Università degli Studi di Cagliari

PHD DEGREE

in Electronic and Computer Engineering

Cycle XXXIII

TITLE OF THE PHD THESIS

SUPERVISORY CONTROL AND ANALYSIS OF PARTIALLY-OBSERVED DISCRETE EVENT SYSTEMS

Scientific Disciplinary Sector(s)

ING-INF/04 AUTOMATICA

PhD Student:	**Dan You**
Supervisor:	**Carla Seatzu**
	Shouguang Wang

Final exam. Academic Year 2019 – 2020

Thesis defence: February 2021 Session

Supervisory Control and Analysis of Partially-observed Discrete Event Systems

by Dan You

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electronic and Computer Engineering
in the Department of Electrical and Electronic Engineering of
University of Cagliari, Italy
2020

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my tutor, Professor Carla Seatzu, for her encouragement, guidance and support in my entire doctoral research career and for providing me opportunities to attend various academic conferences and study aboard (in U.S.A. and China). I really appreciate her detailed comments on my work, which helped me a lot to improve not only the work but also my research ability. I would also like to thank my co-tutor, Professor Shouguang Wang in China, for his long-term discussions and constructive suggestions on my research topics.

I would like to thank Prof. Alessandro Giua for his critical comments on all my presentations at the University of Cagliari and for his lecture that broadened my horizon from Discrete Event Systems to Hybrid Systems. I also thank other teachers whose lectures deepened my understanding in some interesting topics of computer science and thank my colleagues who gave me comments on my work and helped me in my daily life.

I would like to thank Prof. Mengchu Zhou, who supervised me when I visited New Jersey Institute of Technology in U.S.A. I am also very grateful to the colleagues in his group for their discussions and precious suggestions on my work and especially for their companionship and help during that time.

I would like to acknowledge the financial support from University of Cagliari.

Finally, I thank my parents for their understanding and all my friends who helped me during my PhD period.

ABSTRACT

Nowadays, a variety of real-world systems fall into discrete event systems (DES). In practical scenarios, due to facts such as limited sensor technique, sensor failure, unstable network and even the intrusion of malicious agents, it might occur that some events are unobservable, multiple events are indistinguishable in observations, and observations of some events are nondeterministic. Taking into account various practical scenarios, increasing attention in the DES community has been paid to *partially-observed DES*, which in this thesis refer broadly to those DES with partial and/or unreliable observations.

In this thesis, we focus on two topics of partially-observed DES, namely, *supervisory control* and *analysis*.

The first topic includes two research directions in terms of system models. One is the supervisory control of DES with both unobservable and uncontrollable events, focusing on the *forbidden state problem* in this thesis; the other is the supervisory control of DES vulnerable to *sensor-reading disguising attacks* (SD-attacks), which may also be interpreted as DES with nondeterministic observations, addressing not only the *forbidden state problem* but also the *liveness-enforcing problem*. Specifically, Petri nets (PN) are used as a reference formalism in this topic. First, we study the forbidden state problem in the framework of PN with both unobservable and uncontrollable transitions, assuming that unobservable transitions are uncontrollable. Focusing on ordinary PN subject to an admissible *Generalized Mutual Exclusion Constraint* (GMEC), an optimal on-line control policy with polynomial complexity is proposed provided that a particular subnet, called *observation subnet*, satisfies certain conditions in structure. It is then discussed how to obtain an optimal on-line control policy for PN subject to an arbitrary GMEC. Next, we still consider the forbidden state problem but in PN vulnerable to SD-attacks. Assuming the control specification in terms of a GMEC, we propose three methods to derive on-line control policies. The first two lead to an optimal policy but are computationally inefficient when applied to large-size systems, while the third method computes a policy with timely response even for large-size systems but at the expense of optimality. Finally, we investigate the liveness-enforcing problem still assuming that the system is vulnerable to SD-attacks. In this problem, the plant is modelled as a bounded PN, which allows us to off-line compute a supervisor starting from constructing the reachability graph of the considered PN. Then, based on repeatedly computing a more restrictive liveness-enforcing supervisor under no attack and constructing a so-called *basic supervisor*, an off-line method that synthesizes a liveness-enforcing supervisor tolerant to an SD-attack is proposed.

In the second topic, we care about the verification of properties related to system security. Two properties are considered, i.e., *fault-predictability* and *event-based opacity*. The former is a property proposed in the literature, characterizing the situation that the occurrence of any fault in a system is predictable, while the latter is a newly proposed property in the thesis, which is a confidentiality property basically characterizing the fact that secret events of a system should not be revealed to an external observer within their critical horizons. In the case of fault-predictability, DES are modeled by *labeled* PN. A necessary and sufficient condition for fault-predictability is derived by characterizing the structure of the *Predictor Graph*. Furthermore, two rules are proposed to reduce the size of a PN, which allow one to analyze the fault-predictability of the original net by verifying the fault-predictability of the reduced net. When studying event-based opacity, we use *deterministic finite-state automata* as the reference formalism. Considering different scenarios from the simplest one to the most complicated one, we propose four notions, namely, *K-observation event-opacity*, *infinite-observation event-opacity*, *event-opacity* and *combinational event-opacity*. Moreover, appropriate verifiers are proposed to analyze the proposed four properties.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DES	Discrete Event Systems
PN	Petri Nets
GMEC	Generalized Mutual Exclusion Constraints
SD-attacks	Sensor-reading Disguising attacks
HVAC systems	Heating, Ventilation, and Air Conditioning systems
DFA	Deterministic Finite-state Automata
SCC	Strongly Connected Component
BBF	Backward-conflict and Backward-concurrent Free
RG	Reachability Graph
PG	Predictor Graph

CHAPTER I

Introduction

Discrete event systems (DES) are a class of dynamical systems whose states are discrete and whose state evolution is driven by the occurrence of asynchronous events. There are a variety of examples in the real world, ranging from flexible manufacturing systems and intelligent transportation systems to communication systems and distributed software systems. They are either inherently state-discrete and event-driven or modeled in a discrete-abstraction level for the study of problems concerning with their “high-level” logical behavior. In the framework of DES, a lot of problems may be addressed in an elegant manner, related to issues like the safety, information security, and performance of systems in the real world.

For external observers, the behavior of a DES is typically acquired by observing the occurrence of events. Initially, the study on DES is commonly performed under the assumption that we have the complete and reliable observation on its behavior, i.e., the occurrence of events in the system. This is actually an ideal situation and not common to see in practical scenarios, which leads to solutions under such an assumption with limited applications. Indeed, in practical scenarios, the observation on events is realized by getting readings or signals from equipped sensors in the system. Due to facts like limited sensor technique, high cost of sensor installation, and failure of communication and sensors, it could happen that not all events are observable and it could also happen that multiple observable events share the same output reading or signal, i.e., they are indistinguishable in observations. Moreover, some observations of events may be nondeterministic due to sensor failure, unstable network and even the intrusion of malicious agents. Consequently, taking into account various practical scenarios, more and more attention in the DES community has been turned to *partially-observed DES*, which in this thesis refer broadly to those DES with partial and/or unreliable observations.

Mathematically, to model the observation ability on a partially-observed DES, a projection is defined on the event set. It can be a projection that simply erases the outputs of some events that coincide with unobservable events. It can be a more general projection that associates to each event a label so that indistinguishable events may be taken into account. More general, it can be a projection that associates to each event a set of labels, which allows one to capture events with nondeterministic observations. Different projections are used depending on specific problems to be studied.

In this thesis, we focus on two topics of partially-observed DES, namely, *supervisory control* and *analysis*. The first topic further includes two research directions in terms of system models. One is the supervisory control of DES with both unobservable and uncontrollable events, focusing on the *forbidden state problem* in this thesis; the other is the supervisory control of DES vulnerable to network attacks, which may also be interpreted as DES with nondeterministic observations, taking into account not only the *forbidden state problem* but also the *liveness-enforcing problem*. In the second topic, we care about the verification of properties related to system security. Two properties are considered in this thesis, i.e., *fault-predictability* and *event-based opacity*. The former is a property proposed in the literature, characterizing the situation that the occurrence of any fault in a system is predictable, while the latter is a newly proposed property in the thesis, which is a confidentiality property considering that secret events of a system should not be revealed to an external observer within their critical horizons. The outline of the thesis is shown in Fig. 1.1. In what follows, the motivation and related works of the studied problems are introduced, respectively.

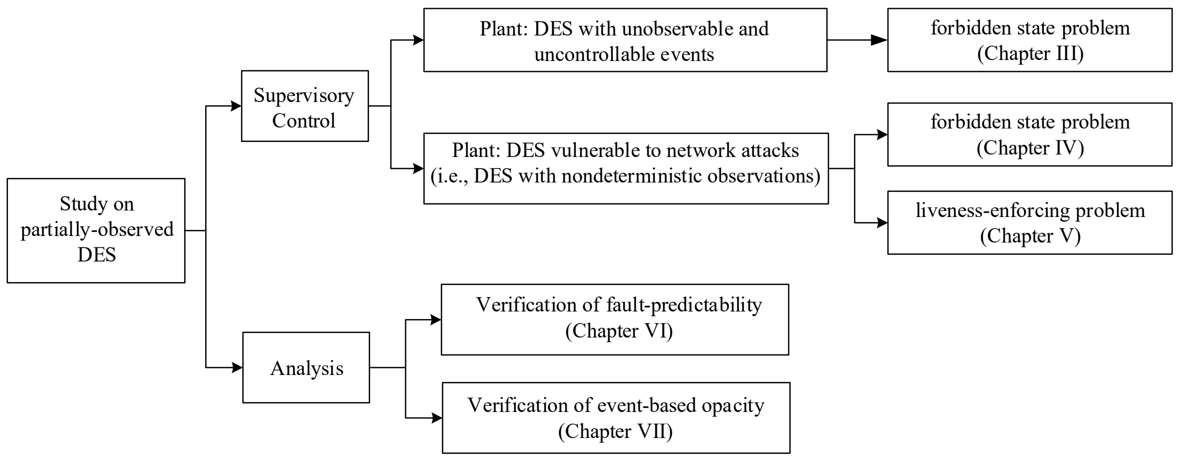

Study on partially-observed DES
Supervisory Control
Analysis
Plant: DES with unobservable and uncontrollable events
Plant: DES vulnerable to network attacks (i.e., DES with nondeterministic observations)
forbidden state problem (Chapter III)
forbidden state problem (Chapter IV)
liveness-enforcing problem (Chapter V)
Verification of fault-predictability (Chapter VI)
Verification of event-based opacity (Chapter VII)

Fig. 1.1 Outline of the thesis

1.1 Motivation and Literature Review

1.1.1 Forbidden State Problem of DES with Unobservable and Uncontrollable Events

The forbidden state problem of DES is a classic problem. It consists in the supervisory control of a DES so that no forbidden states are reached during its evolution (or equivalently, only allowed states are reached during its evolution). Forbidden states refer to those states that a system is not expected to reach, which vary with different control goals and different practical scenarios. They are often the states that possibly cause unnecessary costs and/or lead to catastrophic results; such as the deadlocks in manufacturing systems, the buffer overflow in network communication systems, the collision between trains on the track in transportation systems, and even the leakage of radioactive materials in nuclear power plants. Forbidden states also can be the states that violate some property to be enforced. For example, opacity, an important information flow property related to the information security of a system, whose enforcement can be realized via supervisory control on the considered system. Indeed, many problems in the DES area can be transformed into the forbidden state problem as long as a set of forbidden or allowed states is obtained. Consequently, it is significantly important to investigate the forbidden state problem.

The problem has been well addressed in the case that events of a DES are all controllable and observable. By taking into account real scenarios where some events may be unobservable and/or uncontrollable, many researchers reconsider the forbidden state problem in the presence of unobservable and/or uncontrollable events. In particular, the problem has been mostly investigated under the assumption that certain events are uncontrollable, but all of them are observable. In this thesis, we use Petri nets (PNs) as a modelling tool to deal with the forbidden state problem in the most general setting where transitions (i.e., events) may be unobservable and/or uncontrollable. As typically done in this framework and as it actually occurs in several real applications [68], an assumption is made in this work, namely, unobservable transitions are also uncontrollable. Recently, for such PN under the above assumption, Luo *et al*. [63] propose two approaches to design control policies. The first approach is applicable to arbitrary PN with the state specification being an arbitrary GMEC. Such an approach is efficient but the optimality is not guaranteed. In more detail, this policy requires the transformation of the given GMEC into a disjunction of multiple admissible GMEC. However, the two admissible marking sets before and after transformation, are not coincident and the policy is thereby not optimal. The second approach guarantees an optimal policy with low computational complexity but it only applies under very restrictive assumptions. It is indeed only applicable to the state specification being a GMEC with binary coefficients. Besides, it requires that the uncontrollable subnet is a *state machine* that satisfies some restrictive assumptions. Inspired by [63], this thesis aims to propose an efficient and optimal control policy that can be applied to more general cases.

- **Literature Review**

The forbidden state problem of DES has a long history [49]. Many researchers use PNs as the reference formalism to deal with the forbidden state problem. Usually, a state specification is given to define the state space where a system is allowed to evolve, which is called the *legal marking set* in the framework of PNs. This problem has been addressed since nineties in the case of PNs where all transitions are controllable and observable [38, 109]. For a PN with uncontrollable and/or unobservable transitions, the problem becomes more complicated.

The problem has been mostly investigated for PNs with all transitions observable but some transitions uncontrollable. To make such a PN meet the state specification, it is not sufficient to restrict its behavior within the legal marking set. It could happen that a forbidden marking is reached from some legal markings by firing uncontrollable transitions. Instead, its behavior should be restricted within a subset of the legal marking set, called the *admissible marking set*, as proposed by Krogh and Holloway [49]. Thereafter, the key to solving the forbidden state problem of such PNs turns to the computation of the admissible marking set. A lot of efforts have been made on developing constraint transformation techniques that can efficiently convert *Generalized Mutual Exclusion Constraints* (GMECs) describing the legal marking set into GMECs characterizing the admissible marking set [7, 34, 40, 60-62, 68, 100-103]. In addition, some works study the existence of supervisory policies in the presence of uncontrollable transitions; see, e.g., [26, 85].

When unobservable transitions are also taken into consideration, the forbidden state problem becomes more difficult to handle. Moody *et al.* [68] develop a supervisory synthesis technique for PNs with uncontrollable and unobservable transitions that provides a model of the closed-loop system but cannot guarantee the optimality (namely, maximal permissiveness) of the synthesized supervisor. Achour *et al.* [1] propose an optimal control policy for a subclass of PNs, called *marked graphs*, with unobservable transitions. However, it is only applicable to the case that the marked graph is live and the given GMECs meet some conditions. Ru *et al.* [75] also design an optimal supervisor that is applicable to a much more general case where the state specification is not necessarily a GMEC but can be an arbitrary bounded set. However, the approach requires on-line calculations that are of exponential complexity with respect to the number of places of the net, so it may be not feasible for complex real problems. Cabasino *et al.* [12] deal with a more general problem where the initial marking of the PN is unknown but belongs to a given convex set and indistinguishable transitions are also considered. They propose two solutions to derive control policies. The first one requires the computation of the *marking observer* and the solution of an integer linear programming problem for each controllable transition. Unfortunately, the optimality is not guaranteed by this approach. On the contrary, the second approach guarantees the optimality of the solution and moves the most burdensome parts of the computations off-line. However, it may be too computationally demanding in real complex problems.

