{-1}(P_r(s_a t'_a))) = x_u$ and $t'_a \in \overline{t_a}$, and; (ii) if an augmented sequence $s_a \in \mathcal{L}(G_a)$ ends with an event observation release, *i.e.*, $s_a \in \Sigma_a^*\Sigma_r$ and leas to the estimation of an useful state $f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u \in X_u$, then the current state of the system must be that useful state as well, *i.e.*, $f(x_0, P_a(s_a)) = x_u$.

Example 4.3 *Consider again automaton* $G = (X, \Sigma, f, \Gamma, x_0)$*, depicted in Figure 4.2, where* $X_s = \{3\}$*,* $X_u = \{4\}$*,* $\Sigma_D = \emptyset$ *and* $SD(k) = \{(a, 2), (b, 0), (c, 1)\}$*. Let* H'*, illustrated in Figure 4.4, be the automaton that generates a language* L_a^{UOE} *that satisfies Conditions* ***OEC1–OEC4*** *with respect to* $\mathcal{L}(G)$*. Notice that automaton* H' *has only one sequence that models when the system has just visited an useful state, namely sequence* $s_a = ac$*, whose last event is a plant event and denotes that the current state of the system is an useful state, i.e.,* $f(x_0, P_a(s_a)) = f(x_0, ac) = 4 \in X_u$*. In addition, all of the continuations of sequence* s_a *are such that: (i) either the legitimate receiver will certainly estimate the useful state (when* $(4, \nu, 4)$ *is visited), namely continuations* $t_a = \varepsilon$ *and* $t'_a = a_r$*, or; (ii) the useful state was visited before the current state of the system has changed, namely continuations* $t''_a = a_r c_r$*,* $t'''_a = a_r c_r b$ *and* $t''''_a = a_r c_r b b_r$*. Therefore, the first condition of Definition 4.2 is met. With respect to the second condition of Definition 4.2, sequence* $s_a = aca_r c_r$ *is the unique sequence in* $\mathcal{L}(H')$ *whose last event is an observation release, i.e.,* $s_a \in \Sigma_a^*\Sigma_r$*, and leads the legitimate receiver to estimate an useful state, i.e.,* $f(x_0, \varphi_r^{-1}(P_r(s_a))) = f(x_0, \varphi_r^{-1}(a_r c_r)) = f(x_0, ac) = 4 \in X_u$*. Notice also that* $s_a = aca_r c_r$ *denotes that the current state of the system is that same useful state, since* $f(x_0, P_a(s_a)) = f(x_0, ac) = 4$*, which makes* H' *satisfy the second condition of Definition 4.2, and thus, the augmented system modeled by automaton* H' *has ensured-utility.* □

We now extend Definition 3.3 to our current problem, namely to ensure utility while current-state opacity is being enforced in augmented systems, as follows.

Definition 4.3 (EU-CSO Augmented Systems) *Let a system, whose observable behavior* L *is modeled by an automaton* $G = (X, \Sigma, f, \Gamma, x_0)$*, be augmented to* *automaton G_a, where $\mathcal{L}(G_a) \subseteq L_a^{SD}$, with respect to the set of deletable events Σ_D, secret states X_s and step delay bounds $SD(k)$. Let X_u denote the set of useful states. The augmented system modeled by automaton G_a is said to have ensured-utility current-state opaque (EU-CSO) with respect to Σ_D, X_s, X_u and $SD(k)$ through changes and deletions in the order of event observations if:*

(i) Automaton G_a models an EU augmented system, and;

*(ii) $\mathcal{L}(G_a) \neq \emptyset$ and $\mathcal{L}(G_a) \vDash$ (**OEC1** $\wedge$ **OEC2**).* □

After an augmented language L_a^{UOE} that simultaneously satisfies Conditions **OEC1–OEC4** has been obtained, we present the following result.

Lemma 4.1 *Let a system, whose observable behavior L is modeled by an automaton $G = (X, \Sigma, f, \Gamma, x_0)$, be augmented to automaton G_a, where $\mathcal{L}(G_a) \subseteq L_a^{SD}$, with respect to the set of deletable events Σ_D, secret states X_s and step delay bounds $SD(k)$. Let X_u denote the set of useful states. The augmented system modeled by automaton G_a is said to have ensured-utility current-state opacity (EU-CSO) with respect to Σ_D, X_s, X_u and $SD(k)$ through changes and deletions in the order of event observations if, and only if, $\mathcal{L}(G_a) = L_a^{UOE} \neq \emptyset$ and $\mathcal{L}(G_a) \vDash$ (**OEC1** $\wedge$ **OEC2** $\wedge$ **OEC3** $\wedge$ **OEC4**).*

Proof.

($\Rightarrow$) Assume that either $\mathcal{L}(G_a) = \emptyset$ or $\mathcal{L}(G_a) \nvDash$ (**OEC1** $\vee$ **OEC2** $\vee$ **OEC3** $\vee$ **OEC4**). It is clear that, if either $\mathcal{L}(G_a) = \emptyset$, or $\mathcal{L}(G_a) \nvDash$ **OEC1**, or, still, $\mathcal{L}(G_a) \nvDash$ **OEC2**, then the second condition of Definition 4.3 is violated, which implies that automaton G_a does not model any EU-CSO augmented system.

If $\mathcal{L}(G_a) \nvDash$ **OEC3**, then, $\exists s_a \in \mathcal{L}(G_a)$ such that $f(x_0, P_a(s_a)) = x_u \in X_u$, $f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u$, and $s_a \Sigma \Sigma_a^* \cap \mathcal{L}(G_a) \neq \emptyset$, which means that s_a can be continued with some plant event $\sigma \in \Sigma$. If s_a ends with some plant event ($s_a \in \Sigma_a^* \Sigma$), then $\exists t_a = \sigma \in \mathcal{L}(G_a)/s_a$ where, $\forall t'_a \in (\Sigma_r \cup \Sigma_d)^*$, $t_a \notin \overline{t'_a}$ and only $t'_a = \varepsilon \in \overline{t_a}$, which implies that $f(x_0, \varphi_r^{-1}(P_r(s_a t'_a))) = f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u$. As a consequence, the first condition of Definition 4.2 has been violated, and hence, G_a does not model any EU augmented system. Therefore, according to Definition 4.3, automaton G_a does not model any EU-CSO augmented system.

If $\mathcal{L}(G_a) \nvDash$ **OEC4**, then, $\exists s_a \in \mathcal{L}(G_a)$ such that $f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u \in X_u$, $f(x_0, P_a(s_a)) \neq x_u$ and $s_a \cap \Sigma_a^* \Sigma_r \neq$, which means that the last event of sequence s_a is some $\sigma_r \in \Sigma_r$ ($s_a \in \Sigma_a^* \Sigma_r \cap \mathcal{L}(G_a)$). It is clear that the second condition of Definition 4.2 has been violated, and so, G_a does not model any EU augmented system. According to Definition 4.3, automaton G_a does not model any EU-CSO augmented system either.

($\Leftarrow$) Assume that $\mathcal{L}(G_a) = L_a^{UOE} \neq \emptyset$ and $\mathcal{L}(G_a) \vDash$ (**OEC1** $\wedge$ **OEC2** $\wedge$ **OEC3** $\wedge$ **OEC4**). It is clear that the second condition of Definition 4.3 holds true.

If $\mathcal{L}(G_a) = L_a^{UOE} \vDash$ **OEC3**, then, $\forall s_a \in L_a^{UOE}$ such that $f(x_0, P_a(s_a)) = x_u \in X_u$ but $f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u$, s_a cannot be continued with any plant event $\sigma \in \Sigma$, and as a consequence, it is continued with sequences $t_a \in (\Sigma_r \cup \Sigma_d)^*$ until $f(x_0, \varphi_r^{-1}(P_r(s_a t_a))) = x_u$, which satisfies the first condition of Definition 4.2.

If $\mathcal{L}(G_a) = L_a^{UOE} \vDash$ **OEC4**, then, $\forall s_a \in L_a^{UOE}$ such that $f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u \in X_u$ but $f(x_0, P_a(s_a)) \neq x_u$, the last event in s_a cannot be some $\sigma_r \in \Sigma_r$, and hence, we can write $s_a = w_a v_a$, where $w_a \in \Sigma_a^* \Sigma_r$ and $f(x_0, \varphi_r^{-1}(P_r(w_a))) = x_u$ and $f(x_0, P_a(w_a)) = x_u$, which ensures the second condition of Definition 4.2. Notice that, if prefix w_a is such that $f(x_0, P_a(w_a)) \neq x_u$, it would violate Condition **OEC4**, since $w_a \in L_a^{UOE}$ and $w_a \in \Sigma_a^* \Sigma_r$, which contradicts our assumption. Therefore, G_a models an EU augmented system, which, together with the fact that both $\mathcal{L}(G_a) = L_a^{UOE} \neq \emptyset$ and $\mathcal{L}(G_a) \vDash$ (**OEC1** $\wedge$ **OEC2**), ultimately implies that G_a models an EU-CSO augmented system. ■

After obtaining an augmented language L_a^{UOE} that satisfies Conditions **OEC1**–**OEC4** simultaneously, we also require that the actions taken by the Opacity-Enforcer over augmented sequences $s_a \in L_a^{UOE}$ are unique, which prevents the undesired behaviors **UB2** and **UB3** from happening (conflicts of the Opacity-Enforcer either between release/delete observations and waiting for events to occur or between releasing and deleting event observations, respectively). To this end, we recall the priority order between releasing observations, deleting observations and waiting for events to occur, which was established in Section 3.3, as follows: (i) the release (resp. deletion) of a event observation has the highest (resp. lowest) priority among other actions; (ii) if there is no observation to be released and no held observation is being delayed for its maximum step delay bound, then the Opacity-Enforcer must take no action and wait for an event occurrence, and; (iii) the Opacity-Enforcer must either release at most one observation σ_r at a time, or delete at most one observation σ_d at a time. These priorities are formally expressed in the following unicity conditions:

UC1. $(\forall s_a \in L_a^{UOE})\big((\exists \sigma_r \in \Sigma_r : s_a\sigma_r \in L_a^{UOE}) \rightarrow s_a(\Sigma \cup \Sigma_d)\Sigma_a^* \cap L_a^{UOE} = \emptyset\big)$;

UC2. $(\forall s_a \in L_a^{UOE})\big((\exists \sigma \in \Sigma : s_a\sigma \in L_a^{UOE}) \rightarrow s_a\Sigma_d\Sigma_a^* \cap L_a^{UOE} = \emptyset\big)$;

UC3. $(\forall s_a \in L_a^{UOE})\big((|L_a^{UOE} \cap s_a\Sigma_r| \leq 1) \wedge (|L_a^{UOE} \cap s_a\Sigma_d| \leq 1)\big)$.

Example 4.4 *Let us revisit automaton H', shown in Figure 4.4 and presented in Example 4.3, where $X_s = \{3\}$, $X_u = \{4\}$, $\Sigma_D = \emptyset$, $SD(k) = \{(a,2),(b,0),(c,1)\}$ and whose generated language $\mathcal{L}(H') = L_a^{UOE}$ satisfies Conditions **OEC1–OEC4**. It is worth noticing that automaton H' models the Opacity-Enforcer behavior. However, after sequence $s_a = bb_rc$ has occurred, the Opacity-Enforcer cannot decide between* *waiting for the plant to generate event a and release c_r. Such problem is solved by Condition **UC1**, which prioritizes the observation release c_r, and thus, sequence bb_rca and all of its continuations are removed from $\mathcal{L}(H')$. Notice that, $P_a(\mathcal{L}(H')) = L$ is preserved after any sequence is removed due to Conditions **UC1–UC3**, once these conditions typically choose an unicity solution between sequence continuations whose projections model the same behavior of the plant.* □

4.3 Algorithms

In this section we propose an algorithm that computes automaton R_{UOE}, which models the desired Opacity-Enforcer behavior. Since the Opacity-Enforcer ensures utility while enforcing current-state opacity, we further show that the language generated by R_{UOE} is L_a^{UOE}.

The idea behind the algorithm is to first create an automaton G_a^{SD}, whose generated language is L_a^{SD}, and then, we recursively remove from it all sequences that violate some of the Opacity-Enforcer Conditions **OEC1–OEC4**, resulting in an automaton R_{UOE} whose generated language is L_a^{UOE} that models the desired Opacity-Enforcer behavior.

4.3.1 Computation of automaton G_a^{SD}

It is worth noting that the procedure to obtain language L_a^{SD} is responsible for synchronizing the plant behavior with the shuffling of every event observation release/deletion allowed by $SD(k)$. Thus, automaton G_a^{SD}, whose generated language is L_a^{SD}, is obtained by computing the parallel composition of the following three automata: (i) automaton G, which models the observed behavior of the plant; (ii) automaton D, which generates all possible shuffles of the event occurrences with observation releases/deletions according to the restrictions of a given $SD(k)$, and whose states denote observations of plant events to be released/deleted; (iii) automaton G_{int}, which models all event observation releases that the intruder is allowed to receive so as the secret is not revealed.

With that in mind, automaton $D = (X_D, \Sigma_a, f_D, \Gamma_D, x_{0,D})$ is computed as shown in Subsection 3.4.1. Automaton $G_{int} = Ac(X_{int}, \Sigma_r, f_{int}, \Gamma_{int}, x_{0,int})$ is obtained by taking the accessible part of G after all of its secret states have been removed and the subscript r has been added to all events that label its transitions, and thus, we set $X_{int} = X \setminus X_s$, $f_{int}(x, \varphi_r(\sigma)) = f(x, \sigma)$, if $f(x, \sigma) \in X \setminus X_s$, or undefined, otherwise, $\Gamma_{int}(x) = \{\sigma_r \in \Sigma_r : f_{int}(x, \sigma_r)!\}$, and $x_{0,int} = \{x_0\} \cap X_{int}$. Notice that $L_{int} = \mathcal{L}(G_{int}) = \{\varphi_r(s) : (s \in L) \wedge (\forall t \in \overline{s}, f(x_0, t) \in X \setminus X_s)\}$.

Notice that automata G, D, and G_{int} have no marked states initially, and so, their set of marked states have been omitted from their definitions. However, in order to simplify the algorithm proposed in Subsection 3.4.2, we must define the set of marked states X_m, $X_{m,D}$ and $X_{m,int}$ of automata G, D, and G_{int}, respectively, as follows.

- X_m is formed with all states of G that form some non-trivial SCC and all states that have no transition defined;
- $X_{m,D}$ is formed with the initial state of D, and thus, $X_{m,D} = x_{0,D}$;
- $X_{m,int}$ is formed with reminiscent marked states in $X_m \cap X_{int}$, after the secret states have been removed when computing G_{int}, that are accessible and either still form some non-trivial SCC or that have no transition defined.

Remark 4.2 *In order to avoid the case where there are states with no transitions defined when computing the marked states of automata G and G_{int}, we can proceed as follows. First, we define a new unobservable event σ_{uo} and add it to the event set Σ, and mark all states that form some non-trivial SCC of G. We then compute G_{int} as a copy of G but with no marked sates, remove all secret states of X_s from G_{int}, take its accessible part, add the subscript r to all events that label its transitions, and, finally, mark all states that form some non-trivial SCC of G_{int}. Since σ_{uo} is not an event generated by the plant, after marking states in both automata G and G_{int}, we remove σ_{uo} from their respective event set, and also all transitions labeled with it.* □

The idea behind marking states in the non-trivial SCC of automata G and G_{int} is that, since events are generated spontaneously, the plant will eventually reach either a state that forms some non-trivial SCC or a state with no transition defined. With respect to the set of marked states, it is worth noting that: (i) the set X_m ensures that $\mathcal{L}_m(G) = L_m$ is formed with both, all arbitrarily long length sequences of L and sequences with no continuations; (ii) by setting $X_{m,D} = X_{0,D}$, $\mathcal{L}_m(D)$ is composed of sequences s_a such that $\mathcal{N}\big(s_a, \sigma\big) = \mathcal{N}\big(s_a, \varphi_r(\sigma)\big) + \mathcal{N}\big(s_a, \varphi_d(\sigma)\big)$, for all $\sigma \in \Sigma$, meaning that every event of s_a in the marked language of D had its observation either released or deleted; (iii) since we remove secret states from G when computing G_{int}, some non-trivial SCCs (reached after the visit to some secret state) may have ceased to exist and states that once would inevitably lead the system to secret states are now not coaccessible, and thus, the set $X_{m,int}$ guarantees that $\mathcal{L}_m(G_{int})$ is formed with observation release sequences with either arbitrarily long length or no continuation that lead the intruder to a secret-free state estimation path.

Finally, we build automaton $G_a^{SD} = G \| D \| G_{int} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}, X_{m,a}^{SD})$ that, differently from that presented in Subsection 3.4.1, has marked states.

Example 4.5 *Let us consider automaton G shown in Figure 4.2, where $X_s = \{3\}$ and $X_u = \{4\}$. Since there are no non-trivial SCCs, and the only states with no transition defined are 3, 5, and 8, we have that $X_m = \{3, 5, 8\}$. Automaton $D = (X_D, \Sigma_a, f_D, \Gamma_D, x_{0,D}, X_{m,D})$, which models the observation releases/deletions in accordance with $SD(k) = \{(a, 2), (b, 0), (c, 1)\}$ and $\Sigma_D = \emptyset$, where $X_{m,D} = \{\nu\}$, has been computed according to Algorithm 3.1 and is depicted in Figure 4.5. Automaton $G_{int} = (X_{int}, \Sigma_r, f_{int}, \Gamma_{int}, x_{0,int}, X_{m,int})$, illustrated in Figure 4.6, has only two marked states, which form the set $X_{m,int} = \{5, 8\}$. Notice that state* 3*, that once was marked in G, does not belong to $X_{m,int}$, since it was removed when computing G_{int}. We emphasize that, it may happen that some state x, which once was marked in G ($x \in X_m$), was not removed when computing G_{int} ($x \in X_{int}$) but it ceased to form some non-trivial SCC, and therefore, it would not be marked in G_{int} ($x \notin X_{m,int}$). Finally, automaton $G_a^{SD} = G \| D \| G_{int}$ is shown in Figure 4.7.* □

Remark 4.3 *Not coaccessible states of automaton G_a^{SD} are reached by sequences s_a in L_a^{SD} that satisfy at least one of the following conditions: (i) $P_a(s_a)$ has a continuation with some event $\sigma \in \Sigma$ in L whereas $s_a(\Sigma_r \cup \Sigma_d)^* \sigma \notin L_a^{SD}$, which means that Condition **OEC2** has been violated (the event observation release/deletion policy of the Opacity-Enforcer is inhibiting the behavior of the plant); (ii) s_a is such that $\mathcal{N}(s_a, \sigma) > \mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma))$ for some $\sigma \in \Sigma$ but it has no continuation, which means that the Opacity-Enforcer is indefinitely holding the observation of some non deletable event, and thus, Condition **OEC1** has been violated; (iii) $P_r(s_a)$ has a continuation with some event $\sigma_r \in \Sigma_r$ in $\mathcal{L}(G_{int})$ whereas $s_a(\Sigma \cup \Sigma_d)^* \sigma_r \notin L_a^{SD}$, which means that the intruder is indefinitely waiting for some event that has been inhibited by the Opacity-Enforcer, and as a consequence, the intruder may become suspicious that the information released by the plant is being changed.* □

Finally, automaton G_a^{SD} models all shuffled sequences so as the released observations neither lead the intruder to estimate secret states nor lie outside the modeled behavior. Additionally, the sequences in the marked language $\mathcal{L}_m(G_a^{SD})$ are such that: (i) the plant sequences retrieved by the projection $P_a(\mathcal{L}_m(G_a^{SD}))$ denote that the system behavior has not been constrained; (ii) every event $\sigma \in \Sigma$ is certainly having its observation either released or deleted, and (iii) the order of event observation releases $\sigma_r \in \Sigma_r$ is such that the intruder is always misled to a secret-free state estimation path.

4.3.2 Computation of automaton R_{UOE}

We now present Algorithm 4.1, which computes automaton R_{UOE} (whose generated language is L_a^{UOE}) that models the desired behavior of the Opacity-Enforcer. Notice

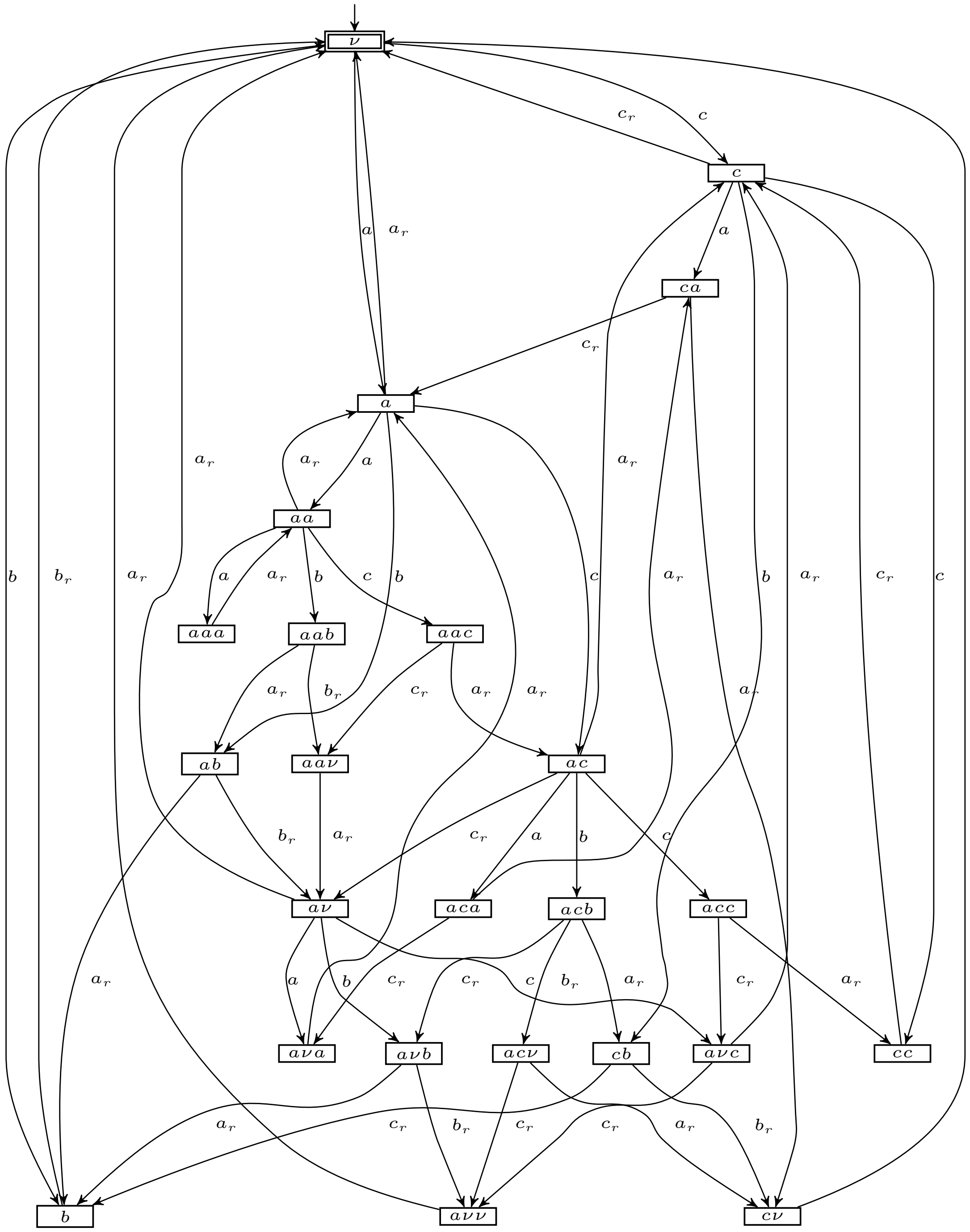


Figure 4.5: Automaton D.