A very rich literature also exists on supervisory control of DES with partial observations based on automata models. The control specifications in this framework are typically language specifications instead of state specifications considered in this work. Specifically, a language specification is defined by a sublanguage K of the system language. It is known that a sublanguage $L \subseteq K$ being controllable and observable is a necessary and sufficient condition for the existence of a supervisor that exactly enforces L [22, 59]. Under the partial observation setting, there may not exist a supremal controllable and observable sublanguage of K. Some efforts have thereby been made on solving this problem. Initially, the concept of *normality* is proposed in [22, 59] and it is proved that the supremal controllable and normal sublanguage exists, which however is usually too restrictive. Thereafter, two different approaches developed in [13] and [90], respectively, provide solutions that are larger than the supremal controllable normal sublanguage but neither of them is the maximal solution. Hadj-Alouane *et al.* [37] propose on-line control algorithms that could derive a maximal solution for prefix-closed language specifications. Yin and Lafortune consider in [114] the specification that is a non-prefix-closed sublanguage. Based on constructing a bipartite transition system named *Non-blocking All Inclusive Controller*, they develop an algorithm that synthesizes a safe, non-blocking and maximally permissive supervisor, if it exists. Moreover, they consider in [115] the property enforcement on DESs by supervisory control, where a supervisor that is property-enforcing and maximally permissive can be computed by constructing a transition system called *All Enforcement Structure*. In addition, they consider in [116] a more generalized supervisory control problem called the *range control problem*, where both a standard upper bound specification that captures the legal behavior and a lower bound specification that captures the minimum required behavior are taken into consideration. An algorithm is provided in [116] that solves this problem by effectively synthesizing a maximally-permissive safe supervisor.

Compared with automata, PN models have the advantage that structural characteristics may be utilized to solve the supervisory control problem. Besides, PN models allow the study on infinite state systems. Due to such advantages, we adopt PN as a modelling tool to deal with the forbidden state problem in this work.

1.1.2 Supervisory Control of DES Vulnerable to Network Attacks

Nowadays, communication networks are increasingly used in practical systems such as various cyber-physical systems whose components are often connected via communication networks for information exchange. The use of networks, however, might compromise the security of a system, making it vulnerable to various network attacks. The attack issues thereby receive more and more attention in both academia and industry.

In DES community, attack issues are typically considered in the closed-loop control system abstracted in Fig. 1.2, where the plant is modelled as a DES and a supervisor enforces a control specification on the plant by enabling/disabling some events according to the current observation on events generated by the plant. In a realistic scenario, the supervisor observes the occurrence of events by getting readings from sensors and enables/disables events by issuing commands to control actuators. In the case that the supervisor communicates with sensors and actuators via networks, the potential attacks may be divided into two categories in terms of locations, i.e., attacks in sensor and actuator channels. The former indicates the tampering with sensor-readings in vulnerable sensor channels, while the latter indicates the tampering with the control commands on actuators in vulnerable actuator channels.

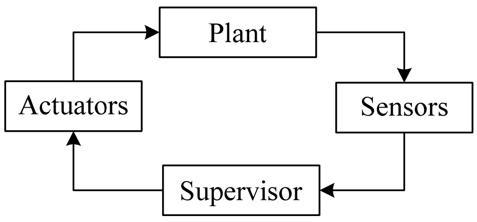


Fig. 1.2 Closed-loop control system

In this thesis, we consider attacks in sensor channels that we call *sensor-reading disguising attacks* (*SD-attacks* for short), which are a subclass of the *replacement-removal attacks* in [97]. In more detail, the intruder may disguise the occurrence of an event in the plant as the occurrence of another event by tampering with the sensor-readings transmitted in vulnerable sensor channels. We study the synthesis of a supervisor *tolerant* to such attacks. It means that the supervisor is able to enforce a control specification to the plant even in the presence of such attacks. We note that in this work we deal with a plant whose events are all controllable and observable but the potential existence of attacks makes the considered system with nondeterministic observations. Thus, the studied problem can also be interpreted as the supervisory control of DES with nondeterministic observations.

Two control specifications are studied in this thesis, respectively. One is the state specification characterized by GMEC and the other is liveness.

When the control specification takes the form of GMEC, we use PN as a reference formalism to study the control problem, which allows us to handle both bounded and unbounded DES. We note that [97] and [112] are closely related to this work. Both of them consider the supervisor synthesis problem. Specifically, their focus is that of synthesizing a supervisor off-line represented by an automaton. Once such a supervisor is synthesized, little on-line computation is needed. However, computing such a supervisor requires an exhaustive reachability analysis, which makes such approaches hardly applicable to DESs with a large number of states and unfeasible in the case of DES with an unbounded state space. In this work, we develop an on-line control policy for a plant modelled by a PN system that can be bounded or unbounded. To ensure the timely response of a control policy, we make efforts on improving its on-line computational efficiency.

Consider the control specification that is liveness. Liveness is an important property for many practical systems, which characterizes a specific dynamical behavior of a system. The problem of liveness enforcing on DES in the framework of PNs has been extensively studied, resulting in a variety of approaches under different problem settings [5, 19, 21, 36, 39, 53, 64, 65, 99, 101, 105, 110, 126, 129]. However, how to enforce liveness in the presence of network attacks has not been yet investigated. As well known, even under no attacks, reachability analysis is inevitable in solving a liveness enforcing problem if no restrictions are made on the considered PN class, which leads to solutions with a complexity that is exponential in the size of the net. Now, in the presence of attacks, observations produced by a system might be inconsistent with the real evolution of the system. Thus, it could happen that a control action associated with an observation forbids/permits some sequences that it does not intend to forbid/permit, whereas the property of liveness is easy to be damaged if sequences are forbidden/permitted in a casual way. Consequently, the difficulty of enforcing liveness is further increased under attacks. We aim to get a solution for such a problem. Specifically, we study the off-line synthesis of a liveness-enforcing supervisor tolerant to SD-attacks considering the plant modelled as a bounded PN system.

- **Literature Review**

Thorsley and Teneketzis [92] study intrusion detection in the supervisory control of DES. The focus of their work is on finding a control specification that can be realized by a supervisor when there is no attack and the supervisor is capable of preventing damages caused by attacks. The attacks considered in their work are attacks in actuator channels. Carvalho *et al.* [14] consider a more explicit type of attacks in actuator channels named *actuator enablement attacks* (AE-attacks), which overwrites the control command on some actuators from “disable” to “enable”. They model the behavior of the controlled system under AE-attacks and then develop a method to determine if the system is AE-safe controllable. For AE-safe controllable systems, the defense strategy is straightforward, consisting in detecting AE-attacks online and disabling all controllable events once an attack is detected. Later, they consider *actuator disablement attacks* and also extend the detection and defense strategy to two types of attacks in sensor channels named *sensor erasure attacks* and *sensor insertion attacks*. Lima *et al.* [55, 56] carry out similar works on proposing defense strategies but deal with a more general attack called the *man-in-the-middle attack* [23], where the intruder/attacker can observe, hide, create or even replace information transiting in a communication channel. In [57], Lima *et al.* improve the defense strategies. In particular, a security module is designed that carefully disables controllable events whose occurrence leads to unsafe states rather than “abruptly” disabling all controllable events once an attack is detected. In essence, the works [14, 55-57, 92] share some similarity with the works on fault detection and diagnosis [24, 127, 132].

Wakaiki *et al.* [97] also investigate the supervisory control of DESs vulnerable to attacks in sensor channels. To be precise, they study the robust control with respect to a set of attackers with different attack abilities. They assume that only one attacker in the set may appear in the system but they do not know which one it is. Then, given a language specification, they want to find a supervisor that enforces the specification to the controlled system no matter whether or not there exist attacks and no matter which attacker they are facing. To this end, they define a new notion of observability of a language specification in the presence of attacks and characterize the existence of a supervisor by the usual notion of controllability and the new notion of observability of the language specification. In particular, for *replacement-removal attacks*, they construct a robust supervisor in the automaton framework and propose a sufficient condition under which a maximally permissive supervisor exists. We notice that the DES under attacks in [97] is actually a DES with nondeterministic observations. In other words, this work has some similarities with the works on supervisory control of DES with nondeterministic observations [96, 108, 112]. Specifically, [96, 108] mainly deal with the supervisor existence problem given a language specification, while [112] solves the supervisor synthesis problem by proposing a model transformation method. Compared with [96, 108, 112], the work [97] handles a more general case on the nondeterministic observation function that is uncertain.

Recently, some works [31, 35, 86, 131] handle intelligent attacks in closed-loop control systems. An intelligent attacker typically has the knowledge of the supervisor and is covert to the system user while inflicting damages. The existing works usually first synthesize an intelligent attack strategy from the viewpoint of the external malicious agent and then consider the design of a control policy. Note that the attacks considered in this thesis are not intelligent.

We conclude that almost all the aforementioned studies address attack issues using automata as a reference formalism, which usually limits their applicability to bounded DES. Moreover, they consider either a state specification or a language specification, which motivates us to consider the liveness specification.

1.1.3 Fault-predictability

A faulty behavior is a deviation of a system from its regular evolution. In a DES framework, faults typically correspond to undesirable events, which may either be observable or unobservable. As an example, in Heating, Ventilation, and Air Conditioning (HVAC) systems, faults are valves that get stuck open or closed, pumps that fail on or off, and controller modules that fail on or off [83, 84].

Rich literature exists to study the problem of fault diagnosis in the case that faults are unobservable, which consists in providing a systematic approach to detect the occurrence of a fault based on the partial observation of the system evolution. Besides, it is fundamental to preliminarily investigate if the occurrence of a fault can be detected within a finite delay, or more precisely, after the occurrence of a finite number of events. Such a problem is known as diagnosibility analysis [6, 10, 32, 45, 46, 51, 73, 83, 122], or codiagnosability analysis when performed in a decentralized setting [72]. Some works also deal with the application of various diagnosis approaches in real systems, such as HVAC systems [84]; manufacturing systems [71]; document processing systems [82]; telecommunication networks [74]; and cloud systems [130].

In this thesis, we study a problem strictly related to diagnosability, namely fault-predictability (or fault-prognosability). In such a case, faults are not necessarily unobservable events. They could be simply undesirable events, whose occurrence should be predicted in advance. Compared with fault diagnosis, fault prediction allows the system operator to be more proactive than passive when facing potential faults. Indeed, once the occurrence of a fault is predicted, timely reactions can be provided for the impending fault, such as limiting the possible future evolution of the system or even halting it. That way, disastrous consequences caused by faults may be avoided, which is of great importance especially for safety-critical systems.

The notion of fault-predictability has been first formalized in [33] and [42] for formal languages. Specifically, a system is fault-predictable if, for any fault that the system may suffer from, its occurrence could be predicted, i.e., we can know for sure from an observation that the fault will definitely occur after the occurrence of a finite number of events. In [33], Genc and Lafortune provide two approaches for testing the predictability of systems modeled by regular languages and present applications of the proposed approaches in HVAC systems and computer intrusion detection problems.

Thereafter, many methods emerge to address the problem of fault prediction under different frameworks, such as probabilistic systems [70], stochastic systems [20], hybrid systems [18], and timed systems [17]. Moreover, some works deal with this problem in a decentralized [48, 50, 119, 120] or distributed framework [89, 111]. Besides, Takai introduces in [87] the concept of robust prognosability and investigates the robust fault prognosis problem.

Most of the above works are based on automata. More recently, taking advantage from PN features, some researchers adopt PNs as the reference formalism. Madalinski *et al.* [66] reduce the predictability problem to that of linear time temporal logic-X model checking. In particular, PN unfolding is used in their work. Lefebvre [51] studies the problem of fault prognosis as well as fault diagnosis using partially observed stochastic PNs. Ammour *et al.* [2, 3] also investigate fault prognosis in the context of partially observed stochastic PNs. Specifically, they propose an incremental approach in [3] to compute the probability of the occurrence of a fault in a future time interval and in [2] develop an approach that can bound the estimation error when the probability of a future fault occurrence is estimated.

Recently, Yin [113] studies the verification of fault-predictability in unbounded labeled PNs. He transforms the verification problem to a PN model checking problem and derives a necessary and sufficient condition for the fault-predictability, which requires to determine whether or not there exists a particular transition sequence in a particular PN associated with the original one. However, how to implement this condition (i.e., how to search particular transition sequences) to verify the fault-predictability is not further investigated.

Motivated by the work of Yin [113], in this thesis, we propose a new approach to verify the fault-predictability of bounded and unbounded labeled PNs still based on PN model checking. As in [113], we construct a new PN that we call *Predictor Net*, which is a more compact PN than the auxiliary PN used in [113]. Most importantly, we construct a special graph called *Predictor Graph*, which is similar to the reachability graph/coverability graph of the Predictor Net but with some appropriate modifications, and then propose a necessary and sufficient condition for the fault-predictability by looking at the structure of the Predictor Graph. Finally, we propose two rules to reduce the size of the given labeled PN and show that the fault-predictability of the original net can be studied by verifying the fault-predictability of the reduced net.

1.1.4 Event-based Opacity

An ever growing amount of information is being exchanged in nowadays society. Opacity is one of the most important information flow properties related to privacy and security arising in cyber/cyber-physical systems. It characterizes the situation in which a "secret" of a system can never be revealed to a potentially malicious intruder who has full knowledge of the system model but partial observability on its behavior. More precisely, a system is *opaque* if for any of its "secret" behavior, there exists a "non-secret" behavior that provides the same observation to the intruder. The notion of opacity has been first proposed in [67] for the analysis of cryptographic protocols in computer science. Later on, Bryans *et al.* extend the notion to the area of DES both in the framework of PN [9] and transition systems [8]. Saboori and Hadjicostis [76] as well as Badouel *et al.* [4] study opacity problems in the framework of finite-state automata. Since then, increasing attention has been devoted to opacity problems in the DES community, leading to a lot of works including the verification and enforcement of opacity [16, 25, 29, 30, 43, 44, 47, 58, 77-81, 88, 93-95, 106, 107, 117, 118, 121, 128]. The reader is referred to the survey in [41] for more references.

In the context of DES, several definitions of opacity have been formalized with the purpose of characterizing different security requirements in different practical scenarios. Opacity notions in the literature may be classified into two main groups, i.e., language-based opacity and state-based opacity. The former defines the secret as a sublanguage of the considered system, while the latter defines the secret as a subset of the system state space. In more detail, the language-based opacity is formulated in [4, 28, 58] with slightly different definitions. In simple words, the secret consists in a set of strings whose occurrence should not be discovered by the intruder. This happens provided that all such strings produce the same observation of non-secret strings. In contrast, the state-based opacity has many different notions like current-state opacity [76], initial-state opacity [80], initial-and-final-state opacity [106], K-step opacity [76] and infinite-step opacity [79].

In this thesis, we propose event-based opacity notions which are different from language-based opacity and state-based opacity notions. They are motivated by the scenario in which the intruder is interested in detecting some specific unobservable events rather than strings or states in a system. In other words, the secret in event-based opacity consists of some unobservable events. Moreover, it is assumed that each secret event has its *critical horizon*, which is characterized by associating a positive integer to each secret event. Specifically, a positive integer K associated to a secret event defines the critical horizon that is the interval from the instant when the secret event occurs until the instant when the K-th observable event occurs after the secret. The intruder aims to establish within the critical horizon of a secret event if the secret has occurred, namely, to establish if a secret event has occurred at the latest when the K-th event is observed after the occurrence of the secret. A motivation example in the domain of business administration is illustrated in the thesis, where it could happen that if a secret is discovered too late (beyond its critical horizon), no strategy can be implemented by the intruder to take advantage of such a private information. Consequently, the event-based opacity of a system basically refers to the property that the intruder may never establish the occurrence of a secret event within its critical horizon.

We study event-based opacity using deterministic finite-state automata (DFA) as the reference formalism. First, we consider the case that the same critical horizon (i.e., the same positive integer K) is associated to each secret event and thus propose the notion of *K-observation event-opacity*, which becomes *infinite-observation event-*

opacity in the case that K is infinite. We observe that the notions of K-observation and infinite-observation event-opacity share some similarity with (K-step) diagnosability [83, 127]. Thus, the relationship among them is analyzed in our work. In simple words, diagnosability indicates that, every time a fault occurs, as long as a sufficiently long observable word is produced after that, the fault can be detected. In contrast, infinite-observation event-opacity implies that, every time a secret occurs, no matter how long the observable word produced after that is, the secret can never be discovered. If we do not distinguish between secret events and fault events, infinite-observation event-opacity is a sufficient condition for a system not being diagnosable. Now, let us focus on K-step diagnosability and K-observation event-opacity. The former requires that, every time a fault occurs, at the latest when the K-th observable event occurs after that, a fault can be detected. In contrast, the latter requires that, every time a secret occurs, starting from that moment, until the K-th observable event occurs, it can never be established that a secret has occurred in the middle of the past K+1 observable events. It is shown in the work that K-observation event-opacity is a sufficient condition for the system not being K-step diagnosable. In addition, K-observation event-opacity is incomparable with diagnosability. Later on, we generalize the problem setting assuming that different critical horizons (i.e., different positive integers) may be associated to different secret events. In this scenario, the notion of *event-opacity* is presented. Furthermore, we also consider the case where the intruder may distinguish among secret events, which motivates us to propose the notion of *combinational event-opacity* w.r.t. multiple sets of secret events. Next, we develop methods for the verification of the proposed properties. To do so, we construct verifiers that allow us to do the verification by checking nodes in them.

1.2 Main Contributions and Organization

The main contributions and the organization of this thesis are summarized as follows. We notice that the content of Chapters III and VI is published in *Information Sciences* and *IEEE Transactions on Automatic Control*, respectively; see [124] and [125], the content of Chapter V is accepted by *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, and the content of Chapters IV and VII is under review by *IEEE Transactions on Automatic Control*.

Chapter II Basic Notions

In this chapter, we recall basic notions used in this thesis, including the notion of languages, graph theory, two modelling formalisms, i.e., Automata and PN, and control policies in the framework of PN.

Chapter III Forbidden state problem of DES with unobservable and uncontrollable events

In this chapter, we use PN as a modelling tool to deal with the forbidden state problem of DES in the presence of both unobservable and uncontrollable events, assuming that unobservable events are uncontrollable.

First of all, it is proved that two state specifications are equivalent if their admissible marking sets coincide. Motivated by this result, we focus on studying how to compute optimal policies with respect to a state specification that is an *admissible GMEC*. Thanks to many approaches in the literature that allow one to efficiently transform an arbitrary GMEC into an admissible one with the admissible marking set unchanged, the proposed result remains useful in the more general case of arbitrary GMECs. Specifically, focusing on ordinary PNs subject to an admissible GMEC, we propose an optimal control policy whose computation mainly lies in the computation of the *unobservable minimal decrease*, a parameter depending on the current observation and the given GMEC. A procedure to compute such a parameter with polynomial complexity is proposed provided that a particular subnet, called *observation subnet*, is acyclic, *backward-conflict and backward-concurrent free* (BBF). As a result, under such assumptions, the optimal control policy could be computed with polynomial complexity.

Chapter IV Forbidden state problem of DES vulnerable to network attacks

In this chapter, we address the forbidden state problem of DES assuming that events are all controllable and observable but the system is vulnerable to network attacks. We consider the so-called *sensor-reading disguising attacks* (*SD-attacks* for short) that may disguise the occurrence of an event as another by tampering with the sensor-readings in sensor communication channels. In particular, we use PNs as a reference formalism to model a plant that is allowed to be bounded or unbounded and assume a control specification in terms of a GMEC. We propose three different methods to derive on-line control policies. The first two lead to an *optimal* (i.e., *maximally permissive*) policy but are computationally inefficient when applied to large-size systems. On the contrary, the third method computes a policy with timely response even for large-size systems but at the expense of optimality.

Chapter V Liveness enforcement on DES vulnerable to network attacks

In this chapter, we study the problem of liveness enforcement on DES still assuming that events are all controllable and observable but the system is vulnerable to *SD-attacks*. Specifically, we consider the plant modelled as a bounded PN and the control specification consisting in liveness enforcing. Based on repeatedly computing a more restrictive liveness-enforcing supervisor under no attack and constructing a so-called *basic supervisor*, an off-line method that synthesizes a liveness-enforcing supervisor tolerant to an SD-attack is proposed.

Chapter VI Verification of fault-predictability

In this chapter, we study the verification of fault-predictability in bounded and unbounded DES modeled by labeled PNs. An approach based on the construction of a *Predictor Net* and a *Predictor Graph* is proposed. In particular, a necessary and sufficient condition for fault-predictability is derived by characterizing the structure of the Predictor Graph. Furthermore, two rules are proposed to reduce the size of a given PN, which allow one to analyze the fault-predictability of the original net by verifying the fault-predictability of the reduced net.

Chapter VII Event-based opacity and its verification

In this chapter, we use deterministic finite-state automata (DFA) as the reference formalism, proposing four notions of event-based opacity, namely, *K-observation event-opacity*, *infinite-observation event-opacity*, *event-opacity* and *combinational* *event-opacity*. In simple words, these properties characterize situations in which an intruder, based on a partial observation of the system evolution, may never establish the occurrence of a secret event within its critical horizon. The critical horizon of a secret event is characterized by a positive integer, e.g., *K*, which defines the horizon from the instant when the secret event occurs until the instant when the *K*-th observable event occurs after the secret. A motivation example is presented in the chapter, relative to a business company taking decisions, some of which should remain secret to external observers. In addition, the relationship between *K*-observation/infinite-observation event-opacity and (*K*-step) diagnosability is analyzed. Moreover, appropriate verifiers are proposed to verify the proposed four properties.

Chapter VIII   Conclusions and future work

In this chapter, we conclude the thesis and discuss several potential future lines of research.

CHAPTER II

Basic Notions

In this chapter, we recall basic notions [15, 38] used in this thesis. Section 2.1 introduces the notion of languages, which provides a formal way to capture the behavior of a DES. Section 2.2 introduces a directed graph and characterizes some structures in it. Two modelling tools, Automata and PNs, are introduced in Section 2.3 together with their related notions. Section 2.4 mainly presents control policies in the framework of PNs. Note that we denote by $\mathbb{N}$ the set of natural numbers, $\mathbb{Z}$ the set of integers, and $\mathbb{Z}^+$ the set of positive integers.

2.1 Languages

Let Σ be an alphabet, i.e., a set of symbols. Σ^* is the *Kleene star* on Σ, defining the set of all finite-length strings over elements in Σ including the empty string ϵ. The *length* of a string is the number of symbols contained in it, counting multiple occurrences of the same symbol. We use $|\sigma|$ to denote the *length* of a string $\sigma\in\Sigma^*$ and it is $|\epsilon|=0$. Given a string $\sigma\in\Sigma^*$, a string $u\in\Sigma^*$ is

- a *prefix* of σ if $\exists v\in\Sigma^*$ such that $uv=\sigma$;
- a *suffix* of σ if $\exists v\in\Sigma^*$ such that $vu=\sigma$.

Given the set of events of a DES as an alphabet Σ, a *language* L defined over Σ is a set of strings formed from symbols in Σ, i.e., $L\subseteq\Sigma^*$. We denote the set of all *prefixes* of a string $\sigma\in\Sigma^*$ as $\overline{\sigma}$, i.e.,

$$\overline{\sigma}=\{u\in\Sigma^* \mid \exists v\in\Sigma^* \text{ s.t. } uv=\sigma\}.$$

The *prefix-closure* of a language $L\subseteq\Sigma^*$ is denoted as $\overline{L}$, that is,

$$\bar{L}=\bigcup_{\sigma\in L}\bar{\sigma}.$$

The *concatenation* of two languages $L_1, L_2 \subseteq \Sigma^*$ is denoted as L_1L_2, that is,

$$L_1L_2=\{\sigma\in\Sigma^* \mid (\sigma=\sigma_1\sigma_2) \wedge (\sigma_1\in L_1) \wedge (\sigma_2\in L_2)\}.$$

Given a language $L \subseteq \Sigma^*$ and a string $\sigma\in L$, the *post-language* of L after σ is denoted as L/σ, that is,

$$L/\sigma=\{u\in\Sigma^* \mid \sigma u\in L\}.$$

2.2 Graph Theory

A *directed graph* is an ordered pair $G=(X, E)$ comprising a set X of *nodes* (or *vertices*) together with a set E of ordered pairs of nodes, called *directed arcs*. Given a node $x\in X$, ${}^\bullet x=\{y \mid (y, x)\in E\}$ is the set of *inputs* of x and $x^\bullet=\{y \mid (x, y)\in E\}$ is the set of *outputs* of x. A node x is said to be a *sink node* if $x^\bullet=\emptyset$ and is said to be a *source node* if ${}^\bullet x=\emptyset$. A sequence of nodes $\pi=x_1x_2\ldots x_n\in X^*$ is called

- a *path* if $\forall i\in\{1, 2, ..., n-1\}$, $x_{i+1}\in x_i^\bullet$;
- a *cycle* if π is a path with $x_1=x_n$;
- an *elementary path* if π is a path and its nodes are all different (except, perhaps, x_1 and x_n).

A *strongly connected component* (SCC) of G is a subgraph of G that is strongly connected and maximal.

2.3 Formalisms

Automata and Petri nets (PNs) are two of the most popular mathematical tools for the modeling, control, analysis, and performance evaluation of DES. In this thesis, Chapters III-VI use PNs as the reference formalism and Chapter VII uses Automata as the reference formalism. We note that Automata and PNs are both directed graphs. In particular, PNs are bipartite graphs with two kinds of nodes, i.e., *places* and *transitions*. The formal definitions of Automata and PNs are presented as follows.

2.3.1 Automata

A *deterministic finite-state automaton* (DFA) is

$$G=(X, \Sigma, \delta, x_0),$$

where

- X is the finite set of *states*;
- Σ is the set of *events*;
- δ: $X\times\Sigma\to X$ is the deterministic *transition function*;
- $x_0\in X$ is the unique *initial state*.

Transition function δ is also extended to the domain $X\times\Sigma^*$ recursively such that $\forall x\in X$, $\delta(x, \epsilon)=x$ and $\delta(x, uv)=\delta(\delta(x, u), v)$, $\forall u\in\Sigma^*$, $v\in\Sigma$. The set $L(G)=\{\sigma\in\Sigma^*|\ \delta(x_0, \sigma)!\}$ is the *language* generated by G, where "!" means "is defined".

2.3.2 Petri Nets

A *Petri net* (PN) is a four-tuple $N= (P, T, F, W)$ where P and T are finite, nonempty, and disjoint sets: P is the set of *places* and T is the set of *transitions*. Graphically, places and transitions are represented by circles and bars, respectively. The set $F\subseteq(P\times T)\cup(T\times P)$ is the *flow relation*, which is represented by directed arcs from places to transitions or from transitions to places. W is a mapping that assigns a weight to each arc such that $W(x, y)>0$ if $(x, y)\in F$, and $W(x, y)=0$ otherwise, where $x, y\in P\cup T$. N is said to be *ordinary*, denoted as $N=(P, T, F)$, if $\forall(x, y)\in F$, $W(x, y)=1$.

A *marking* or *state* of a PN N is a mapping m: $P\to\mathbb{N}$. Generally, m is also denoted by the multi-set notation $\sum_{p\in P} m(p)p$, where $m(p)$ is the number of *tokens* in place p at m. For instance, $m=[1, 0, 3, 0]^T$ is denoted by $m=p_1+3p_3$. A place p is said to be *marked* at m if $m(p)>0$. The *initial marking* of a PN is denoted as m_0 and (N, m_0) is called a *net system* with initial marking m_0.

The *incidence matrix* of N is a matrix $[N]$: $P\times T\to\mathbb{Z}$ such that $[N](p, t)=W(t, p)-W(p, t)$, $\forall p\in P$, $\forall t\in T$. For a place p (transition t), its incidence vector, i.e., a row (column) in $[N]$, is denoted by $[N](p, \cdot)$ ($[N](\cdot, t)$).

Given a node $x\in P\cup T$, the set of *inputs* of x is ${}^\bullet x=\{y\in P\cup T\ |(y, x)\in F\}$, while the set of *outputs* of x is $x^{\bullet}=\{y\in P\cup T\,|(x, y)\in F\}$. Furthermore, $\forall X\subseteq P\cup T$, ${}^{\bullet}X=\cup_{x\in X}{}^{\bullet}x$ and $X^{\bullet}=\cup_{x\in X}x^{\bullet}$.

A transition t is *enabled* at marking m, denoted as $m[t\rangle$, if $\forall p\in{}^{\bullet}t$, $m(p)\geq W(p, t)$. The set of transitions enabled at m is denoted by $En(m)$. Transition t can *fire* at m if it is enabled at m. If t fires at m and a marking m' is reached, we denote this as $m[t\rangle m'$ and it holds that $m'(p)=m(p)+[N](p, t)$, $\forall p\in P$. Given a transition sequence $\sigma=t_1t_2...t_k\in T^*$, σ is *enabled* at m, denoted as $m[\sigma\rangle$, if there exist markings m_1, m_2, … , m_{k-1} such that $m[t_1\rangle m_1[t_2\rangle m_2[t_3\rangle$ … $m_{k-1}[t_k\rangle$. We denote as $m[\sigma\rangle m_k$ if m_k is reached by firing σ at m.

Given a net system (N, m_0), we use $R(N, m_0)$ to denote the set of all reachable markings of N from m_0, i.e., $R(N, m_0)=\{m|\ \exists\sigma\in T^*, m_0[\sigma\rangle m\}$, called the *reachability set* of (N, m_0) and $L(N, m_0)$ the set of all transition sequences of N that are enabled at m_0, i.e., $L(N, m_0)=\{\sigma\in T^*|\ m_0[\sigma\rangle\}$, called the *language* of (N, m_0).

A PN system (N, m_0) is *bounded* if the number of tokens in each place does not exceed a finite number $B\in\mathbb{Z}^+$ for any marking $m\in R(N, m_0)$. Otherwise, it is *unbounded*.

Given a transition sequence $\sigma\in T^*$, the *Parikh vector* of σ is $\vec{\sigma}: T\rightarrow\mathbb{N}$, which maps t in T to the number of occurrences of t in σ. For an arbitrary finite transition sequence $\sigma\in T^*$ such that $m[\sigma\rangle m'$, it holds $m'=m+[N]\cdot\vec{\sigma}$.

A transition t is *live* at a marking m if $\forall m'\in R(N, m)$, $\exists m''\in R(N, m')$ such that $m''[t\rangle$. A net system (N, m_0) is *live* if $\forall t\in T$, t is live at m_0. A transition t is *dead* at a marking m if $\forall m'\in R(N, m)$, t is not enabled at m'. A state m is said to be a *global deadlock state* if no transition can fire at m. Usually, a global deadlock state is simply said to be a *deadlock*.

2.4 Supervisory Control

In this thesis, the supervisory control of DES is investigated in the framework of PNs (see Chapters III-V). Thus, we introduce *control policies* (or equivalently, *supervisors*) in the framework of PNs, taking into account the existence of both *unobservable* and *uncontrollable* transitions. In addition, we introduce the notion of *Generalized Mutual*

Exclusion Constraints (GMECs), which often characterize a state specification on a PN system.

2.4.1 Uncontrollable and Unobservable Transitions

In a practical scenario, a PN may contain *unobservable* and/or *uncontrollable transitions*. As discussed in the Introduction, the presence of unobservable transitions is due to facts like limited sensor technique, high cost of sensor installation, and failure of communication and sensors. There are also many reasons for the presence of uncontrollable transitions: it is inherently uncontrollable (for example, it models a fault or a change of sensor-readings not due to a command); its firing cannot be prevented due to hardware or actuation limitations; or it is modeled as uncontrollable by choice, as for example when the transition has high priority and thus should not be disabled. Thus, there is a partition of the set of transitions in a PN such that

$$T = T_c \cup T_{uc},$$

where

- T_c is the set of *controllable transitions*;
- T_{uc} is the set of *uncontrollable transitions*.