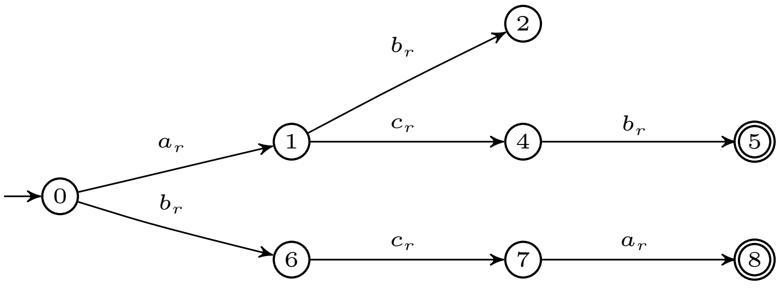


Figure 4.6: Automaton G_{int}.

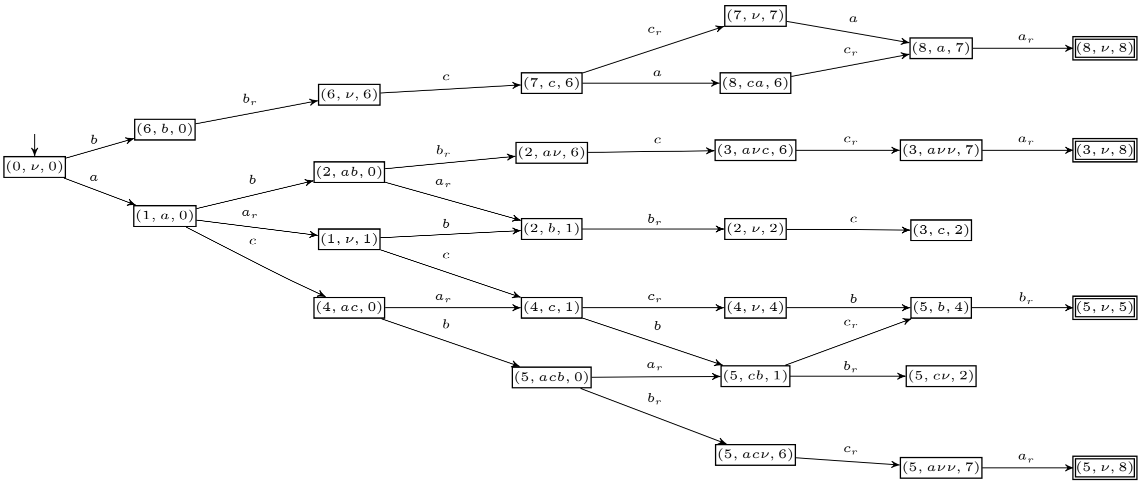


Figure 4.7: Automaton $G_a^{SD} = G \| D \| G_{int}$.

that, in order to obtain language L_a^{UOE} that ensures utility while current-state opacity is being enforced, Algorithm 4.1 recursively removes from automaton G_a^{SD}, which generates language L_a^{SD}, states and transitions whose associated sequences violate any Opacity-Enforcer Condition **OEC1–OEC4**. We recall functions $OEAR(x_a)$ and $NX(x_a)$, presented in Definitions 3.10 and 3.11 (see page 49), which outputs the set of states reached from x_a through events that denote the Opacity-Enforcer actions and the active plant events of states reached through $OEAR(x_a)$, respectively. Finally, with a view to guaranteeing the unicity of the actions taken by the Opacity-Enforcer, Algorithm 4.1 also removes from G_a^{SD} sequences that violate any of the Unicity Conditions **UC1 – UC3**. We assume with no harm that the input automaton G_a^{SD} is CSOE.

Algorithm 4.1 *Computation of automaton R_{UOE}.*

```
Input: $G_a^{SD} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}, X_{m,a}^{SD})$, $X_u$.
Output: $R_{UOE} = (X_{UOE}, \Sigma_a, f_{UOE}, \Gamma_{UOE}, x_{0,UOE}, X_{m,UOE})$.
  1. Set $V = G_a^{SD} = (X_V, \Sigma_a, f_V, \Gamma_V, x_{0,V}, X_{m,V})$.
  2. For each $x_V = (x, x_D, x_{int}) \in X_V$:
     2.1. If $x \in X_u$, $x_{int} \neq x$ and $\Gamma_V(x_V) \cap \Sigma \neq \emptyset$, then remove all $f_V(x_V, \sigma)$ such
          that $\sigma \in \Gamma_V(x_V) \cap \Sigma$, and its associated transitions.
     2.2. If $x_{int} \in X_u$, $x \neq x_{int}$ and there exists $(x'_V, \sigma_r) \in X_V \times \Sigma_r$ such that
          $f_V(x'_V, \sigma_r) = x_V$, then remove $f_V(x'_V, \sigma_r)$ and its associated transitions.
  3. Set $V = Trim(V)$.
  4. flag ← true.
  5. While flag = true, do:
     5.1. flag ← false.
     5.2. For each $x_V = (x, x_D, x_{int}) \in X_V$:
        5.2.1. If $(\Gamma(x) \cap \Gamma_D(x_D)) \setminus NX(x_V) \neq \emptyset$, then $X_V \leftarrow X_V \setminus \{x_V\}$,
               $V \leftarrow Trim(V)$, flag $\leftarrow$ true.
6. For each $x_V = (x, x_D, x_{int}) \in X_V$:
    6.1. If $\Gamma_V(x_V) \cap \Sigma_r \neq \emptyset$, then set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \Sigma_r$ and $V = Ac(V)$.
    6.2. If $\Gamma_V(x_V) \cap \Sigma \neq \emptyset$, then set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \Sigma$ and $V = Ac(V)$.
    6.3. If $|\Gamma_V(x_V) \cap (\Sigma_r \cup \Sigma_d)| > 1$, then:
        6.3.1. Denote $x_D = q_1 \ldots q_{\|x_D\|}$ and set $i = 1$.
        6.3.2. While $(\{\varphi_r(q_i), \varphi_d(q_i)\} \cap \Gamma_V(x_V) = \emptyset) \wedge (i \leq \|x_D\|)$, do $i = i + 1$.
        6.3.3. Set $\Gamma_V(x_V) = \Gamma_V(x_V) \cap \{\varphi_r(q_i), \varphi_d(q_i)\}$ and $V = Ac(V)$.
7. Return $R_{UOE} = V = (X_{UOE}, \Sigma_a, f_{UOE}, \Gamma_{UOE}, x_{0,UOE}, X_{m,UOE})$.
```

Algorithm 4.1 work as follows. In STEP 1, we create a copy V of automaton G_a^{SD}. In STEP 2, we are concerned with ensuring the utility of the system, where in STEP 2.1, if the plant reaches an useful state $x \in X_u$ but the legitimate receiver has not estimated it yet ($x_{int} \neq x$), then, with a view for the Opacity-Enforcer to releasing observations so as state x is estimated as soon as possible, we temporarily prevent the system to further evolve by removing transitions labeled with plant events $\sigma \in \Sigma$ from state x_V. On the other hand, in STEP 2.2, we remove from the Opacity-Enforcer behavior all event observations releases $\sigma_r \in \Sigma_r$ that lead the legitimate receiver to estimate some useful state $x_{int} \in X_u$ while the current state of the plant is not it, *i.e.*, $x \neq x_{int}$. STEPS 2.1 and 2.2 of Algorithm 4.1 ensure Conditions **OEC3** and **OEC4**, respectively.

In STEP 3 we make automaton V become trim and in STEP 4 we define a flag whose initial value is true, to be used in the recursion of STEP 5, where we are concerned with ensuring Conditions **OEC1** and **OEC2**. In STEP 5.2, if there exists some event $\sigma \in \Sigma$ allowed to occur in the plant (*i.e.*, $\sigma \in \Gamma(x)$) that does not violate $SD(k)$ (*i.e.*, $\sigma \in \Gamma_D(x_D)$) but it is not not allowed to occur in any of the states reached through actions taken by the Opacity-Enforcer $NX(x_V)$, meaning that such an event is being inhibited, which is an action that the Opacity-Enforcer cannot perform, then state x_V must be removed so as Condition **OEC2** is ensured. In addition, if states where $x_D \neq \nu$, which are not marked by construction, are not coaccessible, then the operation $Trim(G)$ performed in STEP 5.2 removes them from automaton V, which ensures that no event observation is being held forever (Condition **OEC1**). STEP 5 is repeated until the flag becomes false, which is only possible if no state in X_V needs to be deleted.

In STEP 6, we ensure the unicity of the actions taken by the Opacity-Enforcer as follows. In STEP 6.1, we ensure that an observation event is released $\sigma_r \in \Sigma_r$ whenever it is possible; secondly, in STEP 6.2, the Opacity-Enforcer waits for some event occurrence in the plant $\sigma \in \Sigma$, and; finally, it deletes an event observation $\sigma_d \in \Sigma_d$

only as last resource, hence, the least prioritized action. In addition, STEP 6.3 guarantees that the Opacity-Enforcer will either release or delete at most one observation at each state; thus, if there exists a state that has two or more allowed events to be released, then STEPS 6.3.1 and 6.3.2 find the event among them that occurred first and remove the others, which is performed in STEP 6.3.3. STEPS 6.1–6.3 implement Unicity Conditions **UC1–UC3**. Finally, STEP 7 computes the realization of Opacity-Enforcer automaton $R_{UOE} = V$ that ensures utility while opacity is being enforced.

Remark 4.4 (Computational complexity of Algorithm 4.1) *The* STEPS 1, 4 *and* 7 *of Algorithm 4.1 are constant with respect to the number of states of V whereas* STEPS 2, 3 *and* 6 *are linear. The computational burden of Algorithm 4.1 lies in* STEP 5, *where at least one state of automaton V is removed each time* STEP 5 *is repeated, and thus, $|X_V|$ states are analyzed in the fist time* STEP 5 *is executed, $|X_V| - 1$ states are analyzed in the second time, and this procedure continues until either automaton V meets the requirement for breaking the loop or it becomes empty. Therefore, in the worst case where V becomes empty, $(1 + |X_V|) \times |X_V|/2$ states are analyzed, which implies that the computational complexity of Algorithm 4.1 is $O(|X_V|^2)$. In addition, since (i) $V = G_a^{SD} = G\|D\|G_{int}$, (ii) $|X_{int}| \leq |X|$ and (iii) $X_D = 1 + \left[\sum_{j=0}^{k}(|\Sigma| + 1)^j\right] \times |\Sigma| \leq c \times |\Sigma|^{k+1}$ [6], where $k = max(|k_{\sigma_1}|, \ldots, |k_{\sigma_n}|)$ and c is a big enough natural number, we may conclude that V has at most $c \times |X|^2 \times |\Sigma|^{k+1}$ states, and then, we may conclude that Algorithm 3.2 is $O(|X|^4 \times |\Sigma|^{2k+2})$. Notice that, starting with the observed model requires a priori computation of the observer, and so, $|X|$ is $O(2^{|X_p|})$, where $|X_p|$ is the cardinality of the set of states of the plant. Thus, with respect to the plant model, the worst case computational complexity of our method becomes $O(2^{4|X_p|} \times |\Sigma|^{2k+2})$.* □

Since STEPS 2–5 of Algorithm 4.1 are responsible for ensuring Conditions **OEC1–OEC4** while building automaton R_{UOE}, we present the following result.

Lemma 4.2 *Let $G = (X, \Sigma, f, \Gamma, x_0)$ be the automaton that models the observable behavior of a system, X_u be the set of useful states, and let automaton $G_a^{SD} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}, X_{m,a}^{SD})$ model the augmented system with respect to the deletable events in Σ_D, secret states in X_s and step delay bounds in $SD(k)$. Then, the language L_a^{UOE} generated by automaton R_{UOE}, which is obtained in accordance with Algorithm 4.1, is either empty or satisfy Conditions **OEC1–OEC4** simultaneously.*

Proof.

We start by showing that STEP 2 of Algorithm 4.1 ensures both **OEC3** and **OEC4**. Assume that there exists some sequence $s_a \in L_a^{UOE}$ that violates

OEC3, which means that $(f(x_0, P_a(s_a)) = x_u \in X_u) \wedge (f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u) \wedge (s_a\Sigma\Sigma_a^* \cap L_a^{UOE} \neq \emptyset)$. Let $f_V(x_{0,V}, s_a) = x_V = (x, x_D, x_{int})$, then: (i) $x = x_u$, since $f(x_0, P_a(s_a)) = x_u$; (ii) $x_{int} \neq x_u$, since $x_{int} = f_{int}(x_{0,int}, P_r(s_a)) = f(x_0, \varphi_r^{-1}(P_r(s_a))) \neq x_u$, and; (iii) $\Gamma_V(x_V) \cap \Sigma \neq \emptyset$, since $s_a\Sigma\Sigma_a^* \cap L_a^{UOE} \neq \emptyset$. Thus, sequence s_a meets the conditions of STEP 2.1, which removes all of the transitions associated with $f_V(x_V, \sigma)$, $\sigma \in \Gamma_V(x_V) \cap \Sigma$, which results in the same as removing all continuations of s_a that start with some plant event $\sigma \in \Gamma_V(x_V) \cap \Sigma$. As a consequence, STEP 2.1 ensures Condition **OEC3**.

Assume now that there exists some sequence $s_a \in L_a^{UOE}$ that violates **OEC4**, which means that $(f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u \in X_u) \wedge (f(x_0, P_a(s_a)) \neq x_u) \wedge (s_a \cap \Sigma_a^*\Sigma_r \neq \emptyset)$. Analogously, assume that $f_V(x_{0,V}, s_a) = x_V = (x, x_D, x_{int})$, which implies that: (i) $x_{int} = x_u$, since $x_{int} = f_{int}(x_{0,int}, P_r(s_a)) = f(x_0, \varphi_r^{-1}(P_r(s_a))) = x_u$; (ii) $x \neq x_u$, since $f(x_0, P_a(s_a)) \neq x_u$, and; (iii) the last event of s_a is an observation release, since $s_a \cap \Sigma_a^*\Sigma_r \neq \emptyset$, and so there exists $s_a' \in \Sigma_a^*$ and $\sigma_r \in \Sigma_r$ such that $s_a = s_a'\sigma_r$, which implies that there exists a state $x_V' = f_V(x_{0,v}, s_a')$ in automaton V and a transition associated with $f_V(x_V', \sigma_r) = x_V$. Since the conditions of STEP 2.2 are met, the transitions associated with $f_V(x_V', \sigma_r) = x_V$ are removed, which means that sequences s_a that violate Conditions **OEC4** are removed from L_a^{UOE}.

Condition **OEC1** is satisfied when every sequence s_a of L_a^{UOE} that has no continuation $(L_a^{UOE}/s_a = \emptyset)$ is such that the observation of each plant event $\sigma \in \Sigma$ that forms it is either released or deleted $(\mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma)) = \mathcal{N}(s_a, \sigma), \forall \sigma \in \Sigma)$. We recall from Inequality (3.1) that the number of occurrences of an event $\sigma \in \Sigma$ in a sequence $s_a \in \Sigma_a^*$ must always be greater than or equal to the sum of the observation releases and deletions of σ, which is guaranteed by construction according to Definitions 3.5 and 3.7. Assume that there exists a sequence $s_a \in L_a^{UOE}$ with no continuation and formed with some plant events whose observation has neither been released nor deleted. Let $x_V = (x, x_D, x_{int})$ be the state of automaton V reached by sequence s_a, which implies that $f_V(x_{0,V}, s_a) = x_V$ and $\Gamma_V(x_V) = \emptyset$. Since there exist plant events $\sigma \in \Sigma$ in s_a such that $\mathcal{N}(s_a, \sigma) > \mathcal{N}(s_a, \varphi_r(\sigma)) + \mathcal{N}(s_a, \varphi_d(\sigma))$, the second component of x_V is not "*blank*", *i.e.*, $x_D \neq x_{0,D} = \nu$, and as a consequence, x_V is not marked $(X_{m,V} = X_m \times \{x_{0,D}\} \times X_{m,int})$. Finally, since $\Gamma_V(x_V) = \emptyset$ and $x_V \notin X_{m,V}$, state x_V is not coaccessible, and therefore, it is removed by either STEP 3 or STEP 5.2.1 when computing $Trim(V)$, which results in sequence s_a being removed from L_a^{UOE} and Condition **OEC1** being satisfied.

Finally, assume that there exists some sequence $s_a \in L_a^{UOE}$ that violates **OEC2**, which means that $\exists \sigma \in \Sigma : P_a(s_a)\sigma \in L$ but $P_a(s_a t_a) \neq P_a(s_a)\sigma, \forall t_a \in L_a^{OE}/s_a$. Let $x_V = (x, x_D, x_{int})$ be the state of automaton V reached by sequence s_a. Thus, $OEAR(x_V)$ is a set formed with all states reached from x_V through sequences in $(\Sigma_r \cup \Sigma_d)^*$, which implies that $\forall x_V' \in OEAR(x_V)$, $\exists s_a' \in (\Sigma_r \cup \Sigma_d)^*$ such that

$f_V(x_V, s'_a) = x'_V$. Notice that those states $x'_V \in OEAR(x_V)$ are such that $P_a(s_a s'_a) = P_a(s_a)$, and, in addition, those sequences $s_a s'_a$ cannot be continued with that event $\sigma \in \Sigma$, otherwise we would have $t_a = s'_a\sigma \in L_a^{OE}/s_a$ and $P_a(s_a t_a) = P_a(s_a s'_a \sigma) = P_a(s_a)\sigma$, which contradicts our initial assumption. Therefore, $\sigma \notin NX(x_V)$. On the other hand, since $P_a(s_a)\sigma \in L$ and $f(x_0, P_a(s_a)) = x$, we conclude that $\sigma \in \Gamma(x)$. It remains for us to verify if $\sigma \in \Gamma_D(x_D)$. On the one hand, if all plant events that form x_D have not reached their step delay bound, then $\sigma \in \Gamma_D(x_D)$, which means that $\sigma \in \Gamma(x) \cap \Gamma_D(x_D) \setminus NX(x_V)$ and, as a consequence, state x_V is removed from V by Step 5.2.1, which means that sequence s_a is removed from L_a^{UOE} and Condition **OEC2** is satisfied. On the other hand, if some plant event that form x_D has reached its step delay bound, then $\Sigma \cap \Gamma_D(x_D) = \emptyset$, however, since $s_a \vDash$ **OEC1**, $\exists \sigma_r \in \Sigma_r : s_a\sigma_r \in L_a^{UOE}$ and then, we can define $\hat{s}_a = s_a\sigma_r$ and repeat this analysis for $\hat{s}_a$, since $P_a(\hat{s}_a)\sigma \in L$ but $P_a(\hat{s}_a\hat{t}_a) \neq P_a(\hat{s}_a)\sigma, \forall \hat{t}_a \in L_a^{OE}/\hat{s}_a$. Notice that, the second component $\hat{x}_D$ of state $\hat{x}_V = (\hat{x}, \hat{x}_D, \hat{x}_{int})$ reached through sequence $\hat{s}_a$ is such that $\Sigma \subset \Gamma_D(\hat{x}_D)$, which lies in the aforementioned case where $\sigma \in \Gamma_D(\hat{x}_D)$, and thus, state $\hat{x}_V$ (which immediately follows x_V through a transition labeled with event σ_r) is removed by Step 5.2.1. It is worth highlighting that the state x_V may become not coaccessible after state $\hat{x}_V$ has been removed. In this case, x_V is also deleted by Step 5.2.1 when it performs the trim operation. Therefore, Step 5.2.1 ensures that every sequence s_a that violates Condition **OEC2** is removed from L_a^{UOE}.

Since Step 5 is a recursion, either all of the remaining sequences in L_a^{UOE} satisfy Conditions **OEC1–OEC4**, allowing Algorithm 4.1 to proceed to Steps 6 and 7, which returns automaton R_{UOE} whose generated language is L_a^{UOE}, or Step 5 removes all sequences from L_a^{UOE}, and thus, automaton R_{UOE} becomes empty. ■

We then present the main result of this chapter, which provides necessary and sufficient conditions to verify if the automaton R_{UOE}, obtained according to Algorithm 4.1, models an EU-CSO augmented system, as follows.

Theorem 4.1 *Let $G = (X, \Sigma, f, \Gamma, x_0)$ be the automaton that models the observable behavior of a system, X_u be the set of useful states, and let automaton $G_a^{SD} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}, X_{m,a}^{SD})$ model the augmented system with respect to the deletable events in Σ_D, secret states in X_s and step delay bounds in $SD(k)$. Consider automaton R_{UOE}, which is obtained in accordance with Algorithm 4.1, and whose generated language is L_a^{UOE}. Automaton R_{UOE} models an EU-CSO augmented system if, and only if, $L_a^{UOE} \neq \emptyset$.*

Proof.

($\Rightarrow$) Assume that automaton R_{UOE} models an EU-CSO augmented system, which, together with Definition 4.3, implies that $\mathcal{L}(R_{UOE}) = L_a^{UOE} \neq \emptyset$.

($\Leftarrow$) Assume that the language generated by automaton R_{UOE} is not empty, *i.e.*, $L_a^{UOE} \neq \emptyset$. According to Lemma 4.2, since $L_a^{UOE} \neq \emptyset$, we have that $L_a^{UOE} \models$ (**OEC1**$\wedge$**OEC2**$\wedge$**OEC3**$\wedge$**OEC4**). Finally, according to Lemma 4.1, the augmented system modeled by automaton R_{UOE} is EU-CSO. ■

4.4 Example

We revisit the system presented in Example 4.1, whose observed behavior has been modeled by automaton $G = (X, \Sigma, f, \Gamma, x_0)$, depicted in Figure 4.2. Let $X_s = \{3\}$ and $X_u = \{4\}$ denote the sets of secret and useful states, respectively, and let $SD(k) = \{(a, 2), (b, 0), (c, 1)\}$ denote the set formed with the step delay bounds for each one of the events of Σ, where no event is deletable, *i.e.*, $\Sigma_D = \emptyset$.

In order to illustrate Algorithm 4.1, we must first build automaton $G_a^{SD} = G \| D \| G_{int} = (X_a^{SD}, \Sigma_a, f_{int}, \Gamma_a^{SD}, x_{0,a}^{SD}, X_{m,a}^{SD})$. To this end, as detailed in Example 4.5: (i) we mark states 3, 5 and 8 of automaton G; (ii) we compute automaton $D = (X_D, \Sigma_a, f_D, \Gamma_D, x_{0,D}, X_{m,D})$ that models $SD(k) = \{(a, 2), (b, 0), (c, 1)\}$ and $\Sigma_D = \emptyset$ in accordance with Algorithm 3.1 and mark its initial state, *i.e.*, $X_{m,D} = \{\nu\}$; (iii) we build automaton $G_{int} = (X_{int}, \Sigma_r, f_{int}, \Gamma_{int}, x_{0,int}, X_{m,int})$ and mark states 5 and 8, *i.e.*, $X_{m,int} = \{5, 8\}$. Figures 4.5 and 4.6 depict automata D and G_{int}, respectively. Finally, we compute automaton $G_a^{SD} = G \| D \| G_{int}$, shown in Figure 4.7.

We now start explaining Algorithm 4.1. After V has been computed as a copy of automata G_a^{SD} by Step 1, according to Step 2, we search for transitions that violate either **OEC3** or **OEC4**. In this regard, in Step 2.1, we remove the transition from state $(4, ac, 0)$ to state $(5, acb, 0)$ labeled with event b, and transition from state $(4, c, 1)$ to state $(5, cb, 1)$ labeled with event b, since in both states $(4, ac, 0)$ and $(4, c, 1)$, the plant is currently at the utility state 4 but these transitions labeled with event b would make the system to further evolve without the legitimate receiver estimating the useful state 4, which violates specification **US1** and, consequently, **OEC3**. in Step 2.2, we remove only the transition from state $(5, cb, 1)$ to state $(5, b, 4)$, which is labeled with event c_r, since it means that the release of the observation of event c (namely event c_r) has just made the legitimate receiver estimate the useful state 4 while the current state of the plant is different, which violates specification **US2** and, consequently, **OEC4**. Notice that Steps 2.1 and 2.2 only need to be performed once, since the removal of transitions do not make other states to violate Conditions **OEC3** or **OEC4**. Figure 4.8 shows automaton V throughout Step 2, where the red (resp. blue) colorized transitions violate Condition **OEC3** (resp. **OEC4**) and are removed from V by Step 2.1 (resp. Step 2.2).