Also, there is a partition of the set of transitions such that

$$T = T_o \cup T_{uo},$$

where

- T_o is the set of *observable transitions*;
- T_{uo} is the set of *unobservable transitions*.

The *observation function* O: $T^* \rightarrow T_o^*$ is defined recursively such that

1) $O(\varepsilon)=\varepsilon$;

2) $\forall \sigma \in T^*$, $t \in T$, $O(\sigma t) = \begin{cases} O(\sigma)t & \text{if } \quad t \in T_o \\ O(\sigma) & \text{otherwise} \end{cases}$.

Given a net system (N, m_0), the *observed language* of (N, m_0) is

$$L_o(N, m_0) = \{\delta = O(\sigma) \mid \sigma \in L(N, m_0)\}.$$

Given a transition sequence $\sigma\in L(N, m_0)$, $\delta=O(\sigma)$ is said to be the *observed sequence* of σ. Given an observed sequence $\delta\in L_o(N, m_0)$, we define $O_N^{-1}(\delta)=O^{-1}(\delta)\cap L(N, m_0)$ the set of transition sequences *consistent* with δ in the net system (N, m_0).

2.4.2 Supervisors

We introduce two definitions of *control policies* (or *supervisors*). One is used in Chapter III and the other is used in Chapters IV and V. We notice that the two definitions are essentially the same, only differing in expressions.

In Chapter III, we define a *control action* as a function u: $T_c\to\{0, 1\}$, associating a binary value to each controllable transition such that $\forall t\in T_c$, $u(t)=1$ if t is permitted to fire and $u(t)=0$ otherwise. The set of all such control actions is denoted by U. A *control policy* (or *supervisor*) of a PN system (N, m_0) is a function

$$\rho: L_o(N, m_0) \to U,$$

i.e., ρ associates a control action to each observed sequence of the system. In simple words, a control policy decides which controllable transitions should be prevented from firing or permitted to fire according to the current observation.

In Chapters IV and V, a *control policy* (or *supervisor*) of a PN system (N, m_0) is a function

$$\rho: L_o(N, m_0)\to 2^{T_c},$$

i.e., ρ associates to each observation a set of controllable transitions. We call $\rho(\delta)$ the *disabled set*, indicating the set of controllable transitions that should be prevented from firing at the observation $\delta\in L_o(N, m_0)$.

We denote the system (N, m_0) controlled by supervisor ρ as $(N, m_0)|_\rho$. The language of $(N, m_0)|_\rho$, denoted by $L(N, m_0)|_\rho$, is defined recursively as follows (the first definition of ρ is used here as an example):

1. $\varepsilon\in L(N, m_0)|_\rho$;
2. $[(\sigma\in L(N, m_0)|_\rho) \wedge (\sigma t\in L(N, m_0)) \wedge(t\in T_{uc} \vee\rho[O(\sigma)](t)=1)] \Leftrightarrow [\sigma t\in L(N, m_0)|_\rho]$.

Accordingly, the observed language and the reachability set of $(N, m_0)|_\rho$, denoted by $L_o(N, m_0)|_\rho$ and $R(N, m_0)|_\rho$, are defined as:

$$L_o(N, m_0)|_\rho=\{O(\sigma)|\sigma\in L(N, m_0)|_\rho\};$$

$$R(N, m_0)|_\rho=\{m \mid \exists\sigma\in L(N, m_0)|_\rho, \text{ s.t. } m_0[\sigma\rangle m\}.$$

Given two supervisors ρ_1 and ρ_2 associated with a PN system (N, m_0), we denote $\rho_1\succeq\rho_2$ (resp., $\rho_1\preceq\rho_2$), if

$$L(N, m_0)|_{\rho 1} \supseteq L(N, m_0)|_{\rho 2}$$

$$(\text{resp., } L(N, m_0)|_{\rho 1} \subseteq L(N, m_0)|_{\rho 2}).$$

Moreover, ρ_1 is said to be *more permissive* (resp., *restrictive*) than ρ_2, denoted as $\rho_1\succ\rho_2$ (resp., $\rho_1\prec\rho_2$), if

$$L(N, m_0)|_{\rho 1} \supset L(N, m_0)|_{\rho 2}$$

$$(\text{resp., } L(N, m_0)|_{\rho 1} \subset L(N, m_0)|_{\rho 2}).$$

Given a control specification, a supervisor is said to be *optimal* (i.e., *maximally permissive*) if it guarantees that its controlled system meets the control specification and any other supervisor more permissive than the supervisor cannot guarantee that its controlled system meets the control specification. We note that the optimal supervisor is not necessarily a unique one. In this thesis, we will consider two control specifications. One is a state specification and the other is the property of liveness. The notion of *optimal* supervisors is detailed as follows in the two control specifications.

1) State specification

Given a PN system (N, m_0) and a state specification $\mathcal{Q}\subseteq\mathbf{N}^{|P|}$, a control policy ρ is said to be *acceptable* if $R(N, m_0)|_\rho\subseteq \mathcal{Q}$; and is said to be *optimal* if 1) $R(N, m_0)|_\rho\subseteq \mathcal{Q}$; and 2) $\forall\rho'\succ\rho$, $R(N, m_0)|_{\rho'}\not\subseteq\mathcal{Q}$.

2) Liveness

Given a PN system (N, m_0), a supervisor ρ is said to be a *liveness-enforcing* supervisor if the controlled system $(N, m_0)|_\rho$ is live, i.e., $\forall t\in T$, $\forall m'\in R(N, m_0)|_\rho$, $\exists m''\in R(N, m')|_\rho$, such that $m''[t\rangle$. A liveness-enforcing supervisor ρ is said to be *optimal* if, for any supervisor ρ' more permissive than ρ, the controlled system $(N, m_0)|_{\rho'}$ is not live.

We finally note that in Chapters IV and V where we investigate the control policies in the presence of attacks, the *observed language* of a system will be redefined since attacks may change observations.

2.4.3 Generalized Mutual Exclusion Constraints (GMEC)

A *Generalized Mutual Exclusion Constraint* (GMEC) on the state space of a PN N is defined as a pair (ω, k), where ω: $P\to\mathbf{N}$ is a weight vector and $k\in\mathbf{N}$, which identifies the *legal marking set*:

$$\mathcal{L}_{(\omega, k)}=\{m\in\mathbf{N}^{|P|} \mid \omega\cdot m\le k\}.$$

Moreover, we denote $\varpi = \omega\cdot[N]$. Given a transition t, it holds that $\varpi(t)=\omega\cdot m'-\omega\cdot m$, for any pair of markings m and m' such that m' is the marking reached from m by firing t. In other words, $\varpi(t)$ is the change of the ω-weighted sum of tokens in the net caused by firing t. Note that, given an ordinary PN, it holds that $\varpi(t)=\sum_{p\in t^\bullet}\omega(p)-\sum_{p\in {}^\bullet t}\omega(p)$ for any $t\in T$.

Given a set of GMECs $W=\{(\omega_1, k_1), (\omega_2, k_2), \ldots, (\omega_n, k_n)\}$, where $n\in\mathbb{Z}^+$, the *conjunction* of GMECs in W is denoted as $\wedge W$, which defines the *legal marking set*: $\mathcal{L}_{\wedge W}=\bigcap_{(\omega,k)\in W}\mathcal{L}_{(\omega,k)}$.

We finally note that a GMEC is also simply said to be a *linear constraint* in the literature.

CHAPTER III

Forbidden State Problem of DES with Unobservable and Uncontrollable Events

3.1 Introduction

In this chapter, we use PN as a modelling tool to deal with the forbidden state problem of DES in the presence of both unobservable and uncontrollable events, assuming that unobservable events are uncontrollable. First of all, it is proved that two state specifications are equivalent if their admissible marking sets coincide. Motivated by this result, we focus on studying how to compute optimal policies with respect to a state specification that is an *admissible GMEC*. Thanks to many approaches in the literature that allow one to efficiently transform an arbitrary GMEC into an admissible one with the admissible marking set unchanged, the proposed result remains useful in the more general case of arbitrary GMEC. Specifically, focusing on ordinary PN subject to an admissible GMEC, we propose an optimal control policy whose computation mainly lies in the computation of the *unobservable minimal decrease*, a parameter depending on the current observation and the given GMEC. A procedure to compute such a parameter with polynomial complexity is proposed provided that a particular subnet, called *observation subnet*, is acyclic, *backward-conflict and backward-concurrent free* (BBF). As a result, under such assumptions, the optimal control policy could be computed with polynomial complexity.

This chapter is organized as follows. Section 3.2 recalls the notions used in the chapter. Section 3.3 investigates the equivalence between state specifications. Section

3.4 computes an optimal control policy for PN subject to an admissible GMEC with exponential complexity. The observation subnet is introduced in Section 3.5 and a procedure to compute the unobservable minimal decrease starting from the observation subnet is provided with polynomial complexity. Section 3.6 proposes an optimal and polynomial complexity control policy for a class of PN subject to an admissible GMEC. How to obtain an optimal policy for special classes of PN subject to an arbitrary GMEC is discussed in Section 3.7. Finally, Section 3.8 draws conclusions of this chapter.

We notice that the work of this chapter has been published in *Information Sciences*; see [124].

3.2 Preliminaries

In this section, we introduce an assumption made on uncontrollable and unobservable transitions and a class of PNs used in this chapter. We also note that in this chapter we simply use m_σ to denote the marking reached by firing a transition sequence $\sigma \in T^*$ at the initial marking m_0 of a PN, i.e., $m_0[\sigma\rangle m_\sigma$.

- **Assumption on uncontrollable and unobservable transitions**

In this chapter, we assume that an unobservable transition is also uncontrollable, i.e., $T_{uo} \subseteq T_{uc}$ and $T_c \subseteq T_o$. This assumption, which is common to most of the literature in this framework such as [1, 63, 68], is applicable to several real applications. A discussion in this respect can be found in [68]. Transitions in a PN can thus be divided into three categories: controllable transitions, uncontrollable but observable transitions, and unobservable transitions. Graphically, they are depicted as white, dashed, and black bars, respectively, as shown in Fig. 3.1.

Transitions:
▭ controllable (and observable)
▨ uncontrollable but observable
▬ unobservable (and uncontrollable)

Fig. 3.1 Three different kinds of transitions in this chapter

- **Backward-conflict and backward-concurrent free net**

Definition 3.1 [54]: A PN N=(P, T, F) is *backward-conflict and backward-concurrent free* (BBF) if $\forall p\in P$, $|{}^\bullet p|=1$ and $\forall t\in T$, $|t^\bullet|=1$.

3.3 Equivalence between State Specifications

In this section we investigate the equivalence between state specifications in the general setting that transitions may be uncontrollable and/or unobservable. The major consequence of this is that, if a control policy is optimal with respect to a certain state specification, it is also optimal with respect to any equivalent state specification. Note that, as in [63], we assume that the initial marking of the considered system is known.

3.3.1 Admissible Observed Sequence Set

This subsection introduces a notion called *admissible observed sequence set*. Before that, we recall “the attaching language” defined in [63], which is reformulated and renamed as “set of potential firing sequences” in the following just for better understanding.

Definition 3.2: Given a PN system (N, m_0) and a transition sequence $\sigma\in L(N, m_0)$,

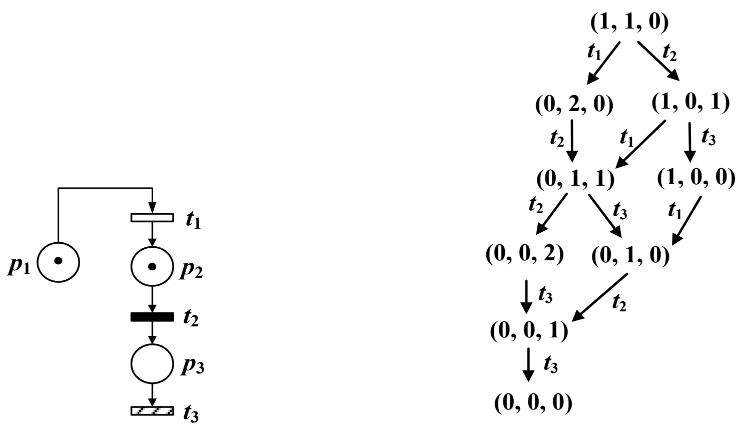
$$e(\sigma)=\{\sigma'\in L(N, m_0) \mid \sigma'=\sigma\sigma'', \text{ where } \sigma''\in T_{uc}^*\}$$

is called the set of *uncontrollable extending sequences* of σ. Furthermore, given an observed sequence $\delta\in L_o(N, m_0)$,

$$\Lambda(\delta)=\bigcup_{\sigma\in O_N^{-1}(\delta)} e(\sigma)$$

is called the set of *potential firing sequences* consistent with δ.

Example 3.1: Consider the PN system (N, m_0) in Fig. 3.2(a), whose *reachability graph* [69] is shown in Fig. 3.2(b). Let us consider the observed sequence $\delta=t_3t_1\in L_o(N, m_0)$. It is $O_N^{-1}(\delta)=\{t_2t_3t_1, t_2t_3t_1t_2\}$. Thus, $\Lambda(\delta)=e(t_2t_3t_1)\cup e(t_2t_3t_1t_2)=\{t_2t_3t_1, t_2t_3t_1t_2, t_2t_3t_1t_2t_3\}\cup\{t_2t_3t_1t_2, t_2t_3t_1t_2t_3\}=\{t_2t_3t_1, t_2t_3t_1t_2, t_2t_3t_1t_2t_3\}$. ♦



(a) (b)

Fig. 3.2 (a) PN system (N, m_0) with $T_c=\{t_1\}$, $T_{uc}=\{t_2, t_3\}$, $T_o=\{t_1, t_3\}$, $T_{uo}=\{t_2\}$ and (b) its reachability graph

Definition 3.3: Given a PN system (N, m_0), a state specification $\mathcal{Q}$, and an observed sequence $\delta\in L_o(N, m_0)$, δ is called *admissible* if $\forall\delta'\in\bar{\delta}$, $\forall\sigma\in\Lambda(\delta')$, $m_\sigma\in\mathcal{Q}$. The *admissible observed sequence set* of (N, m_0) w.r.t. $\mathcal{Q}$ is denoted as $\mathcal{G}(\mathcal{Q})$, i.e.,

$$\mathcal{G}(\mathcal{Q})=\{\delta\in L_o(N, m_0)|\ \forall\delta'\in\bar{\delta},\ \forall\sigma\in\Lambda(\delta'),\ m_\sigma\in\mathcal{Q}\}.$$

Example 3.2: Consider again the PN system (N, m_0) in Fig. 3.2(a). Let $\mathcal{Q}=\{m\in\mathbf{N}^{|P|}|\ m(p_2)\leq 1\}$ be the state specification. Let us establish if the observed sequence $\delta=t_3$ is admissible. Clearly, $\bar{\delta}=\{\varepsilon, t_3\}$. For each $\delta'\in\bar{\delta}$, we check if $\forall\sigma\in\Lambda(\delta')$, $m_\sigma\in\mathcal{Q}$ holds: 1) Consider $\delta'=\varepsilon$. It is $\Lambda(\varepsilon)=\{\varepsilon, t_2, t_2t_3\}$ and $m_\varepsilon=(1, 1, 0)$, $m_{t2}=(1, 0, 1)$, $m_{t2t3}=(1, 0, 0)$. Hence, $\forall\sigma\in\Lambda(\varepsilon)$, $m_\sigma\in\mathcal{Q}$. 2) Now, consider $\delta'=t_3$. It is $\Lambda(t_3)=\{t_2t_3\}$ and $m_{t2t3}=(1, 0, 0)$. Hence, $\forall\sigma\in\Lambda(t_3)$, $m_\sigma\in\mathcal{Q}$. As a result, $\delta=t_3$ is admissible. On the contrary, $\delta=t_1$ is not admissible since $\exists t_1\in\bar{\delta}$, $t_1\in\Lambda(t_1)$, such that $m_{t1}=(0, 2, 0)\notin\mathcal{Q}$. Furthermore, we can see that $\mathcal{G}(\mathcal{Q})=\{\varepsilon, t_3, t_3t_1, t_3t_1t_3\}$. ♦

Next, we present some results related to control policies. We note that, although conditions for the existence and the optimality of a control policy have been established in the literature [38, 49], Theorems 3.1 and 3.2 in the following give new characterizations of them based on the admissible observed sequence set.

Theorem 3.1: Given a PN system (N, m_0) and a state specification $\mathcal{Q}$, there exists a control policy ρ such that $R(N, m_0)|_\rho \subseteq \mathcal{Q}$ *iff* $\varepsilon \in \mathcal{G}(\mathcal{Q})$, i.e., $\forall \sigma \in \Lambda(\varepsilon)$, $m_\sigma \in \mathcal{Q}$.

Proof: Trivially follows from the supervisory control theory in [49]. ■

Theorem 3.2: Given a PN system (N, m_0), a state specification $\mathcal{Q}$, and a control policy ρ, ρ is optimal *iff* $L_o(N, m_0)|_\rho = \mathcal{G}(\mathcal{Q})$.

Proof: It is easy to see that

$$R(N, m_0)|_\rho = \{m_\sigma | \ \sigma \in O_N^{-1}(\delta),\ \delta \in L_o(N, m_0)|_\rho\} = \{m_\sigma | \ \sigma \in \Lambda(\delta),\ \delta \in L_o(N, m_0)|_\rho\}. \quad (1)$$

(=>) Since $L_o(N, m_0)|_\rho = \mathcal{G}(\mathcal{Q})$, then $\forall \delta \in L_o(N, m_0)|_\rho$, it holds $\forall \sigma \in \Lambda(\delta)$, $m_\sigma \in \mathcal{Q}$. Hence, $R(N, m_0)|_\rho \subseteq \mathcal{Q}$ by (1). Let ρ' be a policy more permissive than ρ. Clearly, a policy permits or forbids controllable transitions to fire. Furthermore, by assumption, controllable transitions are observable. Hence, it can be concluded that $L_o(N, m_0)|_{\rho'} \supset L_o(N, m_0)|_\rho$ since $L(N, m_0)|_{\rho'} \supset L(N, m_0)|_\rho$. Let $\delta \in L_o(N, m_0)|_{\rho'} \setminus L_o(N, m_0)|_\rho$. $\delta \notin \mathcal{G}(\mathcal{Q})$ since $\delta \notin L_o(N, m_0)|_\rho$. Hence, $\exists \delta' \in \overline{\delta}$, $\exists \sigma \in \Lambda(\delta')$, such that $m_\sigma \notin \mathcal{Q}$. Clearly, $\delta' \in L_o(N, m_0)|_{\rho'}$. It is $m_\sigma \in R(N, m_0)|_{\rho'}$ by (1). Since $m_\sigma \notin \mathcal{Q}$, $R(N, m_0)|_{\rho'} \not\subset \mathcal{Q}$. Thus, ρ is optimal.

(<=) We know that 1) $R(N, m_0)|_\rho \subseteq \mathcal{Q}$; and 2) $\forall \rho' \succ \rho$, $R(N, m_0)|_{\rho'} \not\subset \mathcal{Q}$. Based on 1), it is $\{m_\sigma | \ \sigma \in \Lambda(\delta),\ \delta \in L_o(N, m_0)|_\rho\} \subseteq \mathcal{Q}$. Note that $\forall \delta \in L_o(N, m_0)|_\rho$, $\overline{\delta} \subseteq L_o(N, m_0)|_\rho$. Hence, it can be seen that $L_o(N, m_0)|_\rho \subseteq \mathcal{G}(\mathcal{Q})$. Based on 2), it is $\forall \rho' \succ \rho$, $\{m_\sigma | \ \sigma \in \Lambda(\delta),\ \delta \in L_o(N, m_0)|_{\rho'}\} \not\subset \mathcal{Q}$. In other words, $\forall \rho' \succ \rho$, $\exists \delta \in L_o(N, m_0)|_{\rho'}$ such that $\delta \notin \mathcal{G}(\mathcal{Q})$. Hence, we may conclude that $L_o(N, m_0)|_\rho = \mathcal{G}(\mathcal{Q})$. ■

The above theorem implies that the admissible observed sequence set exactly characterizes the observed sequence set of a PN system when it is supervised by an optimal control policy. Therefore, the role of the admissible observed sequence set in supervisory control of PNs with both uncontrollable and unobservable transitions is analogous to the role of the *admissible marking set* [49] in supervisory control of entirely observable but partially controllable PNs.

3.3.2 Condition for State Specifications Being Equivalent

By Theorem 3.2, the equivalence of state specifications can be formally defined in terms of the admissible observed sequence set as follows.

Definition 3.4: Given a PN system (N, m_0) and two state specifications $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we say that $\mathcal{Q}_1$ is *equivalent* to $\mathcal{Q}_2$, denoted as $\mathcal{Q}_1 \equiv \mathcal{Q}_2$, if $\mathcal{G}(\mathcal{Q}_1)=\mathcal{G}(\mathcal{Q}_2)$.

Now, we recall the notion of the admissible marking set.

Definition 3.5 [49]: Given a PN system (N, m_0) and a state specification $\mathcal{Q}$, the *admissible marking set* w.r.t. $\mathcal{Q}$ is

$$\mathcal{A}(\mathcal{Q})=\{m\in \mathcal{Q} \,|\, \forall\sigma\in T_{uc}^* \text{ s.t. } m[\sigma\rangle m', \text{ it holds } m'\in \mathcal{Q}\}.$$

The following proposition reveals that the set $\mathcal{G}(\mathcal{Q})$ can be characterized in terms of the admissible marking set of $\mathcal{Q}$.

Proposition 3.1: Given a PN system (N, m_0), a state specification $\mathcal{Q}$, it holds that $\mathcal{G}(\mathcal{Q})=\{\delta\in L_o(N, m_0) |\, \forall \delta'\in \overline{\delta}, \forall\sigma\in \Lambda(\delta'), m_\sigma\in \mathcal{A}(\mathcal{Q})\}$.

Proof: Let $X=\{\delta\in L_o(N, m_0) |\, \forall \delta'\in \overline{\delta}, \forall\sigma\in \Lambda(\delta'), m_\sigma\in \mathcal{A}(\mathcal{Q})\}$. We prove that $\mathcal{G}(\mathcal{Q})\supseteq X$ and $\mathcal{G}(\mathcal{Q})\subseteq X$.

--$\mathcal{G}(\mathcal{Q})\supseteq X$: Clearly, $\mathcal{A}(\mathcal{Q}) \subseteq \mathcal{Q}$. Hence, $\forall\delta\in X$, it holds that $\forall\delta'\in \overline{\delta}$, $\forall\sigma\in\Lambda(\delta')$, $m_\sigma\in \mathcal{Q}$. Therefore, $\mathcal{G}(\mathcal{Q})\supseteq X$.

--$\mathcal{G}(\mathcal{Q})\subseteq X$: By contradiction, suppose that $\exists\delta\in \mathcal{G}(\mathcal{Q})$, such that $\delta\notin X$. It means that $\exists\delta'\in \overline{\delta}$, $\exists\sigma\in\Lambda(\delta')$, such that $m_\sigma\notin \mathcal{A}(\mathcal{Q})$. Then, a marking outside of $\mathcal{Q}$ can be reached from m_σ by firing uncontrollable transitions since $m_\sigma\notin \mathcal{A}(\mathcal{Q})$. In other words, $\exists\sigma'\in T_{uc}^*$ such that $m_\sigma[\sigma'\rangle m'$, $m'\notin \mathcal{Q}$. Let $\sigma''=\sigma\sigma'$. Clearly, m' is exactly $m_{\sigma''}$. Besides, we can see that $\sigma''\in\Lambda(\delta')$. Since $\delta'\in\bar{\delta}$, $\sigma''\in\Lambda(\delta')$, and $m_{\sigma''}\notin\mathcal{Q}$, δ is not admissible, which contradicts the fact that $\delta\in\mathcal{G}(\mathcal{Q})$. Hence, $\mathcal{G}(\mathcal{Q})\subseteq X$. ■

Theorem 3.3: Given a PN system (N, m_0) and two state specifications $\mathcal{Q}_1$ and $\mathcal{Q}_2$, $\mathcal{Q}_1\equiv\mathcal{Q}_2$ if $\mathcal{A}(\mathcal{Q}_1)=\mathcal{A}(\mathcal{Q}_2)$.

Proof: According to Proposition 3.1, $\mathcal{G}(\mathcal{Q}_1)=\mathcal{G}(\mathcal{Q}_2)$ since $\mathcal{A}(\mathcal{Q}_1)=\mathcal{A}(\mathcal{Q}_2)$. Hence, $\mathcal{Q}_1\equiv\mathcal{Q}_2$ by Definition 3.4. ■

We know that for entirely observable but partially controllable PNs, two state specifications are equivalent if they have the same admissible marking set [49]. Theorem 3.3 claims that, even when PNs contain both uncontrollable and unobservable transitions, this result still holds. Consequently, it allows us to focus on proposing optimal policies for the state specification being an admissible GMEC. Indeed, if a certain GMEC is preliminarily transformed into an equivalent admissible GMEC, then an optimal control policy for the given GMEC could be obtained implementing a control policy that is optimal for the equivalent admissible GMEC.

3.4 Optimal Exponential-complexity Control Policy for PN Subject to an Admissible GMEC

In this section, we focus on PNs with the state specification being an admissible GMEC.

Recall that a GMEC (ω, k) characterizes the legal marking set $\mathcal{L}_{(\omega, k)}=\{m\in\mathbb{N}^{|P|} \mid \omega\cdot m\leq k\}$. For simplicity, we use $\mathcal{A}_{(\omega, k)}$ and $\mathcal{G}_{(\omega, k)}$ to respectively denote the admissible marking set and the admissible observed sequence set of a PN system w.r.t. $\mathcal{L}_{(\omega, k)}$. Besides, we define some transition sets w.r.t. a PN and a given GMEC.

Definition 3.6: Let N be a PN subject to a GMEC (ω, k). We define

- $\mathcal{T}_{\omega}^{+}=\{t\in T\mid\varpi(t)>0\}$ the set of *increasing transitions* w.r.t. (ω, k);

- $\mathcal{T}_\omega^- = \{t \in T | \varpi(t) < 0\}$ the set of *decreasing transitions* w.r.t. (ω, k); and

- $\mathcal{T}_\omega^\# = \mathcal{T}_\omega^- \cap T_{uo}$ the set of *unobservable decreasing transitions* w.r.t. (ω, k).

Now, we present the notion of admissible GMECs.

Definition 3.7 [68]: Let N be a PN subject to a GMEC (ω, k). (ω, k) is said to be an *admissible GMEC* if $\forall t \in T_{uc}$, $\varpi(t) \leq 0$.

Remark 3.1: The above definition of admissible GMEC takes into account uncontrollable transitions only. It is different from the one provided by Luo *et al.* in [63], where they not only consider uncontrollable transitions but also enforce particular limitations on unobservable transitions. ♣

In the following, we introduce two key notions in developing control policies in this work.

Definition 3.8: Let (N, m_0) be a PN system subject to a GMEC (ω, k), and $\delta \in L_o(N, m_0)$ be an observed sequence.

$$\Phi_{\omega,\delta} = \min_{\sigma \in O_N^{-1}(\delta)} \sum_{t \in \mathcal{T}_\omega^\#} |\varpi(t)| \cdot \vec{\sigma}(t)$$

is called the *unobservable minimal decrease* w.r.t. (ω, k) induced by δ and

$$\Psi_{\omega,\delta} = \varpi \cdot \vec{\delta}$$

is called the *observable change* w.r.t. (ω, k) induced by δ.

In other words, when the sequence δ is observed, $\Phi_{\omega,\delta}$ is the minimal decrease of the ω-weighted sum of tokens in the PN system induced by the firing of unobservable decreasing transitions; $\Psi_{\omega,\delta}$ is the change of the ω-weighted sum of tokens in the PN system induced by the firing of observable transitions.

Example 3.3: Consider again the PN system (N, m_0) in Fig. 3.2(a). Let (ω, k): $m(p_2) \leq 1$ be the GMEC and $\delta = t_3 t_1$ be the observed sequence. It is easy to compute that $\Psi_{\omega,\delta} = \varpi \cdot \vec{\delta} = [1, -1, 0] \cdot [1, 0, 1]^T = 1$. Now, let us focus on the computation of $\Phi_{\omega,\delta}$. We can see that $O_N^{-1}(\delta) = \{\sigma_1, \sigma_2\}$, where $\sigma_1 = t_2 t_3 t_1$ and $\sigma_2 = t_2 t_3 t_1 t_2$. Note that $\mathcal{T}_\omega^\# = \{t_2\}$. Hence, $\Phi_{\omega,\delta} = \min\{\sum_{t \in \mathcal{T}_\omega^\#} |\varpi(t)| \cdot \vec{\sigma}_1(t),\ \sum_{t \in \mathcal{T}_\omega^\#} |\varpi(t)| \cdot \vec{\sigma}_2(t)\} = \min\{|\varpi(t_2)| \cdot \vec{\sigma}_1(t_2),\ |\varpi(t_2)| \cdot \vec{\sigma}_2(t_2)\} = 1$. ♦

Property 3.1: Let (N, m_0) be a PN system subject to a GMEC (ω, k). It is $\Phi_{\omega,\varepsilon}=0$ and $\Psi_{\omega,\varepsilon}=0$.

Proposition 3.2: Let (N, m_0) be a PN system subject to an admissible GMEC (ω, k) and $\delta\in L_o(N, m_0)$. It holds that $\max_{\sigma\in\Lambda(\delta)} \varpi\cdot\vec{\sigma} = \max_{\sigma\in O_N^{-1}(\delta)} \varpi\cdot\vec{\sigma} = \Psi_{\omega,\delta} - \Phi_{\omega,\delta}$.

Proof: See Appendix. ■

In the rest of this chapter, even if not explicitly claimed, it is assumed that $\varepsilon\in \mathcal{G}_{(\omega, k)}$ holds for any given PN system and any given GMEC (ω, k).

In the following, an on-line control policy ρ (denoted as Policy 3.1) is proposed for PNs subject to an admissible GMEC. It can be explained as follows. Every time the firing of a transition is detected, an observed sequence δ is obtained. According to δ, the policy ρ determines a control action $u=\rho(\delta)$ that establishes which controllable transitions are forbidden to fire, or alternatively, permitted to fire. Specifically, for each controllable transition t, we decide that it is forbidden to fire if $\Psi_{\omega,\delta'} - \Phi_{\omega,\delta'} > k-\omega\cdot m_0$, where $\delta'=\delta t$, and permitted to fire otherwise. Note that the policy ρ firstly determines the control action for the observed sequence being empty. In other words, when no transition in the PN system is observed, $u=\rho(\varepsilon)$ is determined in advance.