In Step 3, states that have become either not accessible or not coaccessible, as mentioned in 4.3, are removed from V when we compute $Trim(V)$. For example,

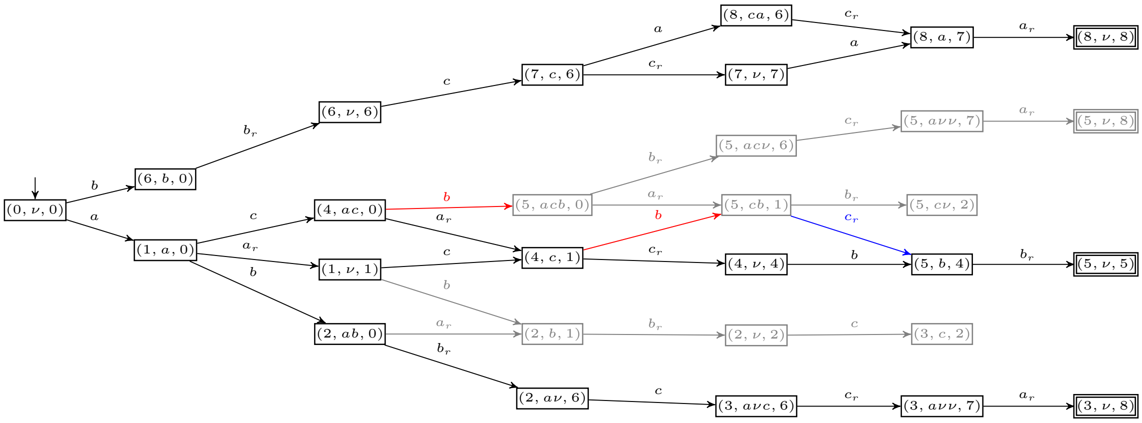


Figure 4.8: Automaton V with colorized transitions that violate STEP 2 of Algorithm 4.1.

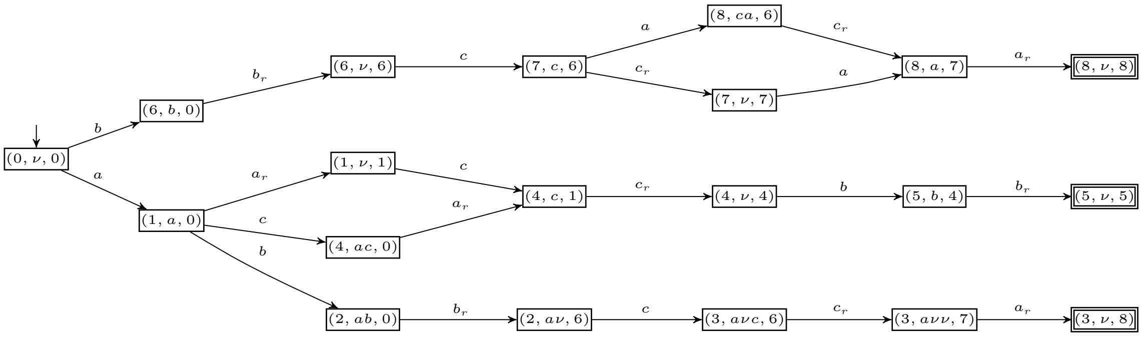


Figure 4.9: Automaton V after STEP 3 of Algorithm 4.1.

both states $(5, c\nu, 2)$ and $(3, c, 2)$ violate **OEC1**, since they have no continuation but their second component means that event observations are being held (they are different from ν), and so, these states are not coaccessible, which implies that they are removed from automaton V when STEP 3 is performed by Algorithm 4.1. On the other hand, state $(5, \nu, 8)$ is coaccessible but became not accessible, due to the transitions removal performed in STEP 2, and as a consequence, it will be removed from V by STEP 3. Those states and transitions that have been removed by STEP 3 are shown in Figure 4.8, where they have been colorized with gray. The resulting automaton V after STEP 3 is illustrated in Figure 4.9.

In the first time STEP 5 is performed by Algorithm 4.1, only state $(1, \nu, 1)$ meets the conditions of STEP 5, since $b \in \Gamma(1)$, $b \in \Gamma_D(\nu)$, but $b \notin NX(\{(1, \nu, 1)\}) = \{c\}$, which means that the following behavior is not modeled in V: event a is generated by the plant, the Opacity-Enforcer immediately releases its observation a_r and event b occurs in the plant. We recall that the Opacity-Enforcer must never inhibit plant event occurrences. Therefore, all sequences that reach state $(1, \nu, 1)$, namely aa_r, actually violate **OEC2**, and thus, should be removed. Next, we compute $Trim(V)$, and now no state violates the condition of STEP 5.2.1, which sets the value of the flag as "*false*" and allows Algorithm 4.1 to proceed to STEP 6.

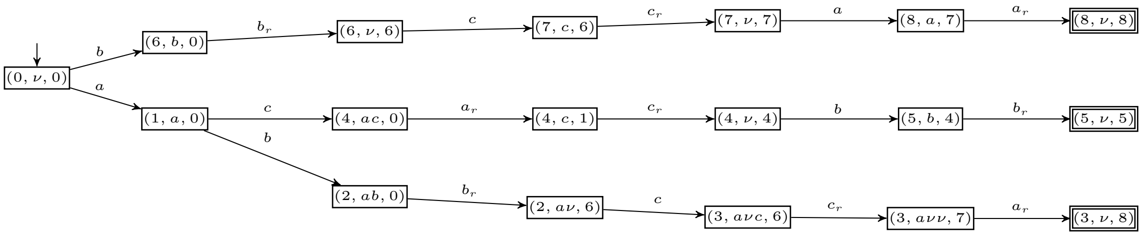


Figure 4.10: Automaton R_{UOE}, obtained in accordance with Algorithm 4.1.

In Step 6, we verify if unicity conditions **UC1–UC3** are being violated. In this regard, the unique state that violates some unicity condition is state $(7, c, 6)$, where the Opacity-Enforcer has to decide between releasing the event observation c_r and waiting for event a to occur. According to **UC1**, event observation releases have the highest priority among the actions that the Opacity-Enforcer is allowed to take. Therefore, in order to ensure **UC1**, we must remove the transition from state $(7, c, 6)$ to state $(8, ca, 6)$ labeled with event a, which is performed in Step 6.1.

Finally, in Step 7, we compute automaton R_{UOE}, which is the resulting automaton after the states/transitions removals performed in Algorithm 4.1. Figure 4.10 depicts automaton R_{UOE}, which models the desired behavior of an Opacity-Enforcer that ensures the utility of the system and, at the same time, enforces current-state opacity. It is worth highlighting that, the Opacity-Enforcer automaton R_{UOE} does not allow the intruder to estimate the secret states in $X_s = \{3\}$, since no state has its third component being a secret state, and even when the current state of the system is a secret one, *e.g.*, states $(3, a\nu c, 6)$ and $(3, a\nu\nu, 7)$, the intruder cannot infer it and estimates a non-secret state instead (states 6 and 7, respectively). Moreover, when the system reaches, in this case, the unique useful state $x_u = 4$, *e.g.*, in state $(4, ac, 1)$, the Opacity-Enforcer automaton R_{UOE} starts releasing/deleting event observations so as the legitimate receiver estimates it as soon as possible, which can be noticed from Figure 4.10, where state $(4, ac, 1)$ has a unique continuation with sequence $a_r c_r$, which leads the legitimate receiver to estimate the useful state in $(4, \nu, 4)$. Notice also that, when the estimates of the legitimate receiver is updated to a useful state, *e.g.*, state $(4, \nu, 4)$, the current state of the system corresponds to it (the first and third components of the state must be identical).

4.5 Concluding remarks

In this chapter, we have addressed the main criticism regarding opacity enforcement strategies, namely that in order to obfuscate the secret behavior of the system from intruders, some transmitted information must also be concealed from the legitimate receiver. In this regard, we have introduced the notion of (state-based) utility of the system, which differs from that approached in [71]. Moreover, we have presented sufficient and necessary conditions that guarantee that the utility of the system is preserved while current-state opacity is being enforced.

In addition, we have also improved the algorithm presented in Chapter 3 for realizing the CSO enforcer. The new algorithm that we have presented in this chapter realizes an Opacity-Enforcer that ensures the utility of the system and, at the same time, enforces current-state opacity, whenever the behavior of the system allows.

Chapter 5

Fault prediction

This chapter presents the second problem that has been solved by using state estimations. This time, we investigate the problem of disjunctive predictability verification and online fault prediction of DES in less restrictive scenarios, *i.e.*, without making the usual assumptions of language liveness and absence of cycles of states connected by unobservable events. To this end, we adapt the test automaton and the verifier, proposed in [102] and [103], respectively, which are used to verify codiagnosability, in order to develop two new strategies to verify disjunctive predictability. The first one is based on the test automaton proposed in [102], whereas the the second one relies on the verifier proposed in [104, 105]. We also introduce the notion of K-copredictability and the problem of its verification.

We have structured this chapter as follows. Section 5.1 presents the problem formulation, recall the assumptions usually made on previous works on decentralized architectures, the formal definition of disjunctive predictability (to be referred here simply as copredictability), and some illustrative examples; Section 5.2 considers the problem of copredictability verification using a diagnoser-like test automaton and a verifier, respectively in Subsections 5.2.1 and 5.2.2; Section 5.3 presents the design of fault predictor systems; Section 5.4 approaches the problem of K-copredictability, presenting an algorithm for its verification; finally, Section 5.6 summarizes all contributions of this chapter. Preliminary results of this chapter have been presented in [101], whereas its full version has been accepted for publication in Automatica.

5.1 Problem formulation

We consider the decentralized architecture depicted in Figure 5.1, which is composed of a plant modeled by automaton G, m Measurement Sites (MS), n Local Predictors (LP), at most $n \times m$ communication channels (ch), and a Coordinator. Each measurement site MS_j, $j \in \{1, \ldots, m\}$ is responsible for recording the occurrence of events $\sigma \in \Sigma_{ms,j}$, and each event is recorded by a specific measurement site,

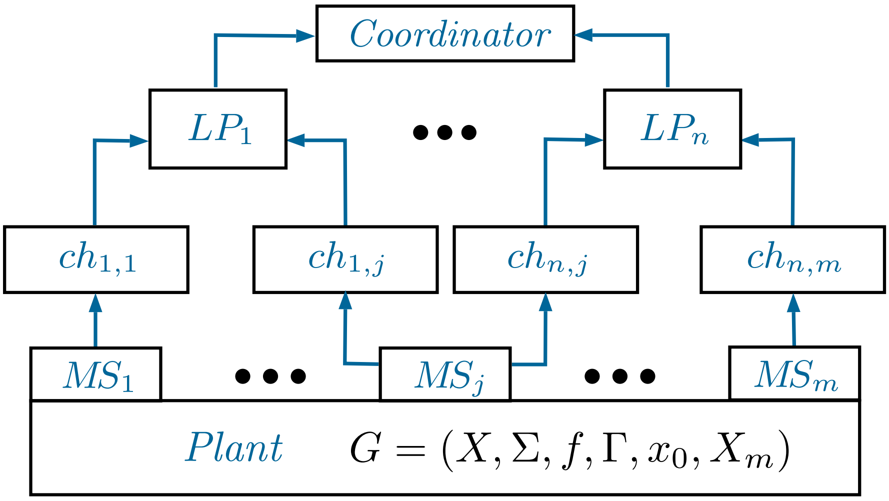


Figure 5.1: Decentralized architecture.

therefore, $\Sigma_{ms,i} \cap \Sigma_{ms,j} = \emptyset, \forall i \neq j$. The communication channel $ch_{i,j}$ is responsible for transmitting the events recorded by measurement site MS_j to local predictor LP_i; and therefore, the transmitted events form the set $\Sigma_{o_{i,j}}$. If a channel $ch_{i,j}$ does not exist, then $\Sigma_{o_{i,j}} = \emptyset$. The local predictors LP_i, $i \in \{1, \dots, n\}$ are responsible for issuing a verdict if a fault event will inevitably occur in a finite number of subsequent event occurrences, thus foreseeing the fault occurrence. The events observed by a local predictor LP_i form the event set $\Sigma_{o,i} = \bigcup_{j=1}^{m} \Sigma_{o_{i,j}}$. The Coordinator manages the information received from the local predictors, in the sense that, it issues a verdict that a fault will eventually happen if at least one local predictor LP_i foresees the fault occurrence. This is referred to as disjunctive predictability, or simply copredictability.

We then assume that: (i) the measurement sites never miss event occurrences, i.e., whenever an event is generated by the system, the corresponding measurement site records its occurrence, and that (ii) there are neither packet loss nor delays in the communication channels, i.e., each event generated by the system automaton always reaches some local predictor with no changes in the observation order.

Notice that, events generated by the plant that are recorded by some measurement site MS_i, $i \in \{1, \dots, m\}$ are the so-called observable events, and thus, the set of observable events is $\Sigma_o = \bigcup_{j=1}^{m} \Sigma_{MS,j} = \bigcup_{i=1}^{n} \Sigma_{o,i}$. On the other hand, events that are generated by the plant but not recorded by any measurement site are considered unobservable, thus forming set Σ_{uo}. Observable events that are not observable by some local predictor are referred to as locally unobservable, being denoted by $\Sigma_{uo,i} = \Sigma \setminus \Sigma_{o,i}$.

Finally, since we intend to approach more general classes of DES, the usual assumptions on non existence of cycle of states connected by unobservable events only and language liveness are not required here. The discarded old assumptions are the following.

OA1. There is no cycle of states connected by unobservable events only.

OA2. The language generated by the system is live.

Remark 5.1 *Notice that several techniques for predictability verification in the literature fail to produce the correct verification answer when Old Assumptions* ***OA1*** *and* ***OA2*** *are violated (see [82]) since they hide all cycles formed with unobservable events only, the so called-hidden cycles [3, 108]. In addition, the removal of Assumption* ***OA1*** *immediately releases Assumption* ***OA2*** *since, from the observation point of view, every non-live language can become live by adding self-loops labeled by unobservable events to every state with undefined transition function. Thus, we will keep the usual definitions of predictability assuming that the language is live.*

Although all definitions are stated for live languages, if the language is not live, it can be made live, from the observation point of view, by concatenating all sequences that do not have continuations with σ_{uo}^*, where σ_{uo} is an unobservable event.

The dynamic evolution of the architecture presented here is as follows: (i) the system automaton generates events; (ii) the observable events are recorded by some Measurement Site MS_j; (iii) the Communication Channel $ch_{i,j}$ transmits the observation to the corresponding Local Predictor LP_i; (iv) based on its own observation model, each Local Predictor LP_i sends the information regarding the predictability of some fault event to the Coordinator; (v) the Coordinator issues a verdict that a fault will inevitably occur if at least one Local Predictor LP_i has predicted it (disjunctive fault prediction). We recall that each Local Predictor LP_i observes events in $\Sigma_{o,i}$ only.

Definition 5.1 *(Disjunctive Predictability/Copredictability) A prefix-closed and live language L is copredictable with respect to $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i \in I = \{1, 2, \ldots, n\}$, and an event set $\Sigma_f = \{\sigma_f\}$ if*

$$\big(\exists z \in \mathbb{N}\big)\big(\forall s \in \Psi(\Sigma_f)\big)\big(\exists t \in \overline{s}\big)\big[(\Sigma_f \notin t) \wedge \boldsymbol{P}\ \big]$$

where $\boldsymbol{P} : \big(\exists i \in I\big)\big(\forall u_i \in L\big)\big(\forall v_i \in L/u_i\big)\big[(P_{o,i}(u_i) = P_{o,i}(t)) \wedge (\Sigma_f \notin u_i) \wedge (\|v_i\| \geq z) \Rightarrow (\Sigma_f \in v_i)\big]$ □

It is clear that Definition 2.12 is a particular case of Definition 5.1 when $I = \{1\}$ and $\Sigma_p = \Sigma_f = \{\sigma_f\}$.

Example 5.1 *Consider automaton H depicted in Figure 5.2, whose generated language is L, its event set is $\Sigma = \{a, b, c, \sigma_f\}$, and fault event set is $\Sigma_f = \{\sigma_f\}$.*

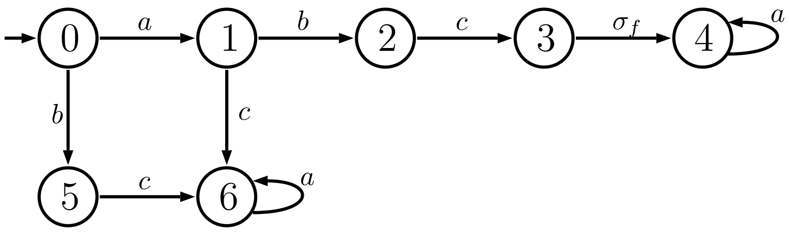


Figure 5.2: Automaton H.

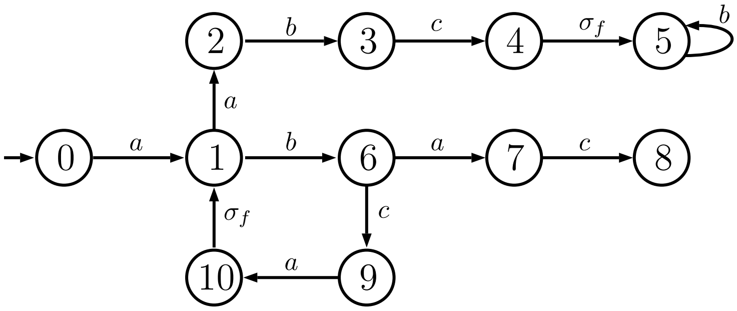


Figure 5.3: Automaton G.

Let us check if L is copredictable with respect to $\Sigma_{o,1} = \{b, c\}$ and $\Sigma_{o,2} = \{a, c\}$. Since, the projection $P_{o,1}$ of all prefixes of $s_f = abc\sigma_f$ ($P_{o,1}(s_f) = bc$) is identical to the projection of some prefix of the arbitrarily long normal sequence $u_1v_1 \in bca^*$ ($P_{o,1}(u_1v_1) = bc$), the language L is not predictable with respect to $P_{o,1}$ and Σ_f. The same conclusion can be drawn when we consider $P_{o,2}$, since for all prefixes $t_f \in \overline{s_f}$, there is a sequence $u_2 \in \overline{aca^*}$ such that $P_{o,2}(t_f) = P_{o,2}(u_2) \in \overline{ac}$. Therefore, language L generated by the system is not copredictable with respect to $P_{o,i}$, $i = 1, 2$, and Σ_f, since none of the local predictors can be sure that the fault will inevitably occur when sequence $s_f = abc\sigma_f$ occurs in the system. □

Remark 5.2 *It is not difficult to show that a non predictable language with respect to P_o and Σ_f will not be copredictable with respect to $P_{o,i}$ and Σ_f, $\forall \Sigma_{o,i} \subseteq \Sigma_o$. This conclusion can be drawn from the fact that a non predictable language has at least one fault sequence $s \in \Psi(\Sigma_f)$ such that the single centralized predictor observing Σ_o cannot predict its occurrence, therefore, no local predictor LP_i that observes $\Sigma_{o,i} \subseteq \Sigma_o$ will be able to predict the fault occurrence in s, hence, the language will be not copredictable with respect to any $P_{o,i}$, $i = 1, 2, \ldots, n$ and Σ_f.* □

Example 5.2 *Consider automaton G depicted in Figure 5.3 where $\Sigma = \{a, b, c, \sigma_f\}$, $\Sigma_f = \{\sigma_f\}$ and $\Sigma_o = \{a, b, c\}$. Let L_{nl} denote the language generated by G. Notice that L_{nl} is not live, and so, as suggested in Remark 5.1, we may set $\Sigma = \Sigma \cup \{\sigma_{uo}\}$, where σ_{uo} is an unobservable event, and then add transition $f(8, \sigma_{uo}) = 8$ to automaton G, so as to obtain a language L that is live and its observed behavior is* *the same as L_{nl}. It is not difficult to check that language L is predictable with respect to P_o and Σ_f, i.e., it is possible to identify prefixes of fault sequences $s_1 = aabc\sigma_f$ and $s_2 = abca\sigma_f$ whose observations are not identical to the observation of all prefixes of arbitrarily long normal sequences.*

Assume that the system is part of a disjunctive decentralized architecture with two local predictors, LP_1 and LP_2, which are able to observe the events of sets $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$, respectively. It is clear that both predictors, LP_1 and LP_2, when working alone, are unable to foresee all fault occurrences. In order to prove this claim, let us consider sequence $s_1 = aabc\sigma_f$. Notice that prefixes $t_1 = aabc$ and $u = abac$ of s_1 and of arbitrarily long normal sequence $s_N = abac\sigma_{uo}^k$, $k \in \mathbb{N}$, respectively, are such that $P_{o,1}(t_1) = P_{o,1}(u) = aac$ and $\Sigma_f \notin L/u = \sigma_{uo}^k$, which shows that LP_1 cannot predict the occurrence of fault sequence s_1. Consider now sequence $s_2 = abca\sigma_f$. It is not hard to see that its longest non-faulty prefix $t_2 = abca$ is such that $P_{o,2}(t_2) = P_{o,2}(u) = aba$, where $u \in \overline{s_N}$, and as a consequence, there exists a fault occurrence that LP_2 cannot predict. However, when both LP_1 and LP_2 work under a disjunctive coordination, every fault occurrence in the system is foreseen, since when s_1 (resp. s_2) occurs in the system, although LP_1 (resp. LP_2) is not sure about the future fault occurrence, local predictor LP_2 (resp. LP_1) is sure that it will inevitably occur. Thus, language L is copredictable with respect to $P_{o,i}$, $i = 1, 2$, and Σ_f. □

Lemma 5.1 *Let L_{nl} be a not live language and L be a live language obtained from L_{nl} by adding $s\sigma_{uo}^*$ into L for all sequences $s \in L_{nl}$ that have no continuation, i.e., $L = L_{nl} \cup \{s\sigma_{uo}^* : s \in L_{nl} \wedge s\Sigma \notin L_{nl}\}$. Then, L is copredictable with respect to $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i \in I = \{1, 2, \ldots, n\}$, and event set $\Sigma_f = \{\sigma_f\}$ if, and only if, L_{nl} is copredictable with respect to $P_{o,i}$, $i \in I = \{1, 2, \ldots, n\}$, and event set $\Sigma_f = \{\sigma_f\}$.*

Proof.

($\Rightarrow$) Assume that L is copredictable with respect to $P_{o,i}$, $i = 1, \ldots, n$, and Σ_f. Then, for all fault sequences $s_f \in \Psi(\sigma_f)$, there exists a prefix with no fault $t_f \in \overline{s_f}, \sigma_f \notin t_f$ such that, for a local $i = 1, \ldots, n$, all sequences $u_i \in L, \sigma_f \notin u_i$ such that $P_{o,i}(u_i) = P_{o,i}(t_f)$ will certainly have all arbitrarily long length continuations with fault, *i.e.*, $\forall v_i \in L/u_i, \|v_i\| \geq z \in \mathbb{N}$ is such that $\sigma_f \in v_i$. This means that, from the observation perspective of local i, all prefixes $u_{N,i}$ of arbitrarily long length normal sequences $s_{N,i}$ cannot be confused with that prefix t_f, *i.e.*, $P_{o,i}(t_f) \notin \overline{P_{o,i}(s_{N,i})}$. Assume now that sequence $s_{N,i}$ have been made arbitrarily long by adding events σ_{uo} to it. Then, we can write $s_{N,i} = s'_{N,i}\sigma_{uo}^*$. However, it still holds true that $P_{o,i}(t_f) \neq P_{o,i}(u'_{N,i})$, where $u'_{N,i} \in \overline{s'_{N,i}}$, since $P_{o,i}(u'_{N,i}) \in \overline{P_{o,i}(s'_{N,i})} = \overline{P_{o,i}(s'_{N,i}\sigma_{uo}^*)} = \overline{P_{o,i}(s_{N,i})}$, event σ_{uo} is unobservable and $P_{o,i}(t_f) \notin \overline{P_{o,i}(s_{N,i})}$. This result means that any sequence s_f that can be predicted in L can also be predicted in L_{nl}, which implies, according to our initial assumption, that L_{nl} is copredictable with respect to $P_{o,i}$, $i = 1, \dots, n$, and Σ_f

($\Leftarrow$) Assume now that L is not copredictable with respect to $P_{o,i}$, $i = 1, \dots, n$, and Σ_f. Then, there exists a fault sequence $s_f \in \Psi(\sigma_f)$, whose all prefixes with no fault $t_f \in \overline{s_f}, \sigma_f \notin t_f$ are confused with some prefix $u_{N,i}$ of arbitrarily long length normal sequences $s_{N,i}$, *i.e.*, $u_{N,i} \in \overline{s_{N,i}}$, from the perspective of any local $i = 1, \dots, n$. Therefore, $P_{o,i}(t_f) = P_{o,i}(u_{N,i}) \in \overline{P_{o,i}(s_{N,i})}$. If it is the case where sequence $s_{N,i}$ has been made arbitrarily long by adding events σ_{uo} to it, then we can write $s_{N,i} = s'_{N,i}\sigma^*_{uo}$. Since event σ_{uo} is unobservable, then $P_{o,i}(s_{N,i}) = P_{o,i}(s'_{N,i}\sigma^*_{uo}) = P_{o,i}(s'_{N,i})$, which implies that $P_{o,i}(t_f) \in \overline{P_{o,i}(s'_{N,i})}$, meaning that the fault in sequence s_f cannot be predicted in L_{nl} either. Thus, L_{nl} is not copredictable with respect to $P_{o,i}$, $i = 1, \dots, n$, and Σ_f. ■

5.2 Copredictability verification

In this section, we address the problem of copredictability verification for regular languages and, to this end, we propose two different verification strategies: *(i)* the first one that deploys the test automaton proposed by VIANA and BASILIO [102]; *(ii)* the second strategy that is based on the verifier proposed by MOREIRA *et al.* [104] (see also MOREIRA *et al.* [105]).