Policy 3.1: An on-line control policy ρ
Plant: A PN system (N, m_0); State specification: An admissible GMEC (ω, k).
Every time a transition sequence $\delta\in L_o(N, m_0)$ is observed, determine the corresponding control action $u=\rho(\delta)$ as follows: $\forall t\in T_c$, $u(t):=0$, if $\Psi_{\omega,\delta'} - \Phi_{\omega,\delta'} > k-\omega\cdot m_0$, where $\delta'=\delta t$, $u(t):=1$, otherwise.

Theorem 3.4: Given a PN system (N, m_0) and an admissible GMEC (ω, k), Policy 3.1 is optimal.

Proof: By Proposition 3.2, the decision condition in Policy 3.1 is equivalent to "$\max_{\sigma\in\Lambda(\delta')} \varpi\cdot\vec{\sigma} > k-\omega\cdot m_0$, where $\delta'=\delta t$". Furthermore, it is equivalent to "$\exists\sigma\in\Lambda(\delta t)$, s.t. $\varpi\cdot\vec{\sigma} > k-\omega\cdot m_0$". Since $\varpi\cdot\vec{\sigma} = \omega\cdot m_\sigma - \omega\cdot m_0$, the decision condition is further equivalent to "$\exists\sigma\in\Lambda(\delta t)$, s.t. $m_\sigma\notin\mathcal{L}_{(\omega,k)}$". Trivially, we can see that $L_o(N, m_0)|_\rho = \mathcal{G}_{(\omega,k)}$. According to Theorem 3.2, Policy 3.1 is optimal. ■

Remark 3.2: Every time Policy 3.1 establishes whether a controllable transition t is permitted to fire in the case that a transition sequence δ is observed, it requires the computation of the unobservable minimal decrease $\Phi_{\omega,\delta'}$ and the observable change $\Psi_{\omega,\delta'}$. The computation of $\Psi_{\omega,\delta'}$ is quite easy, while the computation of $\Phi_{\omega,\delta'}$ by Definition 3.8 is complicated. In more detail, according to Definition 3.8, $\Phi_{\omega,\delta'}$ is computed as follows: first, we enumerate all transition sequences consistent with δ', i.e., we compute the set $O_N^{-1}(\delta')$. Next, for each $\sigma\in O_N^{-1}(\delta')$, we compute $\sum_{t\in T_\omega^\#}|\varpi(t)|\cdot\vec{\sigma}(t)$, i.e., the decrease of the ω-weighted sum of tokens in the PN system induced by the firing of unobservable decreasing transitions in σ. Finally, we pick the minimal decrease that is exactly the value of $\Phi_{\omega,\delta'}$.

We can see that, for arbitrary PN systems, the computation of $O_N^{-1}(\delta')$ is of exponential complexity w.r.t. the number of unobservable transitions in a PN. As a result, the computational complexity of Policy 3.1 is also exponential w.r.t. the number of unobservable transitions in a PN. Clearly, the key to reducing the complexity of Policy 3.1 is to reduce the complexity related to the computation of the unobservable minimal decrease $\Phi_{\omega,\delta'}$. In the following sections, we propose an efficient approach to compute $\Phi_{\omega,\delta'}$ in PNs with specific structures and thereby develop an efficient and optimal control policy for a special class of PNs subject to an admissible GMEC. ♣

3.5 Observation Subnet and Efficient Computation of Unobservable Minimal Decrease

In this section we introduce a series of notions and results that are the basis for the computation of an efficient and optimal control policy for a special class of PNs subject to an admissible GMEC, which is presented in the next section.

In particular, here we define a special subnet of the given net under control, called the *observation subnet*. Besides, we define the so-called *extended observation subnet* that is indeed the observation subnet together with all its related transitions in the given net. Provided that the observation subnet satisfies some specific assumptions, a procedure to compute the unobservable minimal decrease is provided with polynomial complexity by looking at the extended observation subnet. The efficient procedure allows us to compute an optimal control policy with polynomial complexity in the next section.

Note that in the rest of this chapter, PNs are assumed to be ordinary even without explicit claim.

3.5.1 Observation Subnet

Let us first introduce some notation. Given a path π in a PN, $P(\pi)$ and $T(\pi)$ denote the sets of places and transitions in π, respectively. Furthermore, given a set of paths, denoted as Π, $P(\Pi)$ and $T(\Pi)$ denote the sets of all places and transitions in all paths in Π, respectively, i.e., $P(\Pi)=\cup_{\pi\in\Pi}P(\pi)$ and $T(\Pi)=\cup_{\pi\in\Pi}T(\pi)$. Besides, an *unobservable path* is a path in which each transition is unobservable.

Definition 3.9: Given a PN $N=(P, T, F)$ and an unobservable transition $t\in T_{uo}$, a path $\pi=t_{i1}p_{j1}t_{i2}p_{j2}\ldots t_{in}p_{jn}$ is an *elementary observation path* w.r.t. t if

1) $t_{i1}=t$;
2) $T(\pi)\subseteq T_{uo}$;
3) $p_{jn}^{\bullet}\cap T_o\neq\emptyset$.

The set of all elementary observation paths w.r.t. t is denoted as Π_t. Given a set of unobservable transitions $T_X \subseteq T_{uo}$, it is defined that $\Pi(T_X)=\bigcup_{t\in T_X}\Pi_t$.

Example 3.4: Consider the PN in Fig. 3.3 and the unobservable transition t_3. By Definition 3.9, $\pi_1=t_3p_3t_4p_5t_5p_6$ is an elementary observation path w.r.t. t_3 since all transitions in π_1 are unobservable and an output transition of p_6 is observable. Furthermore, $\Pi_{t3}=\{\pi_1, \pi_2\}$, where $\pi_2=t_3p_7$. Similarly, it is $\Pi_{t10}=\emptyset$. ♦

Definition 3.10: Let $N=(P, T, F)$ be a PN subject to a GMEC (ω, k). The *observation subnet* w.r.t. (ω, k) is defined as $N_\omega^\beta=(P_\omega^\beta, T_\omega^\beta, F_\omega^\beta)$ where

- $P_\omega^\beta= P^0\cup P^1\cup\ldots\cup P^n$, where

--$P^0=\{p\in P \mid p$ can reach $p'\in P(\Pi(T^0))$ via an unobservable path π s.t. $T(\pi)\cap T(\Pi(T^0))=\emptyset$, where $T^0=\mathcal{T}_\omega^\#\}$,

--$P^1=\{p\in P \mid p$ can reach $p'\in P(\Pi(T^1))$ via an unobservable path π s.t. $T(\pi)\cap T(\Pi(T^1))=\emptyset$, where $T^1=(P^{0\bullet}\setminus{}^\bullet P^0)\cap T_{uo}\}$,

--…

--$P^n=\{p\in P \mid p$ can reach $p'\in P(\Pi(T^n))$ via an unobservable path π s.t. $T(\pi)\cap T(\Pi(T^n))=\emptyset$, where $T^n=(P^{n-1\bullet}\setminus{}^\bullet P^{n-1})\cap T_{uo}\}$,

- T_ω^β is the set of unobservable input transitions of places in P_ω^β, and
- F_ω^β is the restriction of F to $(P_\omega^\beta\times T_\omega^\beta)\cup(T_\omega^\beta\times P_\omega^\beta)$.

By default, a place p can reach itself by path $\pi=p$ and this path can be regarded as an unobservable path. In other words, $P(\Pi(T^i)) \subseteq P^i$ holds, $\forall i\in \{0, 1, \ldots, n\}$.

We explain in detail how to determine P_ω^β by Definition 3.10. First, let $T^0=\mathcal{T}_\omega^\#$. P^0 can be determined by the following steps:

1) Compute the set of elementary observation paths w.r.t. T^0, i.e., $\Pi(T^0)$;

2) Determine the sets of places and transitions in $\Pi(T^0)$, i.e., $P(\Pi(T^0))$ and $T(\Pi(T^0))$;

3) Search places not in $P(\Pi(T^0))$ but that can reach a place in $P(\Pi(T^0))$ via an unobservable path π satisfying $T(\pi)\cap T(\Pi(T^0))=\emptyset$.

Consequently, P^0 is obtained. It consists of all places in $P(\Pi(T^0))$ and all places considered in Step 3. Next, let $T^1=(P^{0\bullet}\setminus{}^\bullet P^0)\cap T_{uo}$; P^1 can be determined following steps analogous to those used to determine P^0. Similarly, P^2, P^3, … , P^n can be determined one after the other. Note that once a place set P^i is determined to be empty, it is $P^j=\varnothing$, $\forall j>i$. This implies that $P_\omega^\beta= P^0\cup P^1\cup\ldots\cup P^i$ once $P^i=\varnothing$ is determined.

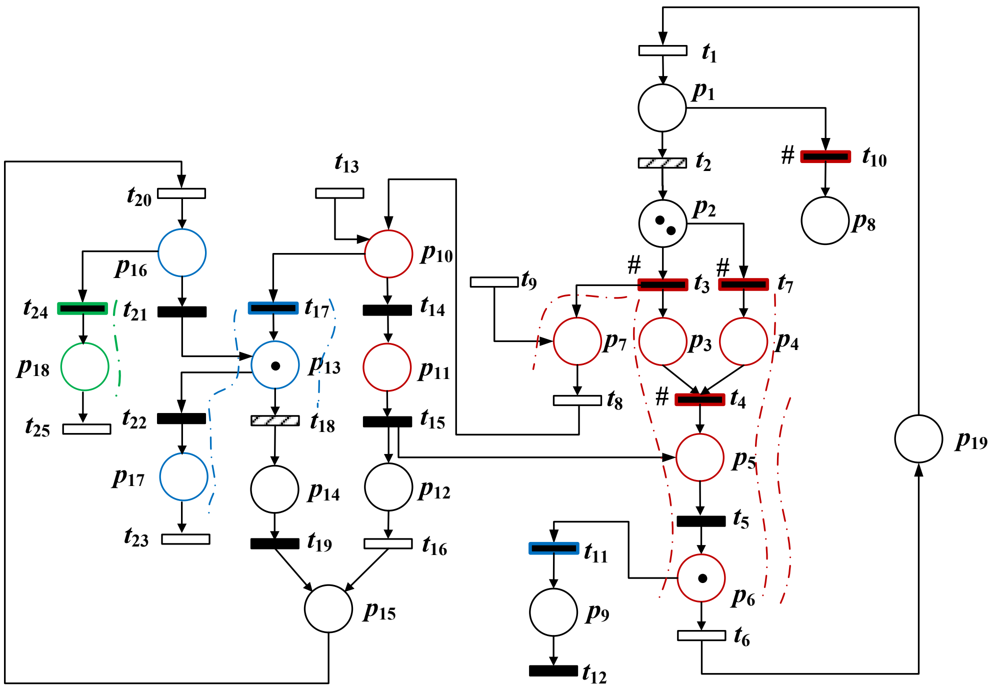


Fig. 3.3 PN system (N, m_0) subject to a GMEC (ω, k): $m(p_3)+2m(p_2)+3m(p_1)\leq 8$

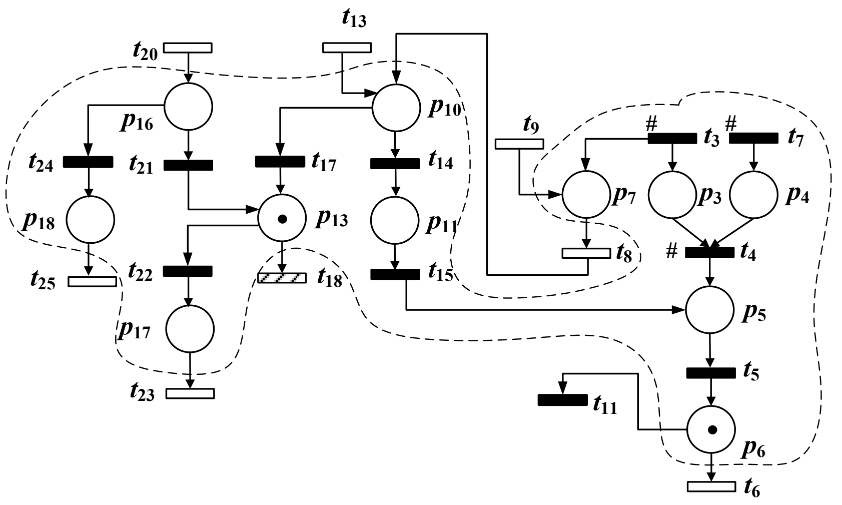


Fig. 3.4 Extended observation subnet of the PN in Fig. 3.3
(The part inside the dashed line is the observation subnet)

Example 3.5: Consider the PN system (N, m_0) in Fig. 3.3 subject to the GMEC (ω, k): $m(p_3)+2m(p_2)+3m(p_1)\leq 8$. It is $\mathcal{T}_\omega^{\#}=\{t_3, t_4, t_7, t_{10}\}$. Note that in figures we use "#" to mark transitions in $\mathcal{T}_\omega^{\#}$ to make them more visible. Now, let us compute the observation subnet N_ω^β by Definition 3.10. We determine the place set P_ω^β as follows.

1) Compute P^0. $T^0=\mathcal{T}_\omega^{\#}$. First, $\Pi(T^0)=\{\pi_1\text{-}\pi_4\}$ is computed, where $\pi_1=t_3p_7$, $\pi_2=t_3p_3t_4p_5t_5p_6$, $\pi_3=t_4p_5t_5p_6$, $\pi_4=t_7p_4t_4p_5t_5p_6$. Hence, $P(\Pi(T^0))=\{p_3\text{-}p_7\}$ and $T(\Pi(T^0))=\{t_3\text{-}t_5, t_7\}$. Next, we search places who can reach a place in $P(\Pi(T^0))$ via an unobservable path π such that $T(\pi)\cap T(\Pi(T^0))=\emptyset$. It is $P^0=P(\Pi(T^0))\cup\{p_{10}, p_{11}\}=\{p_3\text{-}p_7, p_{10}, p_{11}\}$.

2) Compute P^1. It is $T^1=(P^{0\bullet}\setminus{}^\bullet P^0)\cap T_{uo}=\{t_{11}, t_{17}\}$ and $\Pi(T^1)=\{\pi_5, \pi_6\}$, where $\pi_5=t_{17}p_{13}$ and $\pi_6=t_{17}p_{13}t_{22}p_{17}$. By searching places as shown above, we obtain $P^1=P(\Pi(T^1))\cup\{p_{16}\}=\{p_{13}, p_{16}, p_{17}\}$.

3) Compute P^2. It is $T^2=(P^{1\bullet}\setminus{}^\bullet P^1)\cap T_{uo}=\{t_{24}\}$ and $\Pi(T^2)=\{\pi_7\}$, where $\pi_7=t_{24}p_{18}$. Similarly, we obtain that $P^2=P(\Pi(T^2))=\{p_{18}\}$.

4) Compute P^3. It is $T^3=(P^{2\bullet}\setminus{}^\bullet P^2)\cap T_{uo}=\emptyset$. Hence, $P^3=\emptyset$ and we stop.

Therefore, $P_\omega^\beta=P^0\cup P^1\cup P^2=\{p_3\text{-}p_7, p_{10}, p_{11}, p_{13}, p_{16}\text{-}p_{18}\}$. The observation subnet is shown in Fig. 3.4 inside the dashed line. ♦

Finally, we analyze the computational complexity of computing the observation subnet. It is clear that its complexity mainly depends on the complexity of computing P_ω^β and the computation of P_ω^β consists in the computation of P^0, P^1, …, P^i, …. We can see that every time we compute the place set P^i, we need to search all elementary observation paths starting from transitions of T^i, i.e., we need to compute the set $\Pi(T^i)$, which is in general complicated. We note that $\Pi(T^i)$ is computed to determine places and transitions in $\Pi(T^i)$, i.e., determine $P(\Pi(T^i))$ and $T(\Pi(T^i))$. Actually, $P(\Pi(T^i))$ and $T(\Pi(T^i))$ could be computed by the following procedure:

1) Initialization: $P_X:=\emptyset$ and $T_X:=T^i$;
2) Search all output places of T_X and then search all output unobservable transitions from those places. In other words, it is $P_X:= T_X^\bullet$ and then $T_X:= T_X\cup(P_X^\bullet\cap T_{uo})$.

Repeat the above search procedure until no new places and transitions can be found;

3) Denote N_X the subnet of the given PN N determined by P_X and T_X computed above. Delete from N_X those transitions that are sink transitions in N_X and those places that are sink places in N_X and have no observable output transitions in N. Update N_X as the resultant net. Repeat the above deleting procedure until no new places and transitions can be deleted.

Clearly, places and transitions in the resultant net are exactly those constituting the sets $P(\Pi(T^i))$ and $T(\Pi(T^i))$, respectively. We can see that the above procedure to compute $P(\Pi(T^i))$ and $T(\Pi(T^i))$ is of polynomial complexity with respect to the size of the PN model. Moreover, it is known that $P(\Pi(T^i))\subseteq P^i$. Then, the computation of P^i requires finding places in P^i but not in $P(\Pi(T^i))$. These places could be searched simply one after another following the *reverse* unobservable paths that start from the input unobservable transitions of places in $P(\Pi(T^i))$. Consequently, the computation of P^i is still of polynomial complexity with respect to the size of the PN model. We note that there exists a finite number n such that $P_\omega{}^\beta= P^0\cup P^1\cup\ldots\cup P^n$ since a PN has finite places and we always search places that have never been searched in a PN. As a result, we can conclude that the computation of the observation subnet is of polynomial complexity with respect to the size of the PN model.

Remark 3.3: We note that the observation subnet is defined for the purpose of studying the computation of unobservable minimal decrease. Roughly speaking, the unobservable minimal decrease is related to the "least" firing times of unobservable decreasing transitions, i.e., transitions in $\mathcal{T}_\omega{}^\#$. Hence, the observation subnet is constructed starting from unobservable decreasing transitions. Then, necessary unobservable paths are taken into consideration step by step, through which it can be inferred how many times the unobservable decreasing transitions definitely fire in advance when a transition sequence is observed. In order to efficiently compute the unobservable minimal decrease, it is necessary to avoid the exhaustive enumeration of all the possible transition sequences consistent with an observed sequence. To this aim, our basic idea is restricting the observation net to special structures and applying appropriate computations on it. In the next subsection, we show that, in the case that the observation subnet satisfies specific assumptions, a "bottom-up" method can be used to efficiently compute the "least" firing times of unobservable decreasing transitions when a sequence is observed. Consequently, the unobservable minimal decrease is also efficiently computed. ♣

3.5.2 Efficient Computation of Unobservable Minimal Decrease

The computation of the unobservable minimal decrease requires the preliminary definition of a special subnet related to the given GMEC. In the following, we call such a net *extended observation subnet*.

Definition 3.11: Let $N=(P, T, F)$ be a PN subject to a GMEC (ω, k) and $N_\omega^\beta=(P_\omega^\beta, T_\omega^\beta, F_\omega^\beta)$ be the observation subnet. The *extended observation subnet* w.r.t. (ω, k) is defined as $\overline{N_\omega^\beta}=\left(\overline{P_\omega^\beta},\overline{T_\omega^\beta},\overline{F_\omega^\beta}\right)$ such that $\overline{P_\omega^\beta}=P_\omega^\beta$, $\overline{T_\omega^\beta}={}^\bullet P_\omega^\beta\cup P_\omega^{\beta\bullet}$, and $\overline{F_\omega^\beta}$ is the restriction of F to $(\overline{P_\omega^\beta}\times\overline{T_\omega^\beta})\cup(\overline{T_\omega^\beta}\times\overline{P_\omega^\beta})$. Furthermore, $\mathcal{E}(N_\omega^\beta)=\overline{T_\omega^\beta}\setminus T_\omega^\beta$ is called the set of *external related transitions* of N_ω^β.

The extended observation subnet associated with the PN and the GMEC in Example 3.5, is shown in Fig. 3.4.

Now, we introduce some transition sets related to the extended observation subnet, which are fundamental in the computation of the unobservable minimal decrease. We use $\mathcal{T}_o^\beta$ and $\mathcal{T}_{uo}^\beta$ to denote the sets of observable and unobservable transitions, respectively, in an extended observation subnet $\overline{N_\omega^\beta}$. We can see that observable transitions in $\overline{N_\omega^\beta}$ are all external related transitions of the corresponding observation subnet, i.e., $\mathcal{T}_o^\beta\subseteq\mathcal{E}(N_\omega^\beta)$. We thus further divide $\mathcal{T}_o^\beta$ into two sets, namely, $\mathcal{T}_{o(in)}^\beta$ and $\mathcal{T}_{o(out)}^\beta$, denoting the sets of *input* and *output observable transitions* of N_ω^β, respectively, i.e., $\mathcal{T}_{o(in)}^\beta=\mathcal{T}_o^\beta\cap{}^\bullet P_\omega^\beta$ and $\mathcal{T}_{o(out)}^\beta=\mathcal{T}_o^\beta\cap P_\omega^{\beta\bullet}$. Note that it is possible that $t\in\mathcal{T}_o^\beta$ is both an input and an output observable transition of an observation subnet. Besides, we note that $\mathcal{T}_{uo}^{\beta}$ includes all transitions in N_ω^{β} and may include some transitions outside of N_ω^{β}.

Example 3.6: Consider the extended observation subnet in Fig. 3.4. Clearly, $\mathcal{T}_o^{\beta}\subseteq\mathcal{I}(N_\omega^{\beta})$. Besides, $\mathcal{T}_o^{\beta}{}_{(in)}=\{t_8, t_9, t_{13}, t_{20}\}$ and $\mathcal{T}_o^{\beta}{}_{(out)}=\{t_6, t_8, t_{18}, t_{23}, t_{25}\}$. $\mathcal{T}_{uo}^{\beta}$ consists of all transitions in N_ω^{β} and the transition t_{11}. ♦

Some other concepts and results need to be introduced before presenting the computation method.

Definition 3.12: Let (N, m_0) be a PN system, $\delta\in L_o(N, m_0)$ and $t\in T_{uo}$. The *least firing times* of t induced by δ is

$$\varphi_\delta(t)=\min_{\sigma\in O_N^{-1}(\delta)}\vec{\sigma}(t).$$

In words, $\varphi_\delta(t)$ denotes the least times that the unobservable transition t fires when the transition sequence δ is observed.

Given a marking m and a transition sequence σ, we use m^{β} and $\vec{\sigma}^{\,\beta}$ to denote the marking and the Parikh vector obtained restricting m and $\vec{\sigma}$ to the place set and the transition set of the extended observation subnet $\overline{N_\omega^{\beta}}$, respectively. Besides, we use $[N]^{\beta}$ to denote the restriction of the incidence matrix $[N]$ to the extended observation subnet $\overline{N_\omega^{\beta}}$.

Definition 3.13: Let (N, m_0) be a PN system subject to a GMEC (ω, k) and $\delta\in L_o(N, m_0)$. We define

$$\mathcal{B}_v^{\beta}(\delta)=\{\vec{\sigma}^{\,\beta}\mid\sigma\in O_N^{-1}(\delta)\text{ and}\nexists\,\sigma'\in O_N^{-1}(\delta)\text{ s.t. }\vec{\sigma}'^{\,\beta}<\vec{\sigma}^{\,\beta}\}$$

as the set of *basic partial sequence vectors* consistent with δ w.r.t. $\overline{N_\omega^{\beta}}$; and

$$\mathcal{B}_m^{\beta}(\delta)=\{m_0^{\beta}+[N]^{\beta}\cdot\vec{v}\mid\vec{v}\in\mathcal{B}_v^{\beta}(\delta)\}$$

as the set of *basic partial marking* consistent with δ w.r.t. $\overline{N_\omega^{\beta}}$.

We observe that, by the firing law of transitions in PN, given a transition sequence $\sigma\in L(N, m_0)$, it holds that $m_\sigma^{\beta}=m_0^{\beta}+[N]^{\beta}\cdot\vec{\sigma}^{\beta}$. Thus, it is true that $\mathcal{B}_m^{\beta}(\delta)=\{m_\sigma^{\beta}\mid\sigma\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma}^{\beta}\in\mathcal{B}_v^{\beta}(\delta)\}$.

It could happen that $\mathcal{B}_v^{\beta}(\delta)$ contains more than one vector and thus $\mathcal{B}_m^{\beta}(\delta)$ might also contain more than one marking. The following property shows that $\mathcal{B}_v^{\beta}(\delta)$ contains only one vector if and only if there exists a sequence $\sigma\in O_N^{-1}(\delta)$ such that each unobservable transition in the extended observation subnet fires a number of times equal to its least times.

Property 3.2: $|\mathcal{B}_v^{\beta}(\delta)|=1$ *iff* $\exists\sigma\in O_N^{-1}(\delta)$ such that $\vec{\sigma}(t)=\varphi_\delta(t)$, $\forall t\in \mathcal{T}_{uo}^{\beta}$.

Proof: Straightforward from Definitions 3.12 and 3.13. ■

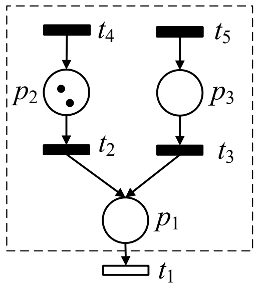


Fig. 3.5 An extended observation subnet
(The part inside the dashed line is the observation subnet)

Example 3.7: Consider the extended observation subnet in Fig. 3.5. Let $\delta=t_1t_1t_1$ be an observed sequence. We can see that $\exists\sigma_1\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma_1}^{\beta}=(\vec{\sigma_1}(t_1), \vec{\sigma_1}(t_2), \vec{\sigma_1}(t_3), \vec{\sigma_1}(t_4), \vec{\sigma_1}(t_5))^T=(3, 0, 3, 0, 3)^T$ and $\nexists\sigma'\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma'}^{\beta}<\vec{\sigma_1}^{\beta}$. Moreover, $\exists\sigma_2\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma_2}^{\beta}=(\vec{\sigma_2}(t_1), \vec{\sigma_2}(t_2), \vec{\sigma_2}(t_3), \vec{\sigma_2}(t_4), \vec{\sigma_2}(t_5))^T=(3, 3, 0, 1, 0)^T$ and $\nexists\sigma'\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma'}^{\beta}<\vec{\sigma_2}^{\beta}$. Hence, $\mathcal{B}_v^{\beta}(\delta)=\{(3, 0, 3, 0, 3)^T, (3, 3, 0, 1, 0)^T\}$. Clearly, $\mathcal{T}_{uo}^{\beta}=\{t_2\text{-}t_5\}$. By Definition 3.12, we have $\varphi_\delta(t_1)=3$ and $\varphi_\delta(t_2)=\varphi_\delta(t_3)=\varphi_\delta(t_4)=\varphi_\delta(t_5)=0$. For this case, we can see that $\nexists\sigma\in O_N^{-1}(\delta)$ s.t. $\vec{\sigma}^{\beta}=(\vec{\sigma}(t_1), \vec{\sigma}(t_2), \vec{\sigma}(t_3), \vec{\sigma}(t_4), \vec{\sigma}(t_5))^T=(3, 0, 0, 0, 0)^T$. ♦

The following theorem indicates that $|\mathcal{B}_v^{\beta}(\delta)|=1$ and $|\mathcal{B}_m^{\beta}(\delta)|=1$ when the observation subnet satisfies certain conditions.

Theorem 3.5: Let (N, m_0) be a PN system subject to a GMEC (ω, k) such that the observation subnet N_ω^β is acyclic and BBF, and $\delta\in L_o(N, m_0)$. Then, it is $|\mathcal{B}_v^\beta(\delta)|=1$ and $|\mathcal{B}_m^\beta(\delta)|=1$.

Proof: See Appendix. ■

In the remainder of this chapter, when given a PN system (N, m_0) subject to a GMEC (ω, k) such that N_ω^β is acyclic and BBF, and an observed sequence $\delta\in L_o(N, m_0)$, we use $\mathcal{M}_{\omega,\delta}$ to denote the unique basic partial marking consistent with δ w.r.t. $\overline{N_\omega^\beta}$, i.e., $\{\mathcal{M}_{\omega,\delta}\}=\mathcal{B}_m^\beta(\delta)$. We observe that the basic partial marking $\mathcal{M}_{\omega,\delta}$ is reached when each unobservable transition in the extended observation subnet fires its least firing times in the case that the firing of the transition sequence δ is observed.

Property 3.3: Let (N, m_0) be a PN system subject to a GMEC (ω, k) such that N_ω^β is acyclic and BBF. Then, it is $\mathcal{M}_{\omega,\varepsilon}= m_0^\beta$.

Proof: Straightforward from Definition 3.13. ■

Property 3.4: Let (N, m_0) be a PN system subject to a GMEC (ω, k) such that N_ω^β is acyclic and BBF and $\delta\in L_o(N, m_0)$. Then, it is $\Phi_{\omega,\delta}=\sum_{t\in T_\omega^\#}|\varpi(t)|\cdot\varphi_\delta(t)$.

Proof: Since N_ω^β is acyclic and BBF, $\exists\sigma\in O_N^{-1}(\delta)$ such that $\vec{\sigma}(t)=\varphi_\delta(t)$, $\forall t\in \mathcal{T}_{uo}^\beta$. Due to Definition 3.8, it is easy to see $\Phi_{\omega,\delta}=\sum_{t\in T_\omega^\#}|\varpi(t)|\cdot\varphi_\delta(t)$. ■

In the following, we present a method to compute the unobservable minimal decrease, which applies under the assumption that the observation subnet is acyclic and BBF. It also returns the basic partial marking, which is necessary to implement the proposed optimal control policy.

Given two sequences δ, $\delta'\in L_o(N, m_0)$, we call δ' a *next observed sequence* of δ (or δ the *last observed sequence* of δ') if $\exists t\in T_o$ such that $\delta'=\delta t$. In what follows, we present *Function NextUnobseValue*, by which the unobservable minimal decrease and the basic partial marking w.r.t. an observed sequence can be computed, provided that those w.r.t. its last observed sequence are known. *Function BottomUpHandle* is called in *Function NextUnobseValue*.

```
$(\Phi, \mathcal{M})$=NextUnobseValue $(\Phi_{\omega,\delta}, \mathcal{M}_{\omega,\delta}, t)$
Input: Unobservable minimal decrease $\Phi_{\omega,\delta}$, basic partial marking $\mathcal{M}_{\omega,\delta}$, and $t \in T_o$;
Output: $\Phi$ and $\mathcal{M}$. /* The output $\Phi$ and $\mathcal{M}$ are $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$, where $\delta' = \delta t$.*/
1)  $(\Phi, \mathcal{M}) := (\Phi_{\omega,\delta}, \mathcal{M}_{\omega,\delta})$;
    /* $\Phi$ and $\mathcal{M}$ are global variables that can be updated in Function BottomUpHandle. */
2)  if $t \in \mathcal{T}_o^{\beta}{}_{(out)}$ then
3)      BottomUpHandle($t$);
4)  end if
5)  if $t \in \mathcal{T}_o^{\beta}{}_{(in)}$ then
6)      for each $p \in t^{\bullet} \cap P_{\omega}^{\beta}$ do
7)          $\mathcal{M}(p) := \mathcal{M}(p)+1$;
8)      end for
9)  end if
10) Output: $\Phi$ and $\mathcal{M}$.
```

```
BottomUpHandle($t$)
Input: a transition $t$.
1)  if ${}^{\bullet}t \cap P_{\omega}^{\beta} \neq \emptyset$ then
2)    for each $p \in {}^{\bullet}t \cap P_{\omega}^{\beta}$ do
3)      if $\mathcal{M}(p) \neq 0$ then
4)          $\mathcal{M}(p) := \mathcal{M}(p)-1$;
5)      else
6)          let $\{t'\} := {}^{\bullet}p \cap T_{\omega}^{\beta}$;   /* $N_{\omega}^{\beta}$ is BBF. */
7)          if $t' \in \mathcal{T}_{\omega}^{\#}$ then
8)              $\Phi := \Phi + |\varpi(t')|$;
9)          end if
10)         BottomUpHandle($t'$);
11)     end if
12)   end for
13) end if
```

Let us briefly explain *Function NextUnobseValue*. First, variables Φ and $\mathcal{M}$ are initialized at $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$. Then, four cases may occur, depending on the current transition t.