5.2.1 Copredictability verification using the diagnoser-like test automaton

The test automaton

As pointed out in Remark 5.1, the information about state cycles connected solely by unobservable events is lost when diagnoser $G_{d,i}$ is built, since $G_{d,i} = \text{Obs}(G_\ell, \Sigma_{o,i})$. The lack of such information makes $G_{d,i}$ inappropriate for predictability verification in the presence of hidden cycles [108]. Such a drawback does not exist in the diagnoser-like test automaton (to be referred here to as test automaton) [102], since it reveals the existence of state cycles connected by unobservable events only. The test automaton has been proposed in [102] as a way to perform the verification of fault codiagnosability in DES, which, differently from that proposed in [18], does not require the restrictive assumptions on language liveness and nonexistence of cycles of states connected with unobservable events only (**OA1** and **OA2**). Another advantage of the verification strategy proposed in [102] is that it is based on the search for non-trivial SCCs, as opposed to the diagnoser proposed in [18], which requires the search for cycles in both, the diagnoser and system automata.

The idea behind the test automaton is to synchronize the behavior observed by the N local diagnosers and the actual behavior of the system. In this regard, in order to build the test automaton, it is first necessary to build the label automaton $A_\ell = (X_{A_\ell}, \{\sigma_f\}, f_{A_\ell}, \Gamma_{A_\ell}, x_{0,A_\ell}, \emptyset)$ and compute $G_\ell = G \| A_\ell = (X_\ell, \Sigma, f_\ell, \Gamma_\ell, x_{0,\ell}, \emptyset)$. Then, for $i = 1, \ldots, n$, we compute the observer automata $G_{d,i} = Obs(G_\ell, \Sigma_{o,i}) = (X_{d,i}, \Sigma_{o,i}, f_{d,i}, \Gamma_{d,i}, x_{0,d,i}, \emptyset)$, which models the i-th local diagnoser automaton.

Finally, the test automaton is obtained as follows:

$$G_{scc} = \left(\|_{i=1}^{n} G_{d,i} \right) \| G_\ell.$$

Copredictability verification using test automaton

Inspired by the test automaton proposed in [102], we propose a copredictability verification algorithm (Algorithm 5.1) that does not require the usual assumptions on language liveness and absence of state cycles connected by unobservable events only. The idea behind Algorithm 5.1 is to verify if for each sequence s_f that ends with fault event σ_f, there exists at least one local predictor LP_i, $i = 1, \ldots, n$, that is able to distinguish some prefix $t_f \in \overline{s_f}$ prior to the fault event ($\sigma_f \notin t_f$) from all prefixes of arbitrarily long length normal sequences s_N. In this regard, given a system modeled by some automaton G, Algorithm 5.1 checks if the language generated by G is copredictable with respect to $P_{o,i}$ and Σ_f, returning either "*Yes*", when every fault occurrence can be predicted by at least one local predictor LP_i, or "*No*", otherwise.

Algorithm 5.1 *Copredictability verification using test automaton*

```
Input: Automaton $G$, event sets $\Sigma_{o,i}$, $i = 1, \ldots, n$.
Output: Copredictability decision: Yes or No.
  1. Build $G_\ell = (X_\ell, \Sigma, f_\ell, \Gamma_\ell, x_{0,\ell}, \emptyset) = G \| A_\ell$.
  2. If $\exists x_\ell \in X_\ell : \Gamma_\ell(x_\ell) = \emptyset$,
       (a) Set $\Sigma_{uo} = \Sigma_{uo} \cup \{\sigma_{uo}\}$
       (b) For all $x \in X_\ell$ : $\Gamma_\ell(x) = \emptyset$, set $f_\ell(x, \sigma_{uo}) = x$.
  3. Find all nontrivial SCCs of $G_\ell$ whose states are labeled with $N$ and form set
     $X_{SCC}^N$.
  4. Create set $X_{m,N} = \{x_\ell \in X_\ell : (\exists s \in \Sigma^*)[f_\ell(x_\ell, s) \in X_{SCC}^N]\}$
  5. Set $G_N = (X_\ell, \Sigma, f_\ell, \Gamma_\ell, x_{0,\ell}, X_{m,N})$.
  6. For $i = 1, \ldots, n$, build $G_i^N = Obs(G_N, \Sigma_{o,i}) = (X_i, \Sigma_{o,i}, f_i, \Gamma_i, x_{0,i}, X_{m,i})$.
  7. Set $X_{m,f} = \{x_\ell \in X_\ell : (\exists y_\ell \in X_\ell)[f_\ell(y_\ell, \sigma_f) = x_\ell]\}$.
  8. For every $x \in X$:
        (a) if $x \in X_{m,f}$, set $\Gamma_{\ell,f}(x) = \emptyset$;
        (b) else set $\Gamma_{\ell,f}(x) = \Gamma_{\ell}(x)$ and $f_{\ell,f}(x,\sigma) = f_{\ell}(x,\sigma)$ for all $\sigma \in \Gamma_{\ell}(x)$.
 9. Build $G_{\ell}^{f} = Ac(X_{\ell}, \Sigma, f_{\ell,f}, \Gamma_{\ell,f}, x_{0,\ell}, X_{m,f})$.
10. Build $G_{scc} = \left(\|_{i=1}^{n} G_i^N\right) \| G_{\ell}^{f} = (X_{scc}, \Sigma, f_{scc}, \Gamma_{scc}, x_{0,scc}, X_{m,scc})$.
11. If $X_{m,scc} = \emptyset$, return “Yes”. Otherwise, return “No”.
```

Remark 5.3 *A marked state $x_{scc} = (x_1, \ldots, x_n, x_{\ell})$ of G_{scc} means that the system has reached a faulty state without any local predictor being sure that the fault would inevitably occur, since (i) a marked state x_{ℓ} of G_{ℓ}^{f} means that a fault has just occurred, and; (ii) a marked state x_i, $i = 1, \ldots, n$, of G_i^N means that, from the point of view of the local predictor LP_i, there is at least one arbitrarily long length normal sequence $s_{N,i}$ whose prefix $t_{N,i} \in \overline{s_{N,i}}$ is such that $f_i(x_{0,i}, P_{o,i}(t_{N,i})) = x_i$. In this regard, for a sequence such that $f_{scc}(x_{0,scc}, s_f) = x_{scc} \in X_{m,scc}$, then, since we know that $s_f \in \Psi(\sigma_f)$ and, for $i = 1, \ldots, n$, there exists some arbitrarily long length normal sequence $s_{N,i}$ whose prefix $t_{N,i} \in \overline{s_{N,i}}$ is such that $P_{o,i}(t_{N,i}) = P_{o,i}(s_f)$, we may conclude that the fault has just occurred and yet, none of the local predictors LP_i was able to predict it. Thus, we may conclude that the presence of marked states in G_{scc} implies that language L is not copredictable with respect to $P_{o,i}$, $i = 1, \ldots, n$ and Σ_f.* □

Based on Algorithm 5.1 and Remark 5.3, we present a necessary and sufficient conditions for copredictability verification by using the diagnoser-like test automaton. We present the following fact first.

Fact 5.1 *Let $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, $P_o : \Sigma^* \to \Sigma_o^*$ and assume that there exists two sequences $s_1, s_2 \in L \subseteq \Sigma^*$ where $P_o(s_1) = P_o(s_2)$. Then, for all $t_1 \in \overline{s_1}$, there exists $t_2 \in \overline{s_2}$ such that $P_o(t_1) = P_o(t_2)$.* □

Fact 5.1 can be clarified as follows. Let us assume, without loss of generality, that $P_o(s_1) = P_o(s_2) = \sigma_1\sigma_2 \ldots \sigma_n$, $\sigma_i \in \Sigma_o$, $i = 1, \ldots, n$. Then, there exist $u_j, w_j \in \Sigma_{uo}^*$, $j = 1, \ldots, n+1$ such that $s_1 = u_1\sigma_1u_2\sigma_2 \ldots u_n\sigma_n u_{n+1}$ and $s_2 = w_1\sigma_1w_2\sigma_2 \ldots w_n\sigma_n$ w_{n+1}. Thus, for any $t_1 \in \overline{s_1}$, say $t_1 = u_1\sigma_1u_2\sigma_2 \ldots u_k\sigma_k u$, where $u \in \overline{u_{k+1}}$, we may define $t_2 \in \overline{s_2}$ such that $t_2 = w_1\sigma_1w_2\sigma_2 \ldots w_k\sigma_k w$, where $w \in \overline{w_{k+1}}$, which satisfy, $P_o(t_1) = P_o(t_2)$.

Theorem 5.1 *A language L is copredictable with respect to projections $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i = 1, \ldots, n$, and fault event set $\Sigma_f = \{\sigma_f\}$ if, and only if, the test automaton G_{scc}, computed in accordance with Algorithm 5.1, has no marked states.*

Proof.
($\Rightarrow$) Assume that automaton G_{scc} has at least one marked state. Since $G_{scc} =$

$\big(\|_{i=1}^{n} G_i^N\big)\|G_\ell^f$, it is clear that $\mathcal{L}_m(G_{scc}) = (\bigcap_{i=1}^{n} P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))) \cap \mathcal{L}_m(G_\ell^f)$. Thus, there exists a sequence $s_f \in \mathcal{L}_m(G_{scc})$ such that: *(i)* $s_f \in \mathcal{L}_m(G_\ell^f)$, which implies that, according to Step 9 of Algorithm 5.1, $s_f \in \Psi(\sigma_f)$, and, *(ii)* $s_f \in P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))$, $i = 1, \ldots, n$. On the other hand, since $G_i^N = \text{Obs}(G_N, \Sigma_{o,i})$, it is not difficult to see that $\mathcal{L}_m(G_i^N) = P_{o,i}(\mathcal{L}_m(G_N))$, which implies that $s_f \in P_{o,i}^{-1}(P_{o,i}(\mathcal{L}_m(G_N)))$, or equivalently, $P_{o,i}(s_f) \in P_{o,i}(\mathcal{L}_m(G_N))$. Let $u_{N,i} \in \mathcal{L}_m(G_N)$ be a sequence such that $P_{o,i}(s_f) = P_{o,i}(u_{N,i})$. According to Steps 3–5, the marked language of G_N is composed of all prefixes of arbitrarily long length normal sequences, and so, there exist arbitrarily long length sequences $s_{N,i} \in \mathcal{L}_m(G_N)$, $i = 1, \ldots, n$, $s_{N,i}$ not necessarily different from $s_{N,j}$ for $i \neq j$, such that $u_{N,i} \in \overline{s_{N,i}}$. According to Fact 5.1, since $P_{o,i}(s_f) = P_{o,i}(u_{N,i})$, we can conclude that for all $t_f \in \overline{s_f}$, there exists $u_i \in \overline{u_{N,i}}$ such that $P_{o,i}(t_f) = P_{o,i}(u_i)$. Write $s_{N,i} = u_i v_i$, and, since $s_{N,i} \in \mathcal{L}_m(G_N) \subseteq \mathcal{L}(G_N) = \mathcal{L}(G_\ell) = L_\ell$, we have that $u_i \in L_\ell$ and $v_i \in L_\ell/u_i$, which implies that:

$$\big(\forall z \in \mathbb{N}\big)\big(\exists s_f \in \Psi(\sigma_f)\big)\big(\forall t_f \in \overline{s_f}\big)\big[(\sigma_f \in t_f) \vee \neg\mathbf{P}\ \big]$$

where $\neg\mathbf{P} : \big(\forall i \in I\big)\big(\exists u_i \in L_\ell\big)\big(\exists v_i \in L_\ell/u_i\big)\big[P_{o,i}(u_i) = P_{o,i}(t_f) \wedge (\sigma_f \notin u_i) \wedge (\|v_i\| \geq z) \wedge (\sigma_f \notin v_i)\big]$, and so, L_ℓ is not copredictable with respect to $P_{o,i}$ and Σ_f. Notice that, if $\mathcal{L}(G) = L$ is live, then L and L_ℓ coincides, *i.e.*, $L = L_\ell$, which implies that L is not copredictable with respect to $P_{o,i}$ and Σ_f. On the other hand, if L is not live, then L_ℓ has been obtained from L by adding sequences $s\sigma_{uo}^*$ into L_ℓ for all $s \in L$ with no continuation (Step 2), which implies, according to Lemma 5.1, that L is not copredictable with respect to $P_{o,i}$ and Σ_f.

($\Leftarrow$) Let L be a live language, otherwise we make it live by adding sequences $s\sigma_{uo}$ into L for all $s \in L$ with no continuation. Assume now that the language L is not copredictable with respect to $P_{o,i}$ and Σ_f. Thus there exist sequences $s_f, s_{N,i} \in L$, such that, $s_f = s\sigma_f$, where $s \in \Sigma^*$ and $\sigma_f \notin s$ without loss of generality (as a consequence of language L being not copredictable, the first fault occurrence in some fault sequence cannot be predicted by any local predictor), and $s_{N,i} = u_i v_i$ ($\sigma_f \notin s_{N,i}$) has arbitrarily long length and satisfies $P_{o,i}(u_i) = P_{o,i}(t_f)$, $i = 1, \ldots, n$, for all $t_f \in \overline{s_f}$. As a consequence, $P_{o,i}(s_f) \in P_{o,i}(\overline{s_{N,i}})$, or equivalently, $s_f \in P_{o,i}^{-1}(P_{o,i}(\overline{s_{N,i}}))$. In addition, $s_f \in \mathcal{L}_m(G_\ell^f)$ since, according to Steps 7–9 of Algorithm 5.1, any sequence that denotes the first fault occurrence leads to a marked state, which is the case of s_f. Since $s_{N,i}$ is unbounded and $\sigma_f \notin s_{N,i}$, it is clear that, according to Steps 3–5, $\overline{s_{N,i}} \subseteq \mathcal{L}_m(G_N)$. According to Step 6, $\mathcal{L}_m(G_i^N) = P_{o,i}(\mathcal{L}_m(G_N))$, and since by construction $\overline{s_{N,i}} \subseteq \mathcal{L}_m(G_N)$, the following relationship can be established: $P_{o,i}(\overline{s_{N,i}}) \subseteq P_{o,i}(\mathcal{L}_m(G_N)) = \mathcal{L}_m(G_i^N)$. Thus $P_{o,i}^{-1}(P_{o,i}(\overline{s_{N,i}})) \subseteq P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))$, which implies that, since $s_f \in P_{o,i}^{-1}(P_{o,i}(\overline{s_{N,i}}))$, it is straightforward to

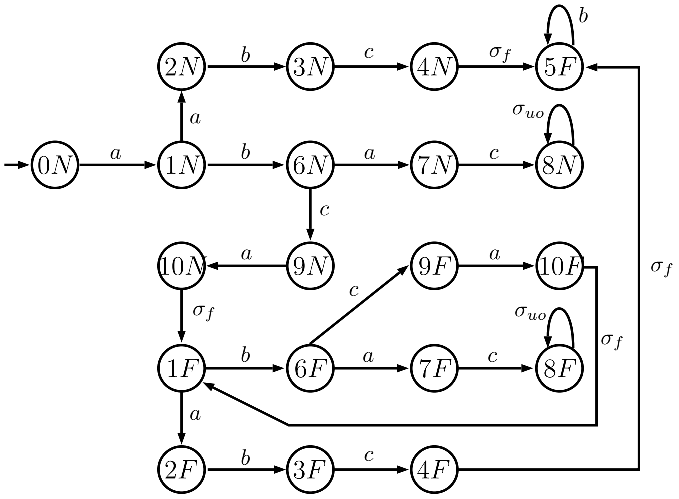


Figure 5.4: Automaton G_ℓ.

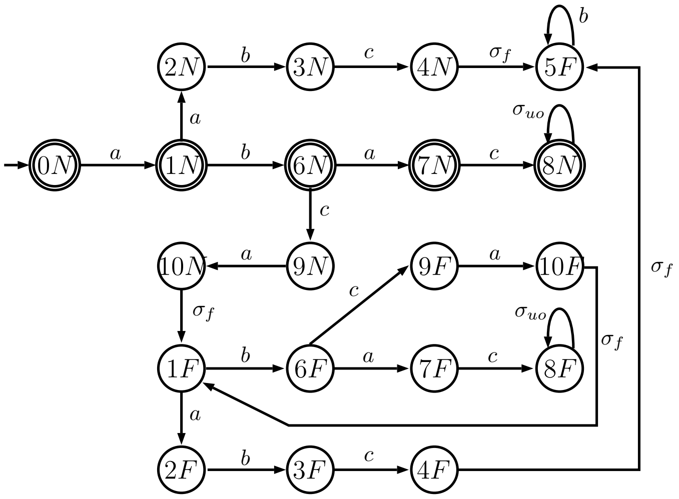
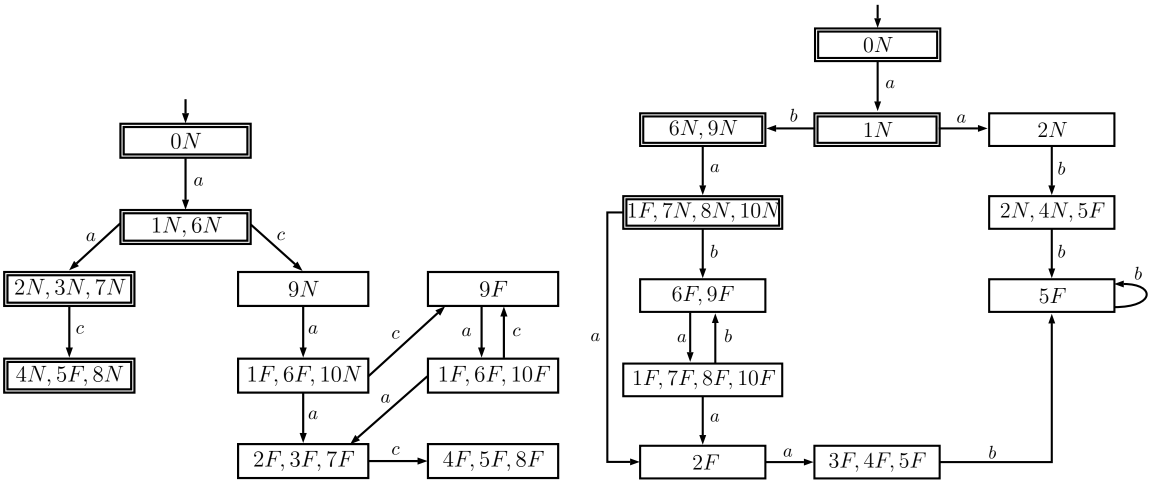


Figure 5.5: Automaton G_N.

see that $s_f \in P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))$, for all $i = 1, \ldots, n$. Finally, since $G_{scc} = \left(\|_{i=1}^n G_i^N\right) \| G_\ell^f$, its marked language is $\mathcal{L}_m(G_{scc}) = (\bigcap_{i=1}^n P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))) \cap \mathcal{L}_m(G_\ell^f)$, which together with the fact that $s_f \in P_{o,i}^{-1}(\mathcal{L}_m(G_i^N))$, for all $i = 1, \ldots, n$, and $s_f \in \mathcal{L}_m(G_\ell^f)$, we may conclude that $s_f \in \mathcal{L}_m(G_{scc})$, which ultimately implies that $\mathcal{L}_m(G_{scc}) \neq \emptyset$, and thus, automaton G_{scc} has at least one marked state. ■

Example 5.3 *We revisit the system presented in Example 5.2, whose behavior is modeled in Figure 5.3. Recall that the observable event sets of the two local predictors LP_1 and LP_2 are $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$, respectively.* Step 1 *of Algorithm 5.1 computes automaton G_ℓ, depicted in Figure 5.4, except for the self-loop with unobservable event σ_{uo}, which is added to G_ℓ in* Step 2. Step 3 *computes all SCCs of G_ℓ formed with normal states only, and so, it creates the set $X_{SCC}^N = \{8N\}$.* Step 4 *computes set $X_{m,N}$ as all states of G_ℓ that reach the states in X_{SCC}^N through some sequence $s \in \Sigma^*$, which results in $X_{m,N} = \{0N, 1N, 6N, 7N, 8N\}$. Then,*



(a) $G_1^N = Obs(G_N, \Sigma_{o,1})$.

(b) $G_2^N = Obs(G_N, \Sigma_{o,2})$.

Figure 5.6: Observer automata G_1^N and G_2^N.

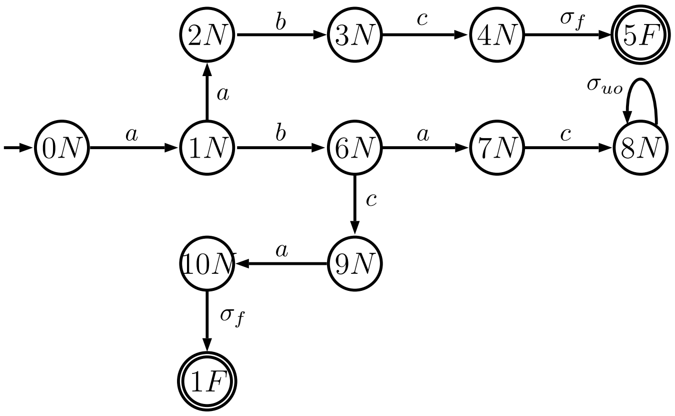


Figure 5.7: Automaton G_ℓ^f.

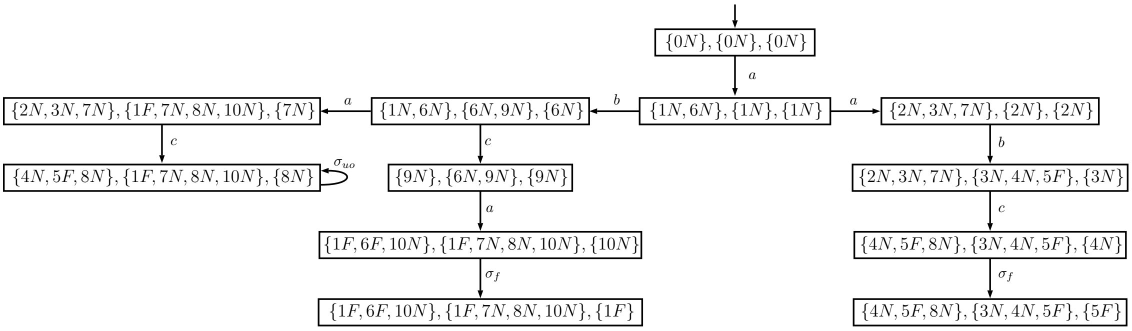

{0N}, {0N}, {0N}
a
{1N, 6N}, {1N}, {1N}
b
{1N, 6N}, {6N, 9N}, {6N}
a
{2N, 3N, 7N}, {1F, 7N, 8N, 10N}, {7N}
c
{4N, 5F, 8N}, {1F, 7N, 8N, 10N}, {8N}
σ_{uo}
c
{9N}, {6N, 9N}, {9N}
a
{1F, 6F, 10N}, {1F, 7N, 8N, 10N}, {10N}
σ_f
{1F, 6F, 10N}, {1F, 7N, 8N, 10N}, {1F}
a
{2N, 3N, 7N}, {2N}, {2N}
b
{2N, 3N, 7N}, {3N, 4N, 5F}, {3N}
c
{4N, 5F, 8N}, {3N, 4N, 5F}, {4N}
σ_f
{4N, 5F, 8N}, {3N, 4N, 5F}, {5F}

Figure 5.8: Relevant part of test automaton $G_{scc} = G_1^N \| G_2^N \| G_\ell^f$.

STEP 5 *builds G_N as a copy of automaton G_ℓ but with the following marked states $X_{m,N} = \{0N, 1N, 6N, 7N, 8N\}$, as shown in Figure 5.5.* STEP 6 *computes the observer automata G_1^N and G_2^N with respect to the event sets $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$, which are depicted in Figures 5.6a and 5.6b, respectively.* STEP 7 *forms the set $X_{m,f}$ with all states of G_ℓ that have been reached by a fault event, i.e., $X_{m,f} = \{1F, 5F\}$, and then,* STEP 8 *either sets $f_{\ell,f}(x, \sigma)$ as a copy of $f_\ell(x, \sigma)$ when state $x \notin X_{m,f}$, or equal to the empty set otherwise.* STEP 9 *computes automaton G_ℓ^f, depicted in Figure 5.7, as the accessible part of automaton G_ℓ after marking states in $X_{m,f}$ and removing all transitions that originate from these newly marked states. Finally,* STEP 10 *builds the test automaton $G_{scc} = G_1^N \| G_2^N \| G_\ell^f$, shown in Figure 5.8. Notice that automaton G_{scc} has no marked states, and thus, Algorithm 5.1 returns "Yes" in* STEP 11, *which means that language L generated by automaton G is copredictable with respect to $P_{o,i}$, $i = 1, 2$, and Σ_f, which concurs with the results presented in Example 5.2.* □

Remark 5.4 *Regarding the system presented in Example 5.1 and shown in Figure 5.2, its test automaton, built in accordance with Algorithm 5.1, has a marked state, $(\{3N, 4F, 6N\}, \{3N, 4F, 6N\}, \{4F\})$, achieved by sequence $abc\sigma_f$, Thus, the copredictability decision given by Algorithm 5.1 is "No", which concurs with the results of Example 5.1.* □

Remark 5.5 *(Computational complexity) The main computational burden of Algorithm 5.1 is the construction of observer automata $G_i^N = Obs(G_N, \Sigma_{o,i})$ in* STEP 6, *which, in the worst case scenario, has $2^{|X_\ell|}$ states, whereas the other steps are either linear (*STEPS 1–5, 7–9 *and* 11*) or polynomial (*STEP 10*) in the number of states. In addition, $|X_\ell| = 2|X|$, since $G_\ell = G \| A_\ell$, and thus, Algorithm 5.1 is $O(2^{2n|X|} \times |2X|) = O(2^{2n|X|} \times |X|)$. It is worth remarking that this algorithm relies on the search for SCCs, which is linear in the number of automaton transitions [109, 110].* □

5.2.2 Copredictability verification using verifiers

The verifier automaton

Other strategies that are typically used in fault codiagnosability verification deploy the so-called verifier automata [104, 111–113], that have, in the worst case, lower computational complexity than that associated with the construction of diagnoser-based automata. In this regard, we follow the approach proposed in [104], which is based on the synchronization of the fault and normal behaviors of the system that are identically observed by the local agents, and also require the least computational complexity among other verification algorithms [111–113].

To this end, as in the procedure for building the test automata, we firstly compute G_ℓ, which is used later on to build G_f, which models the fault behavior, and $G_{N,i}$, which models the normal behavior from the perspective of the i-th diagnoser, $i = 1, \ldots, n$. Automaton G_f is obtained by marking the states of G_ℓ that are labeled with F and then taking its coaccessible part. Similarly, automaton G_N is obtained by marking the SCCs in G_ℓ that are labeled with N and then taking its coaccessible part. Notice that, by construction, G_N has no fault event. Next, we recall the renaming function [104], which adds subscripts R_i, $i = 1, \ldots, n$, to all locally unobservable events in $\Sigma \setminus (\Sigma_{o,i} \cup \Sigma_f)$, as follows.

Definition 5.2 (Renaming function) *Let $\Sigma_N = \Sigma \setminus \Sigma_f$, $\Sigma_i = \{\sigma_{R_i} : \sigma \in \Sigma_N \setminus \Sigma_{o,i}\}$ and $\Sigma_{R_i} = \Sigma_{o,i} \cup \Sigma_i$. The renaming function $R_i : \Sigma_N \to \Sigma_{R_i}$, $i = 1, 2, \ldots, n$, is a mapping, where*

$$R_i(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{o,i}, \\ \sigma_{R_i}, & \text{if } \sigma \in \Sigma_N \setminus \Sigma_{o,i}. \end{cases}$$

The inverse renaming function is the mapping $R_i^{-1} : \Sigma_{R_i} \to \Sigma_N$, $i = 1, 2, \ldots, n$, where $R_i^{-1}(\sigma_{R_i}) = \sigma$, and $R_i^{-1}(\sigma) = \sigma$. □

Notice that, the renaming function (resp. inverse renaming function) can be extended to sequences, as follows: $R_i(s\sigma) = R_i(s)R_i(\sigma)$ (resp. $R_i^{-1}(s\sigma) = R_i^{-1}(s)R_i^{-1}(\sigma)$), where $s \in \Sigma_N^*$ and $\sigma \in \Sigma_N$. We denote $\Sigma_{R_i} = R_i(\Sigma_N)$ the set formed with both the locally observable events $\Sigma_{o,i}$ and the locally unobservable events in $\Sigma_N \setminus \Sigma_{o,i}$ that have been renamed.

Then, for every $\Sigma_{o,i} \subseteq \Sigma_o$, we compute $G_{N,i}$ by renaming all of the locally unobservable events of G_N with the renaming function R_i. Finally, the verifier automaton is computed as follows:

$$G_v = \left(||_{i=1}^{n} G_{N,i}\right) || G_f.$$

Copredictability verification using verifier

Similarly to the verifier proposed in [104], the idea behind the verification algorithm for fault copredictability verification of regular languages using verifiers is to synchronize only those sequences whose last event is faulty and those that are prefixes of arbitrarily long length normal sequences, which are identically observed by all local predictors.

Therefore, the copredictability verification of regular languages generated by some automaton G, with respect to $P_{o,i}$, $i = 1, \ldots, n$, and Σ_f can be performed according to Algorithm 5.2.

Algorithm 5.2 *Copredictability Verification using verifier*

```
Input: Automaton $G$, Event sets $\Sigma_{o,i}$, $i = 1, \dots, n$.
Output: Copredictability decision: Yes or No.
  1. Build $G_\ell = (X_\ell, \Sigma, f_\ell, \Gamma_\ell, x_{0,\ell}, \emptyset) = G \| A_\ell$.
  2. If $\exists x_\ell \in X_\ell : \Gamma_\ell(x_\ell) = \emptyset$,
       (a) Set $\Sigma_{uo} = \Sigma_{uo} \cup \{\sigma_{uo}\}$
       (b) For all $x \in X_\ell :\ \Gamma_\ell(x) = \emptyset$, set $f_\ell(x, \sigma_{uo}) = x$.
  3. Set $X_{m,\ell} = \{x_\ell \in X_\ell : (\exists y_\ell \in X_\ell)[f_\ell(y_\ell, \sigma_f) = x_\ell]\}$.
  4. For all $(x, \sigma) \in X_{m,\ell} \times \Sigma$, set $f_\ell(x, \sigma) = \emptyset$ and $\Gamma_\ell(x) = \emptyset$.
  5. Compute $G_f = Ac(CoAc(G_\ell))$ and set $X_{m,f} = \emptyset$.
  6. For all $x_\ell \in X_\ell$, set $\Gamma_\ell(x_\ell) = \Gamma_\ell(x_\ell) \setminus \{\sigma_f\}$.
  7. Compute $G_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0,N}, \emptyset) = Ac(G_\ell)$.
  8. Find and mark all states in the nontrivial SCCs of $G_N$.
  9. Compute $G_N = CoAc(G_N)$ and set $X_{m,N} = \emptyset$.
 10. For each $\Sigma_{o,i}$, $i = 1, \dots, n$, build $G_{N,i} = (X_{N,i}, \Sigma_{R_i}, f_{N,i}, \Gamma_{N,i}, x_{0,N}, \emptyset)$, where $X_{N,i} = X_N$, $\Sigma_{R_i} = R_i(\Sigma_N)$, $f_{N,i}(x, R_i(\sigma)) = f_N(x, \sigma)$ and $\Gamma_{N,i}(x) = R_i(\Gamma_N(x))$.
 11. Compute $G_v = \big(\|_{i=1}^{n} G_{N,i}\big) \| G_f = (X_v, \Sigma, f_v, \Gamma_v, x_{0,v}, X_{m,v})$.
 12. If $G_v$ has no state $x_v = (x_{N,1}, \dots, x_{N,n}, x_f)$ such that $x_f$ is labeled with $F$, then return "Yes". Otherwise, return "No".
```

Let us define the following projections:

(i) $P_\Sigma : (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2} \cup \cdots \cup \Sigma_{R_N})^* \to \Sigma^*$

(ii) $P_{\Sigma_{R_i}} : (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2} \cup \cdots \cup \Sigma_{R_N})^* \to \Sigma_{R_i}^*$

A necessary and sufficient condition for copredictability verification by means of verifiers, proposed on Algorithm 5.2, can be obtained in a more straightforward way with the help of the following lemma [114, Lemma 1].

Lemma 5.2 *Let $G_v = (\|_{i=1}^{n} G_{N,i}) \| G_f$. Then, for every $s_v \in \mathcal{L}(G_v)$, there exist n sequences $s_{N_1}, s_{N_2}, \dots, s_{N_n} \in \mathcal{L}(G_N)$, s_{N_j} not necessarily different from s_{N_k}, for $j \neq k$, and $s_f \in \mathcal{L}(G_f)$, such that $s_f = P_\Sigma(s_v)$, $s_{N_i} = R_i^{-1}\big(P_{\Sigma_{R_i}}(s_v)\big)$, and $P_{o,i}(s_f) = P_{o,i}(s_{N_i})$, $i = 1, \dots, n$, and conversely, if there exist n sequences $s_{N_1}, s_{N_2}, \dots, s_{N_n} \in \mathcal{L}(G_N)$, s_{N_j} not necessarily different from s_{N_k}, for $j \neq k$, and $s_f \in \mathcal{L}(G_f)$, such that $P_{o,i}(s_f) = P_{o,i}(s_{N_i})$, $i = 1, \dots, n$, then there must exist a sequence $s_v \in \mathcal{L}(G_v)$ such that $s_f = P_\Sigma(s_v)$, $s_{N_i} = R_i^{-1}\big(P_{\Sigma_{R_i}}(s_v)\big)$, $i = 1, \dots, n$.* □

We now present a necessary and sufficient condition for copredictability verification by means of verifiers.

Theorem 5.2 *A language L is copredictable with respect to projections $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i = 1, \ldots, n$, and fault event set $\Sigma_f = \{\sigma_f\}$ if, and only if, x_f is labeled by N in all states $x_v = (x_{N,1}, \ldots, x_{N,n}, x_f)$ of verifier automaton G_v, obtained in accordance with Algorithm 5.2.*

Proof.
$(\Rightarrow)$ Assume that there exists one state $x_v = (x_{N,1}, \ldots, x_{N,n}, x_f)$ of automaton G_v whose component x_f is labeled by F, and let $s_v \in \mathcal{L}(G_v)$ denote the sequence that reaches state x_v, i.e., $f_v(x_{0,v}, s_v) = x_v$. According to Lemma 5.2, there must exist sequences $s_{N,i} \in \mathcal{L}(G_N)$, $i = 1, \ldots, n$, and $s_f \in \mathcal{L}(G_f)$, such that $P_{o,i}(s_f) = P_{o,i}(s_{N,i})$. By construction of G_v, since x_f is F-labeled, we conclude that $s_f \in \Psi(\sigma_f)$. In addition, since $s_{N,i} \in \mathcal{L}(G_N)$, these sequences are prefixes of arbitrarily long length normal sequences $s'_{N,i}$, i.e., $s_{N,i} \in \overline{s'_{N,i}}$, where $\|s'_{N,i}\| \geq z$ for all $z \in \mathbb{N}$. According to Fact 5.1, since $P_{o,i}(s_f) = P_{o,i}(s_{N,i})$, for all $t_f \in \overline{s_f}$, we can find $u_i \in \overline{s_{N,i}}$ and v_i such that $P_{o,i}(t_f) = P_{o,i}(u_i)$ and $s_{N,i} = u_i v_i$. It is worth remarking that $\mathcal{L}(G_N) \subseteq \mathcal{L}(G_\ell) = L_\ell$, and thus, $s_{N,i} \in L_\ell$ implies that $u_i \in L_\ell$ and $v_i \in L_\ell / u_i$. We can, thus, conclude that

$$\big(\forall z \in \mathbb{N}\big)\big(\exists s_f \in \Psi(\sigma_f)\big)\big(\forall t_f \in \overline{s_f}\big)\big[(\sigma_f \in t_f) \vee \neg \mathbf{P}\ \big]$$

where $\neg\mathbf{P} : \big(\forall i \in I\big)\big(\exists u_i \in L_\ell\big)\big(\exists v_i \in L_\ell/u_i\big)\big[P_{o,i}(u_i) = P_{o,i}(t_f) \wedge (\sigma_f \notin u_i) \wedge (\|v_i\| \geq z) \wedge (\sigma_f \notin v_i)\big]$, which implies that L_ℓ is not copredictable with respect to $P_{o,i}$ and Σ_f. Notice that, if $\mathcal{L}(G) = L$ is live, then L and L_ℓ coincides, *i.e.*, $L = L_\ell$, which implies that L is not copredictable with respect to $P_{o,i}$ and Σ_f. On the other hand, if L is not live, then L_ℓ has been obtained from L by adding sequences $s\sigma_{uo}^*$ into L_ℓ for all $s \in L$ with no continuation (Step 2), which implies, according to Lemma 5.1, that L is not copredictable with respect to $P_{o,i}$ and Σ_f.

$(\Leftarrow)$ Let L be a live language, otherwise we make it live by adding sequences $s\sigma_{uo}$ into L for all $s \in L$ with no continuation. Assume now that L is not copredictable with respect to $P_{o,i}$ and Σ_f. Then, there exists a sequence $s_f = s\sigma_f$, where $s \in \Sigma^*$ and, without loss of generality, $\sigma_f \notin s$, such that for each $i \in I = \{1, \ldots, n\}$, there exist arbitrarily long length normal sequences $s_{N,i}$, such that all prefixes $t_f \in \overline{s_f}$ have the same projection with respect to $\Sigma_{o,i}$ as some prefix $u_i \in \overline{s_{N,i}}$, namely $P_{o,i}(u_i) = P_{o,i}(t_f)$ for all $t_f \in \overline{s_f}$, and, in particular, for $t_f = s_f$. Thus, $P_{o,i}(s_f) \in P_{o,i}(\overline{s_{N,i}})$. Without loss of generality, let us choose $u_{N,i} \in \overline{s_{N,i}}$ as the sequence that satisfies $P_{o,i}(u_{N,i}) = P_{o,i}(s_f)$.

According to Steps 1–5 of Algorithm 5.2, sequence s_f is generated by automaton G_f ($s_f \in \mathcal{L}(G_f)$), which implies that there exists $x_f \in X_f$ such that x_f is F-labeled.

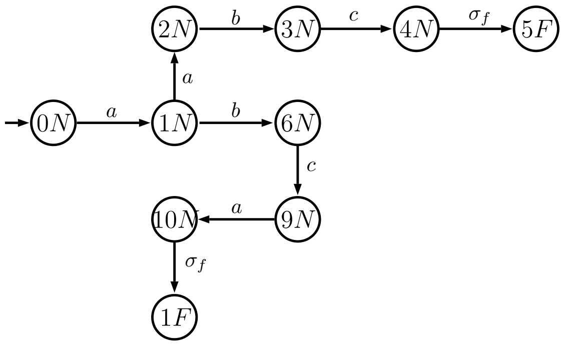
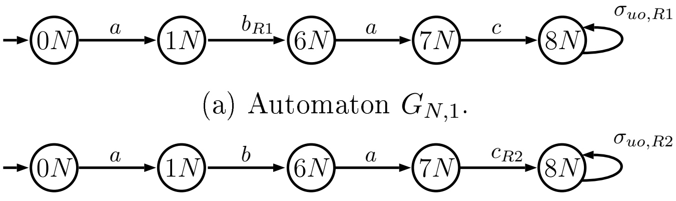


Figure 5.9: Automaton G_f.



(a) Automaton $G_{N,1}$.

(b) Automaton $G_{N,2}$.

Figure 5.10: Automata $G_{N,1}$ and $G_{N,2}$.

According to Steps 6–9, all arbitrarily long length normal sequences $s_{N,i}$ remain in G_N, and thus, $\overline{s_{N,i}} \subseteq \mathcal{L}(G_N)$. Therefore, every $u_{N,i} \in \mathcal{L}(G_N)$, $i = 1, \ldots, n$.

Since $u_{N,i} \in \mathcal{L}(G_N)$, $i = 1, \ldots, n$, $s_f \in \mathcal{L}(G_f)$, and $P_{o,i}(s_f) = P_{o,i}(u_{N,i})$, we conclude, according to Lemma 5.2, that there exists a sequence $s_v \in \mathcal{L}(G_v)$, such that: (i) $P_\Sigma(s_v) = s_f \in \mathcal{L}(G_f)$; (ii) $s_f \in \Psi(\sigma_f)$, and; (iii) $f_v(x_{0,v}, s_v) = x_v = (x_{N,1}, \ldots, x_{N,n}, x_f)$, x_f is F-labeled. ■

Example 5.4 *Consider again the system modeled in Figure 5.3 and presented in Example 5.2. After Algorithm 5.2 has built G_ℓ from G and marked their states,* Steps 3–5 *construct automaton G_f, depicted in Figure 5.9. Next,* Steps 6–9 *compute automaton G_N, and then,* Step 10 *builds automata $G_{N,1}$ and $G_{N,2}$, illustrated in Figures 5.10(a) and 5.10(b), respectively.* Step 11 *computes the verifier automaton $G_v = G_{N,1} \| G_{N,2} \| G_f$, shown in Figure 5.11. Since there is no state $x_v = (x_{N,1}, x_{N,2}, x_f)$ in G_v such that x_f is labeled with F,* Step 11 *returns "Yes", meaning that the language generated by automaton G is copredictable with respect to $P_{o,i}$, $i = 1, 2$, and Σ_f.* □

Remark 5.6 *(Computational complexity) Notice that* Steps 1–10 *and* 12 *require operations that are linear in the size of the automaton state set. Thus, the worst case complexity of Algorithm 5.2 is majored by* Step 11*, where the parallel composition of G_f and $G_{N,i}$, $i = 1, 2, \ldots, n$ is computed. Since, in the worst case scenario, automata G_f and each $G_{N,i}$ have $2|X|$ states (they are obtained from $G_\ell = G \| A_\ell$,*

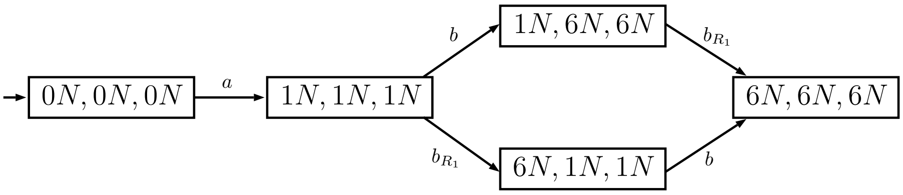


Figure 5.11: Verifier G_v.

where G has $|X|$ states and A_ℓ has two states), G_v has at most $(2|X|)^{n+1}$ states. Thus, Algorithm 5.2 is $O(2^{n+1}|X|^{n+1})$, which is polynomial on the number of the states of the system and exponential on the number of local predictors. However, since the number of local predictors cannot grow without limit (being, in general, dictated by the spacial distribution of the physical system under investigation), the number of states of the system dictates the computational complexity. Notice that the verification algorithm presented in [85], where the non-faulty behavior is represented by an arbitrary closed sublanguage $K \subseteq L$ and a finite automaton R with $\mathcal{L}(R) = K$, also requires polynomial time for checking copredictability. The computational complexity to build their verifier is $O(|X| \times |X_R|^{n+1})$, where X and X_R are the state sets of automata G and R, respectively. However, in the special case when $K = \{s \in L | s \in (\Sigma \setminus \Sigma_f)^*\}$, which is the case of the problem considered here, R can be constructed as a subautomaton of G, and as a consequence, the verification method of [85] is reduced to $O(|X|^{n+1})$, which is the same as the verifier proposed here, considering only the number of states for a fixed number of local predictors. □

5.3 A disjunctive fault predictor system

In this section, we address the problem of designing local fault predictors LP_i, $i = 1, \ldots, n$, for the decentralized structure depicted in Figure 5.1. We assume that the language is copredictable, otherwise there is no need to build such a device. The idea behind the design of local fault predictor systems is presented in the following example.

Example 5.5 *Consider the system whose behavior is modeled in Figure 5.3. Assume that the observable event sets of the local predictors LP_1 and LP_2 are $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$, respectively. From Figure 5.3, it is clear that $\Psi(\sigma_f) = \{aabc\sigma_f, abca\sigma_f, abca\sigma_f bca\sigma_f, \ldots\}$, and that there exists a finite normal sequence*[1] *$s_N = abac$. We consider only sequences $s_{f,1} = aabc\sigma_f$ and $s_{f,2} = abca\sigma_f$, since they are the only sequences that represent the first occurrence of the fault.*

[1] This sequence could be made arbitrarily long length by creating a self-loop in state 8 labeled by an unobservable event σ_{uo}.

Notice that LP_1 can distinguish sequence $s_{f,2} = abca\sigma_f$ from s_N before the occurrence of σ_f, since, for $t_2 = abc \in \overline{s_{f,2}}$, $P_{o,1}(t_2) = ac \notin \overline{P_{o,1}(s_N)} = \overline{aac}$. This means that prefix $t_2 \in \overline{s_{f,2}}$ cannot be mistaken with any prefix of normal sequences. However, LP_1 cannot distinguish between sequences s_N and $s_{f,1} = aabc\sigma_f$, since $\overline{P_{o,1}(s_{f,1})} = \{\varepsilon, a, aa, aac\} \subset \overline{P_{o,1}(s_N)}$, and therefore, LP_1 is able to predict only the fault that occurs in $s_{f,2}$. Analogously, LP_2 is able to predict the fault that occurs in $s_{f,1} = aabc\sigma_f$, since, for $t_1 = aa$, $P_{o,2}(t_1) = aa \notin \overline{P_{o,2}(s_N)} = \overline{aba}$, but is not able to predict the fault that occurs in $s_{f,2}$. We conclude that the prefix of the fault sequence $s_{f,1}$ (resp. $s_{f,2}$) that leads LP_2 (resp. LP_1) to predict the fault occurrence is $t_1 = aa$ (resp. $t_2 = abc$). Therefore, the identification of the prefixes of the fault sequences that make the local predictors foresee the fault occurrence plays a key role in the design of disjunctive decentralized fault predictor systems. □

By assuming that language L is copredictable with respect to $P_{o,i}$ and Σ_f, the problem of designing local fault predictors LP_i, $i = 1, \ldots, n$ for disjunctive decentralized systems can be transformed into a problem of identifying which prefixes of the fault sequences make the local predictors LP_i, $i = 1, \ldots, n$, foresee the fault occurrence.

Notice that, since, by construction, the marked language of observer automaton G_i^N, obtained according to STEP 6 of Algorithm 5.1, is formed with the projections of prefixes of all arbitrarily long length normal sequences, the following cases are possible:

- If the observation s_o of some sequence $s \in L$ reaches marked states of G_i^N, then we are sure that $P_{o,i}^{-1}(s_o) \cap L$ includes prefixes of arbitrarily long length normal sequences, and so, a verdict that a fault will occur cannot be issued. For example, consider state $\{1F, 7N, 8N, 10N\}$ of automaton G_2^N, depicted in Figure 5.6b, which is reached by the observation $s_o = aba$. Notice that $P_{o,2}^{-1}(s_o) \cap L$ includes both a prefix of a faulty sequence $abca \in \overline{abca\sigma_f}$ and a prefix of an arbitrarily long length normal sequence $abac \in \overline{abac\sigma_{uo}^k}$, $k \in \mathbb{N}$, and thus, local predictor LP_2 cannot ascertain that a fault will occur by observing s_o.

- If the observation s'_o (of some sequence $s' \in L$) reaches unmarked states of G_i^N that have at least one element labeled with N, then $P_{o,i}^{-1}(s'_o) \cap L$ does not include prefixes of arbitrarily long length normal sequences, and so, every continuation of s' will inevitably lead G to a fault (otherwise, if it has some arbitrarily long length continuation without any fault occurrence, it would make it a prefix of arbitrarily long length normal sequences, and so, the state reached by s'_o would be marked). For example, consider state $\{2N\}$ in Figure 5.6b, which is reached by the observation $s'_o = aa$, and then, notice that $P_{o,2}^{-1}(s'_o) \cap L = aa$, whose only continuation forms the faulty sequence $aabc\sigma_f$.

- Finally, if the observation s''_o (of some sequence $s'' \in L$) reaches unmarked states of G_i^N and all of their elements are labeled with F, then $P_{o,i}^{-1}(s''_o) \cap L$ corresponds to sequences that have the fault event, and thus, observation s''_o must be discarded for the design of local fault predictors systems. For example, state $\{6F, 9F\}$ of Figure 5.6b is reached by the observation of $s'_o = abaa$; however $P_{o,2}^{-1}(s''_o) \cap L = abca\sigma_f a$ corresponds to a sequence for which the fault event has already occurred.

Therefore, let x represent all first unmarked states that immediately follow a marked state of G_i^N and is composed of some state labeled with N. Thus, sequences $s_i \in L(G_i^N)$ that reach x represent the observation $s_i = P_{o,i}(t_f)$ of the smallest prefix $t_f \in \overline{s_f}$ ($s_f \in \Psi(\sigma_f)$) that is different from every prefix of (arbitrarily long length) normal sequences from the point of view of local predictor LP_i. We illustrate this point with the following example.

Example 5.6 *Consider the local observer automata G_1^N and G_2^N, depicted in Figures 5.6a and 5.6b, respectively. The only unmarked state of G_1^N that immediately follows some marked state is $x_1 = \{9N\}$, which is reached by sequence $t_{o,1} = ac$. Regarding G_2^N, the unmarked states that immediately follow some marked state are $x_2 = \{2N\}$, $x'_2 = \{6F, 9F\}$, and $x''_2 = \{2F\}$. The sequences that reach these states are $t_{o,2} = aa$, $t'_{o,2} = abab$, and $t''_{o,2} = abaa$, respectively. However, notice that the observations of $t'_{o,2}$ and $t''_{o,2}$ lead G_2^N to states x'_2, and x''_2, whose components are all labeled with F, meaning that the fault has already occurred, namely: (i) $t'_{o,2} = abab$ corresponds to sequences $abca\sigma_f b$ and $abca\sigma_f bc$, and; (ii) $t''_{o,2} = abaa$ corresponds to sequence $abca\sigma_f a$. Thus, sequences $t_{o,1}$ and $t_{o,2}$ are the smallest observations that allow local predictors LP_1 and LP_2 to foresee fault occurrences, which coincides with the results obtained in Example 5.5.* □

The existence of prefixes that allow us to foresee that a fault will certainly occur suggests that a fault prediction state should be part of fault predictor systems. Such state is denoted by FP, and, once it is visited, it implies that some fault will inevitably occur. Algorithm 5.3 presents the construction of local fault predictor automata $G_{lp,i}$, $i = 1, \ldots, n$.

Algorithm 5.3 *Construction of Local Fault Predictor Automaton $G_{lp,i}$*

Input: Automaton $G_i^N = (X_i, \Sigma_{o,i}, f_i, \Gamma_i, x_{0,i}, X_{m,i})$.
Output: Local Predictor Automaton $G_{lp,i}$.

```
1. Set $Q = X_i \setminus X_{m,i}$.
2. Set $X_i = X_i \cup \{FP\}$.
3. For each $q \in Q$:
    (a) If $\exists(x_m, \sigma) \in X_{m,i} \times \Sigma_{o,i}$ such that $f_i(x_m, \sigma) = q$ and $q$ has some element
        labeled with $N$, then redefine $f_i(x_m, \sigma) = FP$.
4. Set $X_{m,i} = \{FP\}$.
5. Compute $G_{lp,i} = Ac(G_i^N)$.
```

Step 1 of Algorithm 5.3 creates a set Q formed with all unmarked states of automaton G_i^N. Step 2 adds state FP, used to issue that some fault occurrence has been foreseen, to the state set of G_i^N. Step 3 finds the unmarked states composed with at least one element labeled with N that are immediately succeeded by some marked state, and then, replaces the transition with a new one connecting the marked state to state FP. Step 4 sets state FP as the only marked state. Finally, Step 5 forms $G_{lp,i}$ as the accessible part of G_i^N.

From Algorithm 5.3, it is clear that the marked language of automaton $G_{lp,i}$ is formed with the smallest observed sequences $P_{o,i}(s)$ that lie outside the projection of arbitrarily long length normal sequences $P_{o,i}(s_N)$, and thus, when sequences $s \in \mathcal{L}_m(G_{lp,i})$ are executed by the system, the corresponding local predictor LP_i is sure that a fault will inevitably occur.

Notice that the fault predictor systems developed in Algorithm 5.3 do not issue false-alarms, i.e., if some local fault predictor LP_i observes a sequence that is in its marked language, which implies that the current state of automaton $G_{lp,i}$ is FP, then the fault event will occur for sure. This is ensured by the following lemma.

Lemma 5.3 *Let L be the language generated by an automaton G and assume that L is copredictable with respect to projections $P_{o,i}$, $i = 1, \ldots, n$, and fault event set Σ_f. Then, for all $s_p \in \mathcal{L}_m(G_{lp,i})$, where $G_{lp,i}$ is obtained in accordance with Algorithm 5.3, the following two statements are both true: (i) $P_{o,i}^{-1}(s_p) \cap \overline{\Psi(\sigma_f)} \neq \emptyset$ and; (ii) all sequences in $P_{o,i}^{-1}(s_p) \cap L$ are not prefix of any arbitrarily long length normal sequence of L.* □

Proof. Assume that L is copredictable with respect to Σ_f and all $P_{o,i}$, $i = 1, \ldots, n$, but there exists a sequence $s_p \in \mathcal{L}_m(G_{lp,i})$ such that, either (i) $P_{o,i}^{-1}(s_p) \cap \overline{\Psi(\sigma_f)} = \emptyset$, or (ii) some sequence in $P_{o,i}^{-1}(s_p) \cap L$ is a prefix of some arbitrarily long length normal sequence of L. However, if $P_{o,i}^{-1}(s_p) \cap \overline{\Psi(\sigma_f)} = \emptyset$, then all sequences $s \in P_{o,i}^{-1}(s_p) \cap L$ have no fault continuations, which means that they are prefixes of arbitrarily long length normal sequences, and so, $s \in \mathcal{L}_m(G_N)$. Thus, $P_{o,i}(s) = s_p \in \mathcal{L}_m(G_i^N)$, and, according to Steps 3 and 4 of Algorithm 5.3, $s_p \notin \mathcal{L}_m(G_{lp,i})$, which contradicts the initial assumption. On the other hand, if some sequence in $s \in P_{o,i}^{-1}(s_p) \cap L$ is a prefix of some arbitrarily long length normal sequence of L, then, again, $P_{o,i}(s) = s_p \in \mathcal{L}_m(G_i^N)$, and so, $s_p \notin \mathcal{L}_m(G_{lp,i})$, therefore contradicting the initial assumption. ■

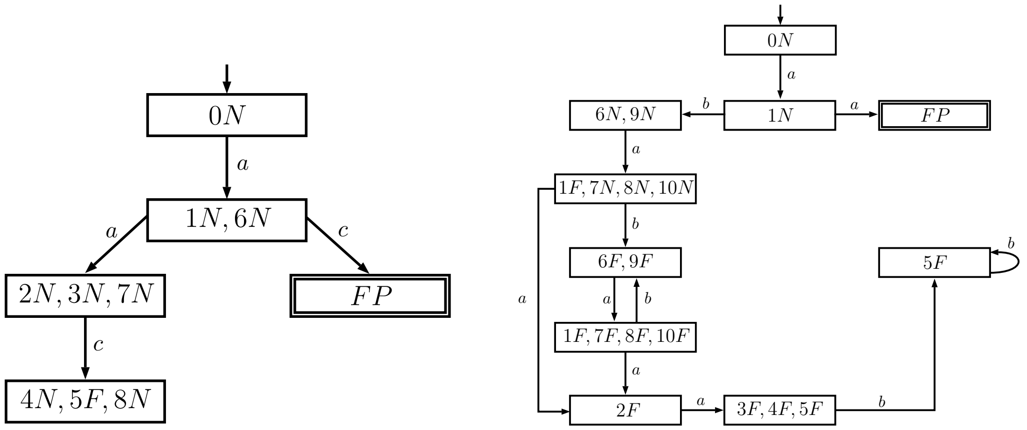
The next result shows that every fault sequence $s_f \in \Psi(\sigma_f)$ is foreseen by some local predictor automaton $G_{lp,i}$, which is obtained in accordance with Algorithm 5.3.

Theorem 5.3 *If the language L generated by an automaton G is copredictable with respect to projections $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i = 1, \ldots, n$, and fault event set $\Sigma_f = \{\sigma_f\}$, then, $\forall s_f \in \Psi(\sigma_f), \exists i \in [1, \ldots, n]$ such that $\mathcal{L}_m(G_{lp,i}) \cap \overline{P_{o,i}(s_f)} \neq \emptyset$ and $\mathcal{L}_m(G_{lp,i}) \cap P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}}) = \emptyset$.*

Proof.

Lemma 5.3 ensures that local predictors constructed in accordance with Algorithm 5.3 successfully predict all faults supposed to be predicted given the available observation. It remains to prove that for every fault sequence $s_f \in \Psi(\sigma_f)$, there exists a local predictor LP_i that foresees its occurrence. We prove that by contradiction, and, to this end, assume that L is copredictable with respect to $P_{o,i}$, $i = 1, \ldots, n$, and Σ_f, but $\exists s_f \in \Psi(\sigma_f), \forall i \in [1, \ldots, n]$ such that either $\mathcal{L}_m(G_{lp,i}) \cap \overline{P_{o,i}(s_f)} = \emptyset$ or $\mathcal{L}_m(G_{lp,i}) \cap P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}}) \neq \emptyset$. It is worth noticing that: *(i)* $L \cap \overline{\Psi(\sigma_f)\Sigma^*}$ is formed by all prefixes of all arbitrarily long length fault sequences, which implies that $L \setminus \overline{\Psi(\sigma_f)\Sigma^*}$ is composed of normal sequences that have only normal sequence continuations; thus, $\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}}$ recovers all prefixes of the sequences of $L \setminus \overline{\Psi(\sigma_f)\Sigma^*}$ that were removed. By construction, $\mathcal{L}_m(G_{lp,i})$ is formed by the projections of all normal sequences that have the same projection $P_{o,i}$ as some sequence of L that is a prefix of some fault sequence that LP_i is capable of predicting. Since language L is copredictable, it is clear that for some $i = 1, \ldots, n$, we have that, according to Algorithm 5.1 and Lemma 5.3, $P_{o,i}(s_f) \notin P_{o,i}(\mathcal{L}_m(G_N))$, and thus, according to Algorithm 5.3, the smallest sequence $s_p = s\sigma \in \overline{P_{o,i}(s_f)}$, where $s \in P_{o,i}(\mathcal{L}_m(G_N))$ and $s\sigma \notin P_{o,i}(\mathcal{L}_m(G_N))$ is such that $s_p \in \mathcal{L}_m(G_{lp,i})$. Notice that we may even have $s = \varepsilon$. We then conclude that $s_p \in \mathcal{L}_m(G_{lp,i}) \cap \overline{P_{o,i}(s_f)}$, and so, $\mathcal{L}_m(G_{lp,i}) \cap \overline{P_{o,i}(s_f)} \neq \emptyset$. In addition, according to Definition 5.1 and Algorithm 5.3, for all sequence t such that $P_{o,i}(t) \in \mathcal{L}_m(G_{lp,i})$, we have that $\big(\exists z \in \mathbb{N}\big)\big(\forall u_i \in L\big)\big(\forall v_i \in L/u_i\big)\big[(P_{o,i}(u_i) = P_{o,i}(t)) \wedge (\sigma_f \notin u_i) \wedge (\|v_i\| \geq z) \Rightarrow (\sigma_f \in v_i)\big]$ holds true, and so, there is no arbitrarily long length normal sequence $s_{N,i} = t_{N,i}u_{N,i}$ such that $P_{o,i}(t_{N,i}) = P_{o,i}(t)$, which implies that $P_{o,i}(t) \notin \mathcal{L}_m(G_{lp,i}) \cap P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})$; thus contradicting our initial assumption. ■

Example 5.7 *Consider automata G, depicted in Figure 5.3, whose language L is copredictable with respect to $\Sigma_f = \{\sigma_f\}$, $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$. By using automaton G_1^N, shown in Figures 5.6a, as an input of Algorithm 5.3,*



(a) Automaton $G_{lp,1}$. (b) Automaton $G_{lp,2}$.

Figure 5.12: Fault predictors.

we construct automaton $G_{lp,1}$ as follows. First, in Step 1, *we form set $Q = \{\{9N\}, \{9F\}, \{1F, 6F, 10N\}, \{1F, 6F, 10F\}, \{2F, 3F, 7F\}, \{4F, 5F, 8F\}\}$ composed of all unmarked states of G_1^N. In* Step 2, *we add a new state FP. At the end of* Step 3, *the only marked state that is immediately succeeded by an unmarked state and has at least one element labeled with N, is state $\{9N\}$, which is connected to the marked state $\{1N, 6N\}$ through a transition labeled with event c, and so, this transition is removed and a new one connecting state $\{1N, 6N\}$ to $\{FP\}$ and labeled with event c is created. In* Step 4, *state $\{FP\}$ becomes the only marked state. Finally, the resulting automaton $G_{lp,1} = Ac(G_1^N)$ is obtained in* Step 5, *being depicted in Figure 5.12(a). Analogously, by using automaton G_2^N as input of Algorithm 5.3, we obtain automaton $G_{lp,2}$, shown in Figure 5.12(b). It is worth remarking that the marked languages $\mathcal{L}_m(G_{lp,1}) = ac$ and $\mathcal{L}_m(G_{lp,2}) = aa$ concur with the results presented in Examples 5.5 and 5.6.* □

Notice that, when a language is copredictable, if some fault sequence is not foreseen by a local fault predictor system, then it is certainly detected by another local fault predictor. For example, for LP_1, which is modeled by automaton $G_{lp,1}$, even though the observation $s_{o,1} = aac$ may correspond to the faulty sequence $s = aabc\sigma_f$, which cannot be predicted by LP_1, it is foreseen by LP_2, since $P_{o,2}(aabc\sigma_f) = aab$ and when LP_2 observes its prefix $s_{o,2} = P_{o,2}(aa) = aa$, state FP is reached in automaton $G_{lp,2}$.

Remark 5.7 *(Computational complexity) All steps of Algorithm 5.3 are linear, and thus, its computational complexity is $O(|X_i|)$. According to* Step 6 *of Algorithm 5.1, automaton $G_i^N = Obs(G_N, \Sigma_{o,i})$, where G_N is a copy of G_ℓ with different marked states, and so, $|X_i| = 2^{|X_\ell|} = 2^{2|X|}$, which implies that Algorithm 5.3 is $O(2^{2|X|})$.* □

5.4 K-copredictability verification

In this section, given that a language is copredictable with respect to $P_{o,i}$, $i = 1, \ldots, n$, and Σ_f, and given an integer K, we consider the problem of verifying if all fault occurrences can be predicted at least K events prior to their occurrences. To this end, we investigate the problem of finding the minimal number of events between the fault prediction by a local fault predictor LP_i and its actual occurrence. We refer to this problem as K-copredictability, whose formal definition is as follows.

Definition 5.3 *(K-Copredictability) Let L be a prefix-closed and live language copredictable w.r.t. $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i \in I = \{1, 2, \ldots, n\}$ and event set $\Sigma_f = \{\sigma_f\}$. Given an integer K, we say that L is K-copredictable if*

$$(\forall s_f \in \Psi(\sigma_f))(\exists i \in I)\big(P_{o,i}(s_f) \notin P_{o,i}(L \setminus \overline{\Psi(\sigma_f)\Sigma^*}) \to \mathbf{K_c}\big),$$

where the K-Copredictability condition $\mathbf{K_c}$ is given by

$$(\exists t, u \in \Sigma^*)[(s_f = tu\sigma_f) \wedge (P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})) \wedge (\|u\| \geq K)]. \qquad \square$$

As explained in the proof of Theorem 5.3, $L \setminus \overline{\Psi(\sigma_f)\Sigma^*}$ is formed with all normal sequences of L that are continued with normal sequences only. Thus, if for all fault sequences $s_f \in \Psi(\sigma_f)$, there exists a local predictor where the projection $P_{o,i}(s_f)$ differs from those of sequences in $L \setminus \overline{\Psi(\sigma_f)\Sigma^*}$, then, in order for L to be K-copredictable, all sequences that end with fault event σ_f must be formed by the concatenation of two sequences: a sequence t that cannot be confused with any normal sequence with normal continuations only (meaning that the fault occurrence has been predicted), and; a sequence u where $\|u\| \geq K$, which ensures that at least K events will occur after the fault prediction and before the actual occurrence of the fault.

In this regard, Definition 5.3 states that when a language is K-copredictable, if the fault occurrence in sequence s_f is predicted by some local predictor LP_i, then there exists a prefix $t \in \overline{s_f}$ whose observation is sufficient for LP_i to predict the fault occurrence and there exist at least K events between the fault prediction and its occurrence.

In order to illustrate Definition 5.3, let us recall the language generated by automaton G depicted in Figure 5.3, which is copredictable. According to Definition 5.3, it is clear that sequence $aabc\sigma_f$, which is predicted by LP_2 only (see Figure 5.12(b)), satisfies 2-copredictability, since the smallest prefix of s_f whose observation allows LP_2 to predict the fault occurrence is $t = aa$, which defines $u = bc$ whose length is 2. On the other hand, sequence $aabc\sigma_f$ does not satisfy 3-copredictability, since in order for $\|u\| = 3$, we must set $t = a$ but $P_{o,2}(t) = a \in P_{o,2}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}}) = \overline{aba}$, which violates Definition 5.3.

We now present an algorithm for the verification of K-copredictability. To this end, let G be the automaton that models the system and $G_{lp,i}$, $i = 1, \ldots, n$, the local fault predictor automata obtained in accordance with Algorithm 5.3. The idea of the verification strategy we propose here is to identify the states of automaton G that correspond to the fault prediction state $\{FP\}$ and then count the number of events after this state up to the first fault event. The whole procedure is given in Algorithm 5.4.

Algorithm 5.4 *K-Copredictability Verification*

```
Input: K, Automata $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ and $G_{lp,i} = (X_{lp,i}, \Sigma_{o,i}, f_{lp,i}, \Gamma_{lp,i}, x_{0,lp,i}, X_{m,lp,i})$, $i = 1, \ldots, n$.
Output: K-copredictability decision: Yes or No.
  1. Set $\widetilde{G}_{lp,i} = G_{lp,i} = (\widetilde{X}_{lp,i}, \Sigma_{o,i}, \widetilde{f}_{lp,i}, \widetilde{\Gamma}_{lp,i}, \widetilde{x}_{0_{lp,i}}, \widetilde{X}_{m_{lp,i}})$ for all $i = 1, \ldots, n$.
  2. For all $\widetilde{G}_{lp,i}$: set $\widetilde{\Gamma}_{lp,i}(\{FP\}) = \Sigma_{o,i}$ and define transition $\widetilde{f}_{lp,i}(\{FP\}, \sigma_{o,i}) = \{FP\}$ for all $\sigma_{o,i} \in \Sigma_{o,i}$.
  3. Set $G_K = \left(\|_{i=1}^{n} \widetilde{G}_{lp,i}\right) \| G = (X_K, \Sigma_K, f_K, \Gamma_K, x_{0,K}, X_{m,K})$.
  4. Set $G_{K_{test}} = (X_{K_t}, \Sigma_{K_t}, f_{K_t}, \Gamma_{K_t}, x_{0_{K_t}}, X_{m_{K_t}}) = G_K$.
  5. Create a new state $y_0$, set $X_{K_t} = X_K \cup \{y_0\}$ and redefine $x_{0_{K_t}} = y_0$.
  6. Set $Y = \emptyset$ and for all $x_{K_t} = (x_{lp,1}, \ldots, x_{lp,n}, x) \in X_{K_t}$, if $\exists i = 1, \ldots, n$ : $x_{lp,i} = \{FP\}$ and $\nexists (x'_{K_t}, \sigma) \in X_{K_t} \times \Sigma$, where $x'_{K_t} = (x'_{lp,1}, \ldots, x'_{lp,n}, x')$, such that $f_{K_t}(x'_{K_t}, \sigma) = x_{K_t}$ and $x'_{lp,i} = \{FP\}$ for some $i = 1, \ldots, n$, then set $Y = Y \cup \{x_{K_t}\}$.
  7. Build $\widetilde{\Sigma} = \{\sigma_i : i = 1, \ldots, |Y|\}$ and set $\Sigma_{K_t} = \Sigma_K \cup \widetilde{\Sigma}$.
  8. For each $(y_i, \sigma_i) \in Y \times \widetilde{\Sigma}$ set $f_{K_t}(y_0, \sigma_i) = y_i$.
  9. For all $(x_{K_t}, \sigma_f) \in X_{K_t} \times \Sigma_f$: if $f_{K_t}(x_{K_t}, \sigma_f) = x'_{K_t}$ for some $x'_{K_t} \in X_{K_t}$, then set $X_{m,K_t} = X_{m,K_t} \cup \{x_{K_t}\}$ and remove transition $(x_{K_t}, \sigma_f, x'_{K_t})$.
  10. Set $G_{K_{test}} = Ac(G_{K_{test}})$.
  11. If $\min_{s \in \mathcal{L}_m(G_{K_{test}})} \{\|s\|\} - 1 \geq K$, then return "Yes". Otherwise, return "No".
```

Algorithm 5.4 works as follows. In STEP 1 we set each automaton $\widetilde{G}_{lp,i}$ as a copy of $G_{lp,i}$, $i = 1, \ldots, n$. Then, in STEP 2, we define a self-loop in state $\{FP\}$ with all locally observable events $\sigma_{o,i} \in \Sigma_{o,i}$ for all automaton $\widetilde{G}_{lp,i}$, so that, in STEP 3, the language $L = \mathcal{L}(G)$ is not harmed when we compute $G_K = \left(\|_{i=1}^{n} \widetilde{G}_{lp,i}\right) \| G$. In STEP 4 we create automaton $G_{K_{test}}$ as a copy of G_K, used to perform the K-copredictability verification. In STEP 5 we add a new state y_0 to $G_{K_{test}}$ and then set it as the initial state. In STEP 6 we search for all states $x_{K_t} \in X_{K_t}$ such that one of their components is $\{FP\}$ and do not immediately succeed some state x'_{K_t} that also has $\{FP\}$ as a component, and store these states in set Y. In Step 7, we create events σ_i, $i = 1, \ldots, |Y|$, and add them to Σ_{K_t}. In Step 8 we create new transitions $f_{K_t}(y_0, \sigma_i) = y_i$ for all $y_i \in Y$. In Step 9, we mark the states where the transitions labeled by the fault originate and then we remove all fault events from automaton $G_{K_{test}}$. In Step 10, we take the accessible part of automaton $G_{K_{test}}$. Finally, we calculate the minimum number of events from the initial to all marked states. This means that the fault will occur after being predicted in at least $K_{min} = \underset{s \in \mathcal{L}_m(G_{K_{test}})}{min} \{\|s\|\} - 1$ events. If $K_{min} \geq K$, then Step 10 returns "*Yes*", meaning that language L is K-copredictable; otherwise, it returns "*No*".

It is not difficult to see that the marked language of automaton $G_{K_{test}}$ obtained in accordance with Algorithm 5.4 is composed of sequences $s = \widetilde{\sigma} u$, where $\widetilde{\sigma}$ is the event added to $G_{K_{test}}$ to label the transition that connects the new initial state y_0 and those states that have one of their components being $\{FP\}$ and do not immediately succeed states x'_{K_t} that also have $\{FP\}$ as a component, and u is the continuation of a sequence t that predicts a fault from the perspective of some local predictor and u immediately precedes a fault occurrence σ_f. This result is formally presented as follows.

Lemma 5.4 *Let L be a prefix-closed and live language generated by automaton G, copredictable with respect to $P_{o,i} : \Sigma^* \to \Sigma^*_{o,i}$, $i \in I = \{1, 2, \ldots, n\}$ and event set $\Sigma_f = \{\sigma_f\}$, and let $G_{K_{test}}$ be the automaton obtained in accordance with Algorithm 5.4. Then, $\big(\forall s = \widetilde{\sigma} u \in \mathcal{L}_m(G_{K_{test}}), (\widetilde{\sigma} \in \widetilde{\Sigma}) \wedge (u \in \Sigma^*)\big)(\exists i \in I)(\exists t \in L)\big[\big(tu\sigma_f \in \Psi(\sigma_f)\big) \wedge \big(P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})\big)\big]$.*

Proof.
By construction, if $s \in \mathcal{L}_m(G_{K_{test}})$, then $\exists \widetilde{\sigma} \in \widetilde{\Sigma}$ and $u \in \Sigma^*$ such that $s = \widetilde{\sigma} u$. Therefore, $(\exists x_{m_{K_t}} \in X_{m_{K_t}})[f_{K_t}(y_0, s) = x_{m_{K_t}}]$, and since $X_{m_{K_t}} \subseteq X_K$ we can conclude that a transition $f_{K_t}(x_{m_{K_t}}, \sigma_f)$ has been removed (see Step 9). According to Step 8, it is clear that state $x_{K_t} = (x_{lp,1}, \ldots, x_{lp,n}, x)$ reached by event $\widetilde{\sigma}$ is such that $x_{K_t} \in Y$, and thus, according to Step 6, $(\exists i = 1, \ldots, n)[x_{lp,i} = \{FP\}]$ and $\nexists (x'_{K_t}, \sigma) \in X_{K_t} \times \ \Sigma$ and $\nexists i = 1, \ldots, n$ such that $f_k(x'_{K_t}, \sigma) = x_{K_t}$ and $x'_{lp,i} = \{FP\}$, which means that state x_{K_t} has at least one component $\{FP\}$ and is not preceded by states with $\{FP\}$ as some of its components. Thus, associated with $s = \widetilde{\sigma} u \in \mathcal{L}_m(G_{K_{test}})$, there exists a sequence $s' = tu\sigma_f \in \mathcal{L}(G_K)$. Let $x_K \in X_K$ be a state such that $f_K((x_{0,lp,1}, \ldots, x_{0,lp,n}, x_0), t) = x_K$. Since one of the elements of x_{K_t} is $\{FP\}$, it is clear that $\exists i = 1, \ldots, n$ such that $P_{o,i}(t) \in \mathcal{L}_m(G_{lp,i})$, where automaton $G_{lp,i}$ is one of the inputs of Algorithm 5.4. Since $L(\widetilde{G}_{lp,i})$ is obtained after removing sequences of $L(G_i^N)$ and then concatenating the "pruned sequences" with $\Sigma^*_{o,i}$, we have that $\mathcal{L}(G_i^N) \subseteq \mathcal{L}(\widetilde{G}_{lp,i})$. It is worth remarking that

$L = \mathcal{L}(G_{scc}) = \mathcal{L}\big(\|_{i=1}^{n} G_i^N\big) \cap \mathcal{L}(G)$ whereas $\mathcal{L}(G_K) = \mathcal{L}\big(\|_{i=1}^{n} \widetilde{G}_{lp,i}\big) \cap L \subseteq L$, and so, $L = \mathcal{L}(G_{scc}) \subseteq \mathcal{L}(G_K) \subseteq L$, which implies that $\mathcal{L}(G_K) = L$. Therefore, sequence $tu\sigma_f \in L$, and as a consequence, $tu\sigma_f \in \Psi(\sigma_f)$. Finally, since L is copredictable, then, according to Theorem 5.3, we can conclude that $\exists i \in \{1, \ldots, n\}$ such that $P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})$. ■

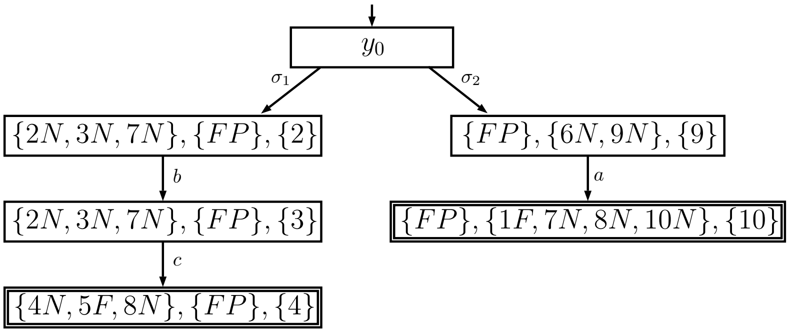
We now present a necessary and sufficient condition for K-copredictability verification based on automaton $G_{K_{test}}$, constructed in accordance with Algorithm 5.4.

Theorem 5.4 *Let L be a prefix-closed and live language copredictable with respect to $P_{o,i} : \Sigma^* \to \Sigma_{o,i}^*$, $i = 1, \ldots, n$, and event set $\Sigma_f = \{\sigma_f\}$, and let $G_{K_{test}}$ be the automaton obtained in accordance with Algorithm 5.4. Given an integer $K \in \mathbb{N}$, then L is K-copredictable if, and only if, $\underset{s \in \mathcal{L}_m(G_K)}{min} \{\|s\|\} - 1 \geq K$.*

Proof.
($\Rightarrow$) Assume that $\underset{s \in \mathcal{L}_m(G_{K_{test}})}{min} \{\|s\|\} - 1 < K$. First, notice that $\mathcal{L}_m(G_{K_{test}})$ is composed of sequences $\widetilde{\sigma}u$, where $\widetilde{\sigma} \in \widetilde{\Sigma}$ and $u \in \Sigma^*$, and so, according to Lemma 5.4, $(\exists i \in I)$ $(\exists t \in L)$ $[tu\sigma_f \in \Psi(\sigma_f) \wedge P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})]$. Let us define $s_f = tu\sigma_f$. Notice that, there exists at least a sequence u such that $\|u\| < K$, where u is the longest sufix of t such that $P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})$ holds true, therefore, $\forall t', u' \in \Sigma^*$ such that $s_f = t'u'\sigma_f$, either $\|u'\| < K$ or $P_{o,i}(t) \in P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})$, which implies that L is not K-copredictable.

($\Leftarrow$) Assume that L is not K-copredictable, and thus $(\exists s_f \in \Psi(\sigma_f))(\forall i \in I)\big[\big(P_{o,i}(s_f) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})\big) \wedge \big(\forall t, u \in \Sigma^*, (s_f \neq tu\sigma_f) \vee (\|u\| < K) \vee (P_{o,i}(t) \in P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}}))\big)\big]$. Since L is copredictable, consider without loss of generality a fault sequence $s_f = tu\sigma_f$ such that for all $i = 1, \ldots, n$, either s_f is not predictable or $P_{o,i}(t) \notin P_{o,i}(\overline{L \setminus \overline{\Psi(\sigma_f)\Sigma^*}})$. Considering the locals i where s_f is predictable, and since, by assumption, L is not K-copredictable, sufix u must be such that $\|u\| < K$. According to Algorithm 5.3 and Theorem 5.3, prefix t reaches state $\{FP\}$ in automaton $G_{lp,i}$. After computing $G_{K_{test}}$ in accordance with Steps 1-5, state x_{K_t} reached by sequence t is added to set Y in Step 6, which is connected to the new initial state y_0 through event $\widetilde{\sigma}$ in Steps 7-8, which results in a new sequence $\widetilde{\sigma}u\sigma_f \in \mathcal{L}(G_{K_{test}})$, which is later pruned in Step 9 and becomes the sequence $\widetilde{\sigma}u \in \mathcal{L}_m(G_{K_{test}})$. Thus, $\underset{s \in \mathcal{L}_m(G_{K_{test}})}{min} \{\|s\|\} - 1 \leq \|\widetilde{\sigma}u\| - 1 = \|u\| < K$. ■

Example 5.8 *Consider automaton G, depicted in Figure 5.3, whose language L is copredictable with respect to $\Sigma_f = \{\sigma_f\}$, $\Sigma_{o,1} = \{a, c\}$ and $\Sigma_{o,2} = \{a, b\}$, and automata $G_{lp,1}$ and $G_{lp,2}$ obtained in accordance with Algorithm 5.3. In order to compute automaton $G_{K_{test}}$ according to Algorithm 5.4, first, in* Steps *1-2, we create automaton $\widetilde{G}_{lp,1}$ (resp, $\widetilde{G}_{lp,2}$) as a copy of $G_{lp,1}$ (resp, $G_{lp,2}$) and add*



y_0
σ_1
σ_2
$\{2N, 3N, 7N\}, \{FP\}, \{2\}$
$\{FP\}, \{6N, 9N\}, \{9\}$
b
a
$\{2N, 3N, 7N\}, \{FP\}, \{3\}$
$\{FP\}, \{1F, 7N, 8N, 10N\}, \{10\}$
c
$\{4N, 5F, 8N\}, \{FP\}, \{4\}$

Figure 5.13: Automaton $G_{K_{test}}$.

a self-loop with events a, c (resp, a, b) to state $\{FP\}$. We then build $G_{K_{test}} = \widetilde{G}_{lp,1} \| \widetilde{G}_{lp,2} \| G$ in STEPS 3-4. *In* STEP 5, *we add a new state $\{y_0\}$ to $G_{K_{test}}$ and set it as the initial state. In* STEP 6, *only states $(\{2N, 3N, 7N\}, \{FP\}, \{2\})$ and $(\{FP\}, \{6N, 9N\}, \{9\})$ are stored in set Y. In* STEP 7, *since $|Y| = 2$, we define events σ_1 and σ_2, which form set $\widetilde{\Sigma} = \{\sigma_1, \sigma_2\}$. In* STEP 8, *we connect the initial state $\{y_0\}$ to the states in Y with the events in $\widetilde{\Sigma}$. In* STEP 9, *all fault events are removed and states $(\{4N, 5F, 8N\}, \{FP\}, \{4\})$ and $(\{FP\}, \{1F, 7N, 8N, 10N\}, \{10\})$ become marked. Finally, in* STEP 10, *we take its accessible part, resulting in automaton $G_{K_{test}}$, depicted in Figure 5.13. From Figure 5.13, it is clear that every local predictor foresees the fault occurrence at least $K = min\{\|\sigma_1 bc\|, \|\sigma_2 a\|\} - 1 = 1$ event prior to the fault occurrence, and thus, according to* STEP 11, *language L is 1-copredictable.* □

Remark 5.8 *(Computational complexity) The computational complexity of Algorithm 5.4 lies in* STEP 3, *where we compute $G_K = \left(\|_{i=1}^{n} \widetilde{G}_{lp,i}\right) \| G$. Since $|\widetilde{X}_{lp,i}| = |X_{lp,i}|$, the computational complexity for building $\widetilde{G}_{lp,i}$ is identical to the one of $G_{lp,i}$, which is $O(2^{2|X|})$ according to Algorithm 5.3 and Remark 5.7. Thus, Algorithm 5.4 is $O(|X| \times 2^{2n|X|})$.* □

5.5 Comparison between the verification methods proposed here and by KUMAR and TAKAI [85]

In this section we compare the verifier automata obtained in Examples 1, 2, and 4 and in Section VII of KUMAR and TAKAI [85] with the verifier and diagnoser-based test automata proposed in this chapter.

Differently from this work, where the fault behavior starts with the occurrence of the fault event σ_f, the approach by KUMAR and TAKAI [85] is based on the predictability of sequences that are not in a non-failure specification language $K \subset L$. By assuming that $G = (X, \Sigma, f, x_0)$ and $G' = (X', \Sigma, f', x'_0)$ are the automata

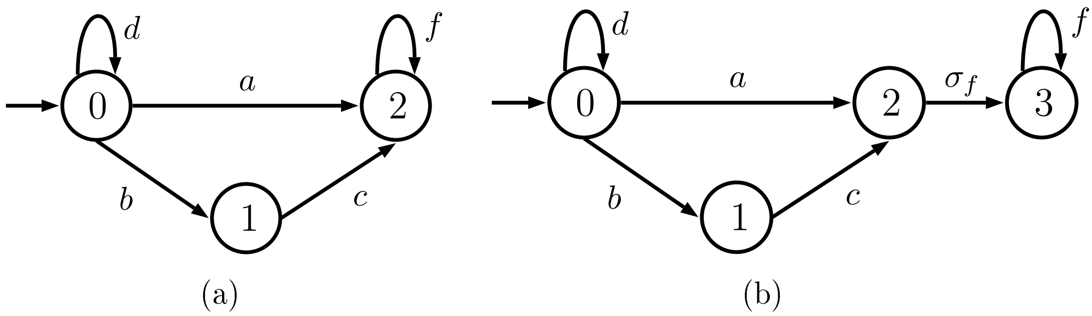


Figure 5.14: Automaton G of Example 1 of [85] (a) and its equivalent automaton G_{eq} (b).

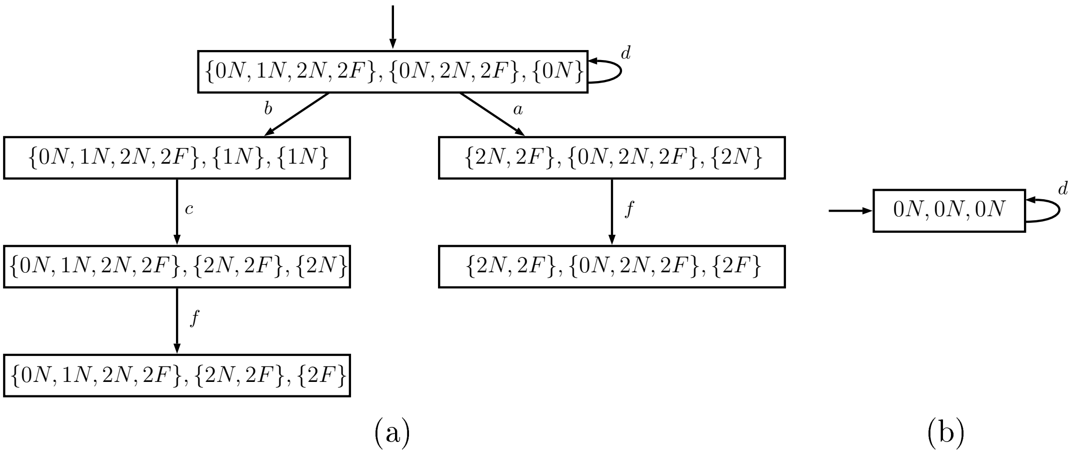

$\{0N, 1N, 2N, 2F\}, \{0N, 2N, 2F\}, \{0N\}$
d
b
a
$\{0N, 1N, 2N, 2F\}, \{1N\}, \{1N\}$
$\{2N, 2F\}, \{0N, 2N, 2F\}, \{2N\}$
c
f
$\{0N, 1N, 2N, 2F\}, \{2N, 2F\}, \{2N\}$
$\{2N, 2F\}, \{0N, 2N, 2F\}, \{2F\}$
f
$\{0N, 1N, 2N, 2F\}, \{2N, 2F\}, \{2F\}$
$0N, 0N, 0N$
d
(a)
(b)

Figure 5.15: Test automaton G_{scc} (a) and verifier automaton G_v (b) for Example 1 of [85].

whose generated language are L and K, respectively, we can build an equivalent automaton $G_{eq} = (X_{eq}, \Sigma \cup \{\sigma_f\}, f_{eq}, x_{0,eq})$ to G, whose normal sequences form the non-failure specification language K and $P'(\mathcal{L}(G_{eq})) = L$, where $P' : (\Sigma \cup \{\sigma_f\})^* \rightarrow \Sigma^*$ (see CARVALHO *et al.* [115, Alg. 4]).

Figure 5.14(a) shows automaton G of KUMAR and TAKAI [85, Example 1], where $\Sigma_{o,1} = \{a, d\}$ and $\Sigma_{o,2} = \{b, c, d\}$. For this example, the non-failure specification language is $K = \overline{d^*(a + bc)}$. Figure 5.14(b) depicts the equivalent automaton G_{eq}, where a new state (state 3) and a transition between states 2 and 3 labeled by the fault event σ_f have been introduced in order to make G appropriate to the set up of Algorithms 5.1 and 5.2. Notice that the normal sequences of automaton G_{eq} are the same as those of the non-failure specification language K. Figures 5.15(a) and 5.15(b) show the test automaton G_{scc} and the verifier automaton G_v, constructed in accordance with Algorithms 5.1 and 5.2, respectively. Notice that, as shown in the first row of Table 5.1, the verifier $T = (X_T, \Sigma, f_T, x_{0,T}, \emptyset)$ proposed in [85] used to check the copredictability of the language L generated by G has 17 states and 45 transitions, whereas the diagnoser-based test automaton G_{scc}, built according to Algorithm 5.1, has 6 states and 6 transitions and the verifier automaton G_v, con-

Table 5.1: Comparison between automata T, G_{scc} and G_v, where $|T_T|$, $|T_{scc}|$ and $|T_v|$ (resp. $|X_T|$, $|X_{scc}|$ and $|X_v|$) denote the number of transitions (resp. states) of automata T, G_{scc} and G_v, respectively.

	G	T		G_{eq}	G_{scc}		G_v	
	$\|X\|$	$\|X_T\|$	$\|T_T\|$	$\|X_{eq}\|$	$\|X_{scc}\|$	$\|T_{scc}\|$	$\|X_v\|$	$\|T_v\|$
Ex. 1	3	17	45	4	6	6	1	1
Ex. 2	5	43	178	6	6	6	7	14
Ex. 4	8	88	500	9	10	10	16	37
Sec. VII	7	107	643	8	8	8	32	80

structed according to Algorithm 5.2, has only 1 state and 1 transition. Therefore, for this example, the automata proposed here for copredictability verification have much less states and transitions than that proposed in [85]. Similar results have been obtained for Examples 2, 4 and the one of Section VII of KUMAR and TAKAI [85], as seen in rows 2–4 of Table 5.1.

It is worth remarking that these results are expected for the following reasons: (i) regarding the diagnoser-based test automata G_{scc}, as conjectured by CLAVIJO and BASILIO [106], diagnoser-based automata are likely to have, on average, fewer states than verifiers for systems with 60 events or less; (ii) regarding the verifiers, although, as stated in Remark 5.6, the verifiers proposed here and by KUMAR and TAKAI [85] have the same worst-case computational complexity regarding the number of states of the system, verifier G_v, obtained in accordance with Algorithm 5.2, is expected to have fewer states on average, since $G_v = \left(\|_{i=1}^{n} G_{N,i}\right) \| G_f$ and both G_f and each one of the automata $G_{N,i}$, $i = 1, \ldots, n$, are obtained by removing states and transitions from automaton G_ℓ and stops when ambiguities cease to exist, whereas verifier T generates the complete language, including those sequences whose continuations lie in specification K.

5.6 Concluding remarks

In this chapter, we have revisited the problem of disjunctive fault predictability of DESs. Two algorithms for its verification have been proposed: the first algorithm is based on a test automaton [102] and on the search for nontrivial SCCs, whereas the second algorithm is based on verifiers [104]. We have provided a procedure for designing fault predictor systems, which can be used for online fault prediction. We have also addressed the problem of K-copredictability of DESs and presented a necessary and sufficient conditions for K-copredictability and an algorithm for its verification. These approaches have been developed without requiring either language liveness or absence of state cycles connected by unobservable events only.

Finally, a comparison between the automata proposed here for copredictability verification and that proposed by KUMAR and TAKAI [85] has shown that the verification automata built in accordance with the algorithms proposed here are likely to have fewer states than that by KUMAR and TAKAI [85]; however, an additional work, similar to that performed in [106], is necessary to make a formal claim regarding the results obtained in the comparison performed in the chapter.

Chapter 6

Conclusion and future works

6.1 Conclusion

In this section we summarize all of the contributions given by this thesis.

C1. Current-state opacity enforcement in DES

In Chapter 3, we provided a strategy that realizes the so-called Opacity-Enforcer by leveraging the possibility of delaying and deleting some event observations. The Opacity-Enforcer manipulates the event observations outputted by the system with a view to misleading the Intruder to never estimate secret states.

The proposed Opacity-Enforcer keeps track not only of the events executed by the system, but also of the release and deletion of their observation signals, and has shown to have the potential to be used to enforce opacity and, at the same time, not mislead the legitimate receiver as much as the intruder. In this regard, by assuming that the legitimate receiver is aware of the actions taken by the Opacity-Enforcer, we presented a protocol that refines the current-state estimates of the system by the legitimate receiver, hence, mitigating the negative effect of the opacity enforcement on its estimation capability.

C2. Ensuring utility while enforcing CSO in DES

The second contribution given by this work is presented in Chapter 4 and was motivated by the fact that, in order to obfuscate the secret behavior of the system from intruders, some transmitted information must also be concealed from the legitimate receiver. To this end, the notion of utility (which differs from that presented in [71], as discussed in Remark 4.1) is introduced. Moreover, we have presented sufficient and necessary conditions that preserves utility while enforcing current-state opacity.

The algorithm presented in Chapter 3 for realizing the Opacity-Enforcer has also been simplified. The new algorithm obtained in Chapter 4 realizes an Opacity-Enforcer that now ensures the utility of the system and, at the same time, enforces current-state opacity, whenever the behavior of the system allows.

C3. Fault copredictability in DES

In Chapter 5, we revisited the problem of disjunctive fault predictability of DESs, also known as fault copredictability. Two algorithms for its verification have been proposed: the first algorithm is based on a test automaton [102], whereas the second is based on a verifier automaton [104].

Although the results presented in Chapter 5 and those presented in KUMAR and TAKAI [85] may seem alike, as far as the computational complexity is concerned, the verification automata built in accordance with Algorithms 5.1 and 5.2 proposed here are likely to have fewer states than that by KUMAR and TAKAI [85]. However, an additional work, similar to that performed in CLAVIJO and BASILIO [106], is necessary to make a formal claim regarding the results obtained in the comparison performed in Section 5.5.

Moreover, we provided a procedure for designing local fault predictor systems, which can be used for online fault prediction. The problem of K-copredictability of DESs has also been addressed, where a necessary and sufficient conditions for K-copredictability and an algorithm for its verification have been given. All of the results obtained do not require either language liveness or absence of state cycles connected by unobservable events only.

6.2 Future works

We now present a brief idea for some future works, regarding both opacity and fault predictability, so as to extend the results provided throughout this thesis. The first idea (**FW1**) is to address opacity and fault predictability in time-weighted DES. The second suggestion (**FW2**) is to propose notions of robust fault copredictability and networked fault copredictability, and also to develop strategies for their verifications. We discuss these ideas with more details in the sequel.

FW1. Time-weighted and real-time DES

Time-weighted systems have been proposed by SU *et al.* [116], where the problem of synthesizing a minimum-makespan supervisor is solved. A finite time-weighted automaton is a 2-tuple $\mathcal{G} = (G, w)$, where G is the finite-state automaton and w is the weighting function that assigns a non-negative weight to each transition of

G. As shown by VIANA *et al.* [114], it is possible to obtain an equivalent untimed model for this class of automaton.

With respect to opacity, the idea is to use the restrictions imposed by the time-weighted automaton on the equivalent untimed model to generate feasible CSO enforcers that also take into account the time elapsed between event occurrences. Notice however that WANG *et al.* [41] have approached the problems of LBO and ISO for a timed class of automata, the so-called real-time automata [117], which may have some similarity with time-weighted automata, and thus, requires a deeper investigation.

In addition, fault (co)prediction for time-weighted automata has not been explored as far as the author of this work knows. Notice that, since the weighting function w adds information to the model, the state estimation can be improved, and, as a consequence, it may be the case that system is not predictable with respect to fault occurrences when modeled by automaton G but its time-weighted version $\mathcal{G}$ is predictable.

FW2. Robust fault copredictability

We have provided, in this work, a strategy for disjunctive fault predictability verification in DESs under the assumption that communication channels are ideal. However, it is not uncommon to have disruptions in these channels, which may cause intermittent or even permanent losses of observation. It is worth noting that the problem of robust fault predictability against intermittent/permanent loss of observations has already been addressed by XIAO and LIU [83]. However, the results presented here can be combined with the strategies presented in [3–5, 115] in order to obtain a new strategy capable of verifying robust fault predictability against intermittent/permanent loss of observations under disjunctive architectures (in short words, robust fault copredictability against observation loss). In addition, the architecture presented in Chapter 5 can be extended to networked DES, where some communication channels may be vulnerable to cyber-attacks and the observation of events may be delayed, as done in [17], in the context of fault diagnosability.

References

[1] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Events Systems.* 2nd ed. New York, NY, USA, Springer, 2008.

[2] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. “Generalized robust diagnosability of discrete event systems”, *IFAC Proceedings Volumes*, v. 44, n. 1, pp. 8737–8742, 2011.

[3] 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.

[4] 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.

[5] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. “Diagnosability of intermittent sensor faults in discrete event systems”, *Automatica*, v. 79, pp. 315–325, 2017.

[6] NUNES, C. E., MOREIRA, M. V., ALVES, M. V., et al. “Codiagnosability of networked discrete event systems subject to communication delays and intermittent loss of observation”, *Discrete Event Dynamic Systems*, v. 28, n. 2, pp. 215–246, 2018.

[7] HADJICOSTIS, C. N. *Estimation and Inference in Discrete Event Systems.* New York, NY, USA, Springer, 2020.

[8] ALVES, M. V. S., BASILIO, J. C. “State Estimation and Detectability of Networked Discrete Event Systems with Multi-Channel Communication Networks”. In: *2019 American Control Conference (ACC)*, pp. 5602–5607, 2019. doi: 10.23919/ACC.2019.8814302.

[9] RAMADGE, P. J., WONHAM, W. M. “The control of discrete event systems”, *Proceedings of the IEEE*, v. 77, n. 1, pp. 81–98, 1989.

[10] 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.

[11] ALVES, M. V., 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.

[12] ALVES, M. V., CARVALHO, L. K., BASILIO, J. C. "Supervisory control of networked discrete event systems with timing structure", *IEEE Transactions on Automatic Control*, v. 66, n. 5, pp. 2206–2218, 2020.

[13] LIMA, P. M., CARVALHO, L. K., MOREIRA, M. V. "Detectable and undetectable network attack security of cyber-physical systems", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 179–185, 2018.

[14] LIMA, P. M., ALVES, M. V. S., CARVALHO, L. K., et al. "Security against communication network attacks of cyber-physical systems", *Journal of Control, Automation and Electrical Systems*, v. 30, n. 1, pp. 125–135, 2019.

[15] LIMA, P. M., ALVES, M. V. S., CARVALHO, L. K., et al. "Confidentiality of cyber-physical systems using event-based cryptography". In: *21st IFAC World Congress 2020*, pp. 1761–1766, Berlin, Germany, 2020.

[16] LIMA, P. M., ALVES, M. V., CARVALHO, L. K., et al. "Security of Cyber-Physical Systems: Design of a Security Supervisor to Thwart Attacks", *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2021.

[17] ALVES, M. V., BARCELOS, R. J., CARVALHO, L. K., et al. "Robust decentralized diagnosability of networked discrete event systems against DoS and deception attacks", *Nonlinear Analysis: Hybrid Systems*, v. 44, pp. 101162, 2022.

[18] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[19] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "Coordinated decentralized protocols for failure diagnosis of discrete event systems", *Discrete Event Dynamic Systems: Theory and Applications*, v. 10, pp. 33–86, 2000.

[20] GENC, S., LAFORTUNE, S. “Predictability of event occurrences in partially-observed discrete-event systems”, *Automatica*, v. 45, n. 2, pp. 301–311, 2009.

[21] JIANG, S., KUMAR, R. “Failure diagnosis of discrete-event systems with linear-time temporal logic specifications”, *IEEE Transactions on Automatic Control*, v. 49, n. 6, pp. 934–945, 2004.

[22] BRYANS, J. W., KOUTNY, M., RYAN, P. Y. “Modelling Opacity Using Petri Nets”, *Electronic Notes in Theoretical Computer Science*, v. 121, n. Supplement C, pp. 101–115, 2005.

[23] LIN, F. “Opacity of discrete event systems and its applications”, *Automatica*, v. 47, pp. 496–503, 2011.

[24] WU, Y.-C., LAFORTUNE, S. “Comparative analysis of related notions of opacity in centralized and coordinated architectures”, *Discrete Event Dynamic Systems: Theory and Applications*, v. 23, n. 3, pp. 307–339, 2013.

[25] SHU, S., LIN, F., YING, H. “Detectability of discrete event systems”, *IEEE Transactions on Automatic Control*, v. 52, n. 12, pp. 2356–2359, 2007.

[26] SHU, S., LIN, F. “Co-detectability of multi-agent discrete event systems”. In: *2011 Chinese Control and Decision Conference (CCDC)*, pp. 1708–1713. IEEE, 2011.

[27] RAMADGE, P. J., WONHAM, W. M. “Supervisory control of a class of discrete event processes”, *SIAM journal on control and optimization*, v. 25, n. 1, pp. 206–230, 1987.

[28] LIN, F., WONHAM, W. M. “On observability of discrete-event systems”, *Information sciences*, v. 44, n. 3, pp. 173–198, 1988.

[29] BADOUEL, E., BEDNARCZYK, M., BORZYSZKOWSKI, A., et al. “Concurrent secrets”, *Discrete Event Dynamic Systems: Theory and Applications*, v. 17, n. 4, pp. 425–446, 2007.

[30] DUBREIL, J., DARONDEAU, P., MARCHAND, H. “Supervisory control for opacity”, *IEEE Transactions on Automatic Control*, v. 55, n. 5, pp. 1089–1100, 2010.

[31] SABOORI, A., HADJICOSTIS, C. N. “Notions of security and opacity in discrete event systems”. In: *46th IEEE Conference on Decision and Control*, pp. 5056–5061, 2007.

[32] SABOORI, A., HADJICOSTIS, C. N. "Verification of initial-state opacity in security applications of DES". In: *9th International Workshop on Discrete Event Systems (WODES)*, pp. 328–333, 2008.

[33] SABOORI, A., HADJICOSTIS, C. N. "Verification of infinite-step opacity and analysis of its complexity", *IFAC Proceedings Volumes*, v. 42, n. 5, pp. 46–51, 2009.

[34] SABOORI, A., HADJICOSTIS, C. N. "Coverage analysis of mobile agent trajectory via state-based opacity formulations", *Control Engineering Practice*, v. 19, pp. 967–977, 2011.

[35] SABOORI, A., HADJICOSTIS, C. N. "Verification of K-step opacity and analysis of its complexity", *IEEE Transactions on Automation Science and Engineering*, v. 8, n. 3, pp. 549–559, 2011.

[36] DUBREIL, J., JÉRON, T., MARCHAND, H. "Monitoring Confidentiality by Diagnosis Techniques", *Proceedings of the European Control Conference*, pp. 2584–2589, 2009.

[37] YIN, X., LAFORTUNE, S. "A new approach for the verification of infinite-step and k-step opacity using two-way observers", *Automatica*, v. 80, pp. 162–171, 2017.

[38] MA, Z., YIN, X., LI, Z. "Verification and enforcement of strong infinite-and k-step opacity using state recognizers", *Automatica*, v. 133, pp. 109838, 2021.

[39] FALCONE, Y., MARCHAND, H. "Enforcement and validation (at runtime) of various notions of opacity", *Discrete Event Dynamic Systems: Theory and Applications*, v. 25, n. 4, pp. 531–570, 2015.

[40] CASSEZ, F. "The Dark Side of Timed Opacity". In: *Advances in Information Security and Assurance: Third International Conference and Workshops, ISA 2009, Seoul, Korea, June 25-27, 2009. Proceedings*, pp. 21–30, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[41] WANG, L., ZHAN, N., AN, J. "The opacity of real-time automata", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 37, n. 11, pp. 2845–2856, 2018.

[42] SABOORI, A., HADJICOSTIS, C. N. "Current-state opacity formulations in probabilistic finite automata", *IEEE Transactions on automatic control*, v. 59, n. 1, pp. 120–133, 2014.

[43] BÉRARD, B., MULLINS, J., SASSOLAS, M. "Quantifying opacity", *Mathematical Structures in Computer Science*, v. 25, n. 2, pp. 361–403, 2015.

[44] KEROGLOU, C., HADJICOSTIS, C. N. "Probabilistic system opacity in discrete event systems", *Discrete Event Dynamic Systems*, v. 28, n. 2, pp. 289–314, 2018.

[45] YIN, X., LI, Z., WANG, W., et al. "Infinite-step opacity and K-step opacity of stochastic discrete-event systems", *Automatica*, v. 99, pp. 266–274, 2019.

[46] BRYANS, J. W., KOUTNY, M., MAZARÉ, L., et al. "Opacity generalised to transition systems", *International Journal of Information Security*, v. 7, n. 6, pp. 421–435, 2008.

[47] ZHANG, K., YIN, X., ZAMANI, M. "Opacity of Nondeterministic Transition Systems: A (Bi)Simulation Relation Approach", *IEEE Transactions on Automatic Control*, v. 64, n. 12, pp. 5116–5123, 2019.

[48] TONG, Y., LAN, H. "Current-State Opacity Verification in Modular Discrete Event Systems". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 7665–7670, 2019.

[49] YANG, S., HOU, J., YIN, X., et al. "Opacity of Networked Supervisory Control Systems Over Insecure Communication Channels", *IEEE Transactions on Control of Network Systems*, v. 8, n. 2, pp. 884–896, 2021.

[50] DENG, W., YANG, J., JIANG, C., et al. "Opacity of Fuzzy Discrete Event Systems". In: *2019 Chinese Control And Decision Conference (CCDC)*, pp. 1840–1845. IEEE, 2019.

[51] CONG, X., FANTI, M. P., MANGINI, A. M., et al. "On-line verification of current-state opacity by Petri nets and integer linear programming", *Automatica*, v. 94, pp. 205–213, 2018.

[52] CONG, X., FANTI, M. P., MANGINI, A. M., et al. "On-line verification of initial-state opacity by Petri nets and integer linear programming", *ISA Transactions*, v. 93, pp. 108 – 114, 2019.

[53] AN, L., YANG, G.-H. "Opacity Enforcement for Confidential Robust Control in Linear Cyber-Physical Systems", *IEEE Transactions on Automatic Control*, v. 65, n. 3, pp. 1234–1241, 2020.

[54] RAMASUBRAMANIAN, B., CLEAVELAND, R., MARCUS, S. I. "Notions of Centralized and Decentralized Opacity in Linear Systems", *IEEE Transactions on Automatic Control*, v. 65, n. 4, pp. 1442–1455, 2020.

[55] JACOB, R., LESAGE, J.-J., FAURE, J.-M. "Overview of discrete event systems opacity: Models, validation, and quantification", *Annual Reviews in Control*, v. 41, pp. 135–146, 2016.

[56] LAFORTUNE, S., LIN, F., HADJICOSTIS, C. N. "On the history of diagnosability and opacity in discrete event systems", *Annual Reviews in Control*, v. 45, pp. 257–266, 2018.

[57] SABOORI, A., HADJICOSTIS, C. N. "Opacity-enforcing supervisory strategies for secure discrete event systems". In: *47th IEEE Conference on Decision and Control (CDC)*, pp. 889–894, 2008.

[58] SABOORI, A., HADJICOSTIS, C. N. "Opacity-Enforcing Supervisory Strategies via State Estimator Constructions", *IEEE Transactions on Automatic Control*, v. 57, n. 5, pp. 1155–1165, 2012.

[59] YIN, X., LAFORTUNE, S. "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems", *IEEE Transactions on Automatic Control*, v. 61, n. 8, pp. 2140–2154, 2016.

[60] TONG, Y., LI, Z., SEATZU, C., et al. "Current-state opacity enforcement in discrete event systems under incomparable observations", *Discrete Event Dynamic Systems*, v. 28, n. 2, pp. 161–182, 2018.

[61] XIE, Y., YIN, X., LI, S. "Opacity enforcing supervisory control using nondeterministic supervisors", *IEEE Transactions on Automatic Control*, 2021. doi: 10.1109/TAC.2021.3131125.

[62] CASSEZ, F., DUBREIL, J., MARCHAND, H. "Synthesis of opaque systems with static and dynamic masks", *Formal Methods in System Design*, v. 40, n. 1, pp. 88–115, 2012.

[63] DULCE-GALINDO, J. A., ALVES, L. V., RAFFO, G. V., et al. "Enforcing State-Based Opacity using Synchronizing Automata". In: *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 7009–7014. IEEE, 2021.

[64] BARCELOS, R. J., BASILIO, J. C. "Enforcing current-state opacity through shuffle in event observations", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 100–105, 2018.

[65] WU, Y.-C., LAFORTUNE, S. "Synthesis of insertion functions for enforcement of opacity security properties", *Automatica*, v. 50, n. 5, pp. 1336–1348, 2014.

[66] WU, Y.-C., LAFORTUNE, S. "Synthesis of optimal insertion functions for opacity enforcement", *IEEE Transactions on Automatic Control*, v. 61, n. 3, pp. 571–584, 2016.

[67] JI, Y., WU, Y.-C., LAFORTUNE, S. "Enforcement of opacity by public and private insertion functions", *Automatica*, v. 93, pp. 369–378, 2018.

[68] KEROGLOU, C., LAFORTUNE, S. "Verification and Synthesis of Embedded Insertion Functions for Opacity Enforcement". In: *56th IEEE Conference on Decision and Control*, pp. 377–383, 2017.

[69] JI, Y., YIN, X., LAFORTUNE, S. "Enforcing opacity by insertion functions under multiple energy constraints", *Automatica*, v. 108, pp. 108476, 2019.

[70] KEROGLOU, C., RICKER, L., LAFORTUNE, S. "Insertion functions with memory for opacity enforcement", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 394–399, 2018.

[71] WU, Y.-C., RAMAN, V., RAWLINGS, B. C., et al. "Synthesis of Obfuscation Policies to Ensure Privacy and Utility", *Journal of Automated Reasoning*, v. 60, n. 1, pp. 107–131, 2018.

[72] JI, Y., LAFORTUNE, S. "Enforcing Opacity by Publicly Known Edit Functions". In: *56th IEEE Conference on Decision and Control*, pp. 377–383, 2017.

[73] JI, Y., YIN, X., LAFORTUNE, S. "Opacity enforcement using nondeterministic publicly known edit functions", *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4369–4376, 2019.

[74] WINTENBERG, A., BLISCHKE, M., LAFORTUNE, S., et al. "Enforcement of K-Step Opacity with Edit Functions". In: *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 331–338. IEEE, 2021.

[75] LI, X., HADJICOSTIS, C. N., LI, Z. "Extended Insertion Functions for Opacity Enforcement in Discrete Event Systems", *IEEE Transactions on Automatic Control*, 2021. doi: 10.1109/TAC.2021.3121249.

[76] ZHANG, B., SHU, S., LIN, F. "Maximum information release while ensuring opacity in discrete event systems", *IEEE Transactions on Automation Science and Engineering*, v. 12, n. 3, pp. 1067–1079, 2015.

[77] BEHINAEIN, B., LIN, F., RUDIE, K. "Optimal Information Release for Mixed Opacity in Discrete-Event Systems", *IEEE Transactions on Automation Science and Engineering*, v. 16, n. 4, pp. 1960–1970, 2019.

[78] YIN, X., LI, S. “Synthesis of dynamic masks for infinite-step opacity”, *IEEE Transactions on Automatic Control*, v. 65, n. 4, pp. 1429–1441, 2020.

[79] HOU, J., YIN, X., LI, S. “A Framework for Current-State Opacity under Dynamic Information Release Mechanism”, *arXiv preprint arXiv:2012.04874*, 2020.

[80] BARCELOS, R. J., BASILIO, J. C. “Enforcing current-state opacity through shuffle and deletions of event observations”, *Automatica*, v. 133, pp. 109836, 2021.

[81] TAKAI, S. “Robust prognosability for a set of partially observed discrete event systems”, *Automatica*, v. 51, pp. 123–130, 2015.

[82] BARCELOS, R. J., CORRÊA, M. A., BASILIO, J. C. “Predictability of Discrete-Event Systems with Cycles of States Connected with Unobservable Events”, *Journal of Control, Automation and Electrical Systems*, v. 31, n. 4, pp. 842–849, 2020.

[83] XIAO, C., LIU, F. “Robust Fault Prognosis of Discrete-Event Systems Against Loss of Observations”, *IEEE Transactions on Automation Science and Engineering*, 2021. doi: 10.1109/TASE.2021.3049400.

[84] WATANABE, A. T. Y., LEAL, A. B., CURY, J. E., et al. “Combining online diagnosis and prognosis for safe controllability”, *IEEE Transactions on Automatic Control*, 2021. doi: 10.1109/TAC.2021.3124185.

[85] KUMAR, R., TAKAI, S. “Decentralized prognosis of failures in discrete event systems”, *IEEE Transactions on Automatic Control*, v. 55, n. 1, pp. 48–59, 2010.

[86] 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.

[87] KHOUMSI, A., CHAKIB, H. “Conjunctive and disjunctive architectures for decentralized prognosis of failures in discrete-event systems”, *IEEE Transactions on Automation Science and Engineering*, v. 9, n. 2, pp. 412–417, 2012.

[88] YIN, X., LI, Z. “Decentralized fault prognosis of discrete event systems with guaranteed performance bound”, *Automatica*, v. 69, pp. 375–379, 2016.

[89] 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.

[90] YIN, X., LI, Z. "Decentralized fault prognosis of discrete-event systems using state-estimate-based protocols", *IEEE transactions on cybernetics*, v. 49, n. 4, pp. 1302–1313, 2019.

[91] ZHOU, Y., CHEN, Z., LIU, Z., et al. "Three kinds of coprognosability for partially-observed discrete event systems via a matrix approach", *Nonlinear Analysis: Hybrid Systems*, v. 42, pp. 101073, 2021.

[92] CHANG, M., DONG, W., JI, Y., et al. "On fault predictability in stochastic discrete event systems", *Asian journal of Control*, v. 15, n. 5, pp. 1458–1467, 2013.

[93] CHEN, J., KUMAR, R. "Stochastic failure prognosability of discrete event systems", *IEEE Transactions on Automatic Control*, v. 60, n. 6, pp. 1570–1581, 2015.

[94] LIAO, H., LIU, F., ZHAO, R. "Reliable Co-Prognosability of Decentralized Stochastic Discrete-Event Systems and a Polynomial-Time Verification", *IEEE Transactions on Cybernetics*, 2021. doi: 10.1109/TCYB.2021.3051260.

[95] LEFEBVRE, D. "State estimation and fault prediction with partially observed Petri nets". In: *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–8. IEEE, 2013.

[96] LEFEBVRE, D. "Fault diagnosis and prognosis with partially observed Petri nets", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 44, n. 10, pp. 1413–1424, 2014.

[97] YIN, X. "Verification of prognosability for labeled Petri nets", *IEEE Transactions on Automatic Control*, v. 63, n. 6, pp. 1828–1834, 2018.

[98] YOU, D., WANG, S., SEATZU, C. "Verification of fault-predictability in labeled Petri nets using predictor graphs", *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4353–4360, 2019.

[99] CASSEZ, F., GRASTIEN, A. "Predictability of event occurrences in timed systems". In: *International conference on formal modeling and analysis of timed systems*, pp. 62–76. Springer, 2013.

[100] WATANABE, A. T., SEBEM, R., LEAL, A. B., et al. "Fault prognosis of discrete event systems: An overview", *Annual Reviews in Control*, v. 51, pp. 100–110, 2021.

[101] BARCELOS, R. J., BASILIO, J. C. "New predictability verification tests for discrete-event systems modeled by finite state automata", *IFAC-PapersOnLine*, v. 53, n. 4, pp. 243–249, 2020.

[102] VIANA, G. S., BASILIO, J. C. "Codiagnosability of discrete event systems revisited: A new necessary and sufficient condition and its applications", *Automatica*, v. 101, pp. 354 – 364, 2019.

[103] VIANA, G., MOREIRA, M. V., BASILIO, J. C. "Codiagnosability Analysis of Discrete-Event Systems Modeled by Weighted Automata", *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4361–4368, 2019.

[104] 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, pp. 1679–1684, 2011.

[105] MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. "'Polynomial time verification of decentralized diagnosability of discrete event systems' versus 'Decentralized failure diagnosis of discrete event system': A critical appraisal", *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2015.

[106] CLAVIJO, L. B., BASILIO, J. C. "Empirical studies in the size of diagnosers and verifiers for diagnosability analysis", *Discrete Event Dynamic Systems*, v. 27, n. 4, pp. 701–739, 2017.

[107] SANTORO, L. P., MOREIRA, M. V., BASILIO, J. C. "Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers", *Automatica*, v. 77, pp. 93–102, 2017.

[108] BASILIO, J. C., LIMA, S. T. S., LAFORTUNE, S., et al. "Computation of minimal event bases that ensure diagnosability", *Discrete Event Dynamic Systems: Theory and Applications*, v. 22, n. 3, pp. 249–292, 2012.

[109] TARJAN, R. "Depth first search and linear graph algorithms", *SIAM Journal of Computer*, v. 1, n. 2, pp. 146–160, 1972.

[110] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to Algorithms*. 3rd ed. Cambridge, MA, MIT Press, 2009.

[111] JIANG, S., HUANG, Z., CHANDRA, V., et al. "A polynomial algorithm for testing diagnosability of discrete-event systems", *IEEE Trans. on Automatic Control*, v. 46, n. 8, pp. 1318–1321, 2001.

[112] YOO, T.-S., LAFORTUNE, S. "Polynomial-time verification of diagnosability of partially observed discrete-event systems", *Automatic Control, IEEE Transactions on*, v. 47, n. 9, pp. 1491–1495, 2002.

[113] QIU, W., KUMAR, R. "Decentralized failure diagnosis of discrete event systems", *IEEE Trans. on Systems, Man and Cybernetics, Part A*, v. 36, n. 2, pp. 384–395, 2006.

[114] VIANA, G. S., MOREIRA, M. V., BASILIO, J. C. "Codiagnosability Analysis of Discrete-Event Systems Modeled by Weighted Automata", *IEEE Transactions on Automatic Control*, v. 64, n. 10, pp. 4361–4368, 2019.

[115] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. "Comparative analysis of related notions of robust diagnosability of Discrete-Event Systems", *Annual Reviews in Control*, v. 51, pp. 23–36, 2021.

[116] SU, R., VAN SCHUPPEN, J. H., ROODA, J. E. "The Synthesis of Time Optimal Supervisors by Using Heaps-of-Pieces", *IEEE Transactions on Automatic Control*, v. 57, n. 1, pp. 105–118, 2012.

[117] DIMA, C. "Real-time automata", *Journal of Automata, Languages and Combinatorics*, v. 6, n. 1, pp. 3–24, 2001.