Case 1: $t \notin \mathcal{T}_{o}^{\beta}$. In this case, Φ and $\mathcal{M}$ are directly output without further computations.

Case 2: $t \in \mathcal{T}_{o}^{\beta}{}_{(out)} \setminus \mathcal{T}_{o}^{\beta}{}_{(in)}$. *BottomUpHandle*($t$) is called. Essentially, its execution determines from bottom up which transitions in N_{ω}^{β} definitely fire a number of times larger than their least firing times induced by δ in the case that the firing of t is observed following δ. Such a computation relies on the basic partial marking. Specifically, when the place $p \in {}^{\bullet}t \cap P_{\omega}^{\beta}$ contains no token at $\mathcal{M}$, we can determine that its single input unobservable transition definitely fires one more time since otherwise t cannot fire. Moreover, every time it is determined that a transition definitely fires one more time, Function *BottomUpHandle* is called again to determine whether its upper transitions in N_{ω}^{β} definitely fire one more time. We can see that Φ and $\mathcal{M}$ are accordingly updated during the execution of *BottomUpHandle*(t). Φ and $\mathcal{M}$ are output when the function ends.

Case 3: $t \in \mathcal{T}_{o}^{\beta}{}_{(in)} \setminus \mathcal{T}_{o}^{\beta}{}_{(out)}$. In this case, $\mathcal{M}$ is updated by adding one token to each output place of t in N_{ω}^{β}, while Φ remains unchanged.

Case 4: $t \in \mathcal{T}_{o}^{\beta}{}_{(in)} \cap \mathcal{T}_{o}^{\beta}{}_{(out)}$. In this case, *BottomUpHandle*(t) is called first and then $\mathcal{M}$ is updated again by adding one token to each output place of t in N_{ω}^{β}.

Proposition 3.3: Let (N, m_0) be a PN system subject to a GMEC (ω, k) such that N_{ω}^{β} is acyclic and BBF, and δ, $\delta' \in L_o(N, m_0)$ such that $\delta' = \delta t$. *Function NextUnobseValue* outputs $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$ when $\Phi_{\omega,\delta}$, $\mathcal{M}_{\omega,\delta}$, and t are the inputs.

Proof: See Appendix. ■

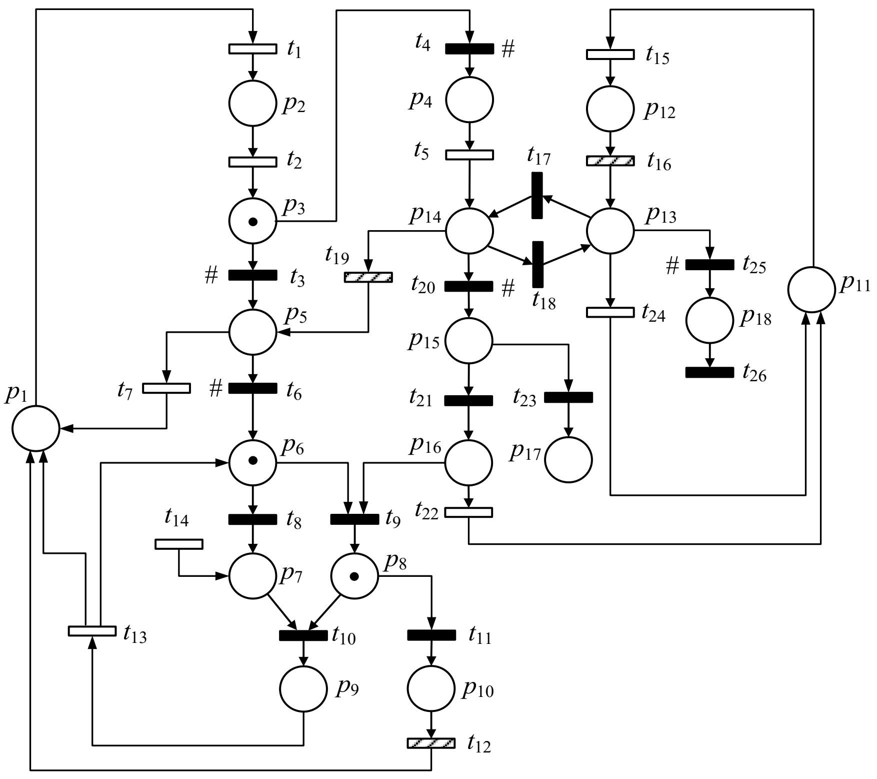


Fig. 3.6 PN system (N, m_0) subject to a GMEC (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})+2m(p_{13})+2m(p_{12})\leq 8$

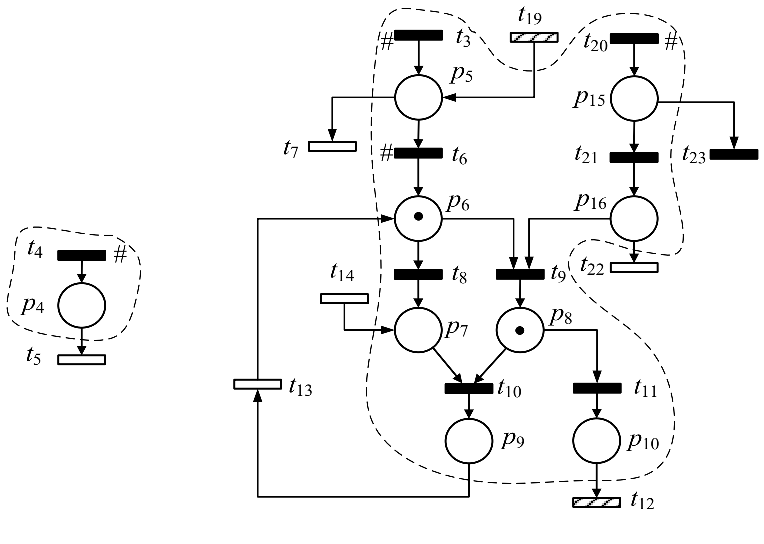


Fig. 3.7 Extended observation subnet of the PN in Fig. 3.6 (The part inside the dashed line is the observation subnet)

The following example is given to illustrate Function *NextUnobseValue*. Here, we define a *reverse path* as a string $x_1x_2\ldots x_n\in(P\cup T)^*$ such that $x_{i+1}\in{}^\bullet x_i$ for all i=1, 2, …, n−1.

Example 3.8: Consider the PN system (N, m_0) in Fig. 3.6. Let (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})+2m(p_{13})+2m(p_{12})\leq 8$ be a GMEC on it. Then, $\mathcal{T}_\omega^\#=\{t_3, t_4, t_6, t_{20}, t_{25}\}$. By Definitions 3.10 and 3.11, we obtain the observation subnet N_ω^β and the extended one $\overline{N_\omega^\beta}$, as depicted in Fig. 3.7. N_ω^β is clearly acyclic and BBF. Hence, we compute $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$ for $\delta=t_{12}$ by Function *NextUnobseValue*. It is easy to see $\Phi_{\omega,\varepsilon}=0$ and $\mathcal{M}_{\omega,\varepsilon}=m_0^\beta=p_6+p_8$. We input $\Phi_{\omega,\varepsilon}$, $\mathcal{M}_{\omega,\varepsilon}$ and t_{12} into Function *NextUnobseValue*. First, $\Phi=\Phi_{\omega,\varepsilon}=0$ and $\mathcal{M}=\mathcal{M}_{\omega,\varepsilon}=p_6+p_8$. Next, *BottomUpHandle*(t_{12}) is called since t_{12} is an output observable transitions of N_ω^β. Following the reverse path $\pi_1=t_{12}p_{10}t_{11}p_8$, we can see that the firing of t_{12} consumes a token from p_8 and t_{11} definitely fires one time. Hence, $\mathcal{M}$ is updated to $\mathcal{M}=p_6$ and Φ remains unchanged since $t_{11}\notin\mathcal{T}_\omega^\#$. As a result, $\Phi_{\omega,\delta}=0$ and $\mathcal{M}_{\omega,\delta}=p_6$. Consider the observed sequence $\delta'=t_{12}t_{13}$. We can compute $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$ using Function *NextUnobseValue* since $\delta=t_{12}$ is the last observed sequence of δ' and $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$ are known. We input $\Phi_{\omega,\delta}$, $\mathcal{M}_{\omega,\delta}$, and t_{13} into Function *NextUnobseValue*. First, $\Phi=\Phi_{\omega,\delta}=0$ and $\mathcal{M}=\mathcal{M}_{\omega,\delta}=p_6$. Next, *BottomUpHandle*(t_{13}) is called since t_{13} is an output observable transitions of N_ω^β. Following the reverse path $\pi_2=t_{13}p_9t_{10}p_7t_8p_6$, we can see that the firing of t_{13} consumes a token from p_6 and both t_{10} and t_8 are required to fire one time. Accordingly, $\mathcal{M}$ is updated to $\mathcal{M}=0$. We observe that t_{10} has another input place p_8. Hence, following the reverse path $\pi_3=p_8t_9p_6t_6p_5t_3$, we can see that all transitions in π_3 are also required to fire one time since t_{10} is required to fire one time. In particular, since $t_6, t_3\in\mathcal{T}_\omega^\#$, Φ is updated with $\Phi=\Phi+|\varpi(t_6)|+|\varpi(t_3)|=2$. Besides, we observe that t_9 has another input place p_{16}. Similarly, following the reverse path $\pi_4=p_{16}t_{21}p_{15}t_{20}$, all transitions in π_4 are required to fire one time since t_9 is required to fire one time. Since $t_{20}\in \mathcal{T}_{\omega}^{\#}$, Φ is thus updated again with Φ=Φ+$|\varpi(t_{20})|$=4. Then, considering that t_{13} is also an input observable transition of N_{ω}^{β}, $\mathcal{M}$ is updated again by adding a token to p_6, i.e., $\mathcal{M}$=p_6. Finally, Φ=4 and $\mathcal{M}$=p_6 are output that are exactly $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$. ♦

Remark 3.4: We observe that the computational time of Function *NextUnobseValue* mainly depends on the times of the recursive calls of Function *BottomUpHandle*. Besides, it is decided from bottom up in the observation subnet N_{ω}^{β} whether Function *BottomUpHandle* needs to be called. Since N_{ω}^{β} is acyclic, the times of recursive calls of Function *BottomUpHandle* are polynomial w.r.t. the number of unobservable transitions of N_{ω}^{β} in the worst case. Hence, Function *NextUnobseValue* is of polynomial complexity w.r.t. the unobservable transitions of the considered PN. ♣

3.6 Optimal Polynomial-complexity Control Policy for a Class of PN Subject to an Admissible GMEC

This section presents an optimal and efficient control policy for ordinary PNs subject to an admissible GMEC such that the observation subnet is acyclic and BBF. First, we introduce the following function, by which the observable change w.r.t. an observed sequence is computed.

```
Ψ=NextObseValue (Ψ_{ω,δ}, t)
Input: The observable change $\Psi_{\omega,\delta}$ and $t\in T_o$;
Output: Ψ. /*The output Ψ is exactly the observable change $\Psi_{\omega,\delta'}$, where $\delta'=\delta t$.*/
1) $\Psi:=\Psi_{\omega,\delta}+\varpi(t)$;
2) Output: Ψ.
```

Proposition 3.4: Let (N, m_0) be a PN system subject to a GMEC (ω, k) and δ, $\delta'\in L_o(N, m_0)$ such that $\delta'=\delta t$. *Function NextObseValue* outputs $\Psi_{\omega,\delta'}$ when $\Psi_{\omega,\delta}$ and t are given as inputs.

Based on Functions *NextUnobseValue* and *NextObseValue*, the following Function *ComputeControlAction* computes a control action corresponding to an observed sequence δ given $\Psi_{\omega,\delta}$, $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$.

```
u=ComputeControlAction($\Psi_{\omega,\delta}$, $\Phi_{\omega,\delta}$, $\mathcal{M}_{\omega,\delta}$)
Input: The observable change $\Psi_{\omega,\delta}$, the unobservable minimal decrease $\Phi_{\omega,\delta}$, and
the basic partial marking $\mathcal{M}_{\omega,\delta}$;
Output: The control action $u=\rho(\delta)$.
1)  for each $t_c \in T_c \setminus \mathcal{T}_\omega^+$ do
2)      $u(t_c):=1$;
3)  end for
4)  for each $t_c \in T_c \cap \mathcal{T}_\omega^+$ do
5)      $(\Phi_{\omega,\delta}, \mathcal{M}_{\omega,\delta}):=NextUnobseValue(\Phi_{\omega,\delta}, \mathcal{M}_{\omega,\delta}, t_c)$;
6)      $\Psi_{\omega,\delta} := NextObseValue(\Psi_{\omega,\delta}, t_c)$;
7)      if $\Psi_{\omega,\delta} - \Phi_{\omega,\delta} > k - \omega m_0$ then
8)          $u(t_c):=0$;
9)      else
10)         $u(t_c):=1$;
11)     end if
12) end for
13) Output: $u$.
```

Finally, Policy 3.2 provides a polynomial complexity optimal control policy to enforce an admissible GMEC, which could be applied provided that the observation subnet is acyclic and BBF.

We explain Policy 3.2 in more detail. The variables Ψ, Φ, $\mathcal{M}$ in Policy 3.2 correspond to the observable change, the unobservable minimal decrease, and the basic partial marking w.r.t. the current observed sequence, respectively. The policy works as follows: First, Ψ, Φ, $\mathcal{M}$ are initialized at "0", "0", m_0^β, respectively, and the control action $u=\rho(\varepsilon)$

is computed using Function u:=*ComputeControlAction*(Ψ, Φ, $\mathcal{M}$). Then, every time the firing of a transition t is observed, a control action is computed via the following two steps:

1) Update Ψ, Φ, $\mathcal{M}$ using Functions (Φ, $\mathcal{M}$):= *NextUnobseValue* (Φ, $\mathcal{M}$, t) and then Ψ:=*NextObseValue* (Ψ, t);

2) Compute the control action $u=\rho(\delta)$ using Function u:=*ComputeControlAction*(Ψ, Φ, $\mathcal{M}$).

More intuitively, Policy 3.2 is summarized in Fig. 3.8.

Policy 3.2: An on-line control policy ρ

Plant: A PN system (N, m_0);

State specification: An admissible GMEC (ω, k).

Condition: The observation subnet N_o^β is acyclic and BBF

Initialization: Ψ:=0, Φ:=0, $\mathcal{M}$:=m_0^β.

First, the control action $u=\rho(\varepsilon)$ is determined by calling Function u:=*ComputeControlAction*(Ψ, Φ, $\mathcal{M}$).

Then, every time the firing of a transition $t\in T_o$ is observed, a control action is determined by calling the following functions one after another:

(Φ,$\mathcal{M}$):=*NextUnobseValue* (Φ,$\mathcal{M}$, t);

Ψ:=*NextObseValue* (Ψ, t); and

u:=*ComputeControlAction*(Ψ, Φ, $\mathcal{M}$).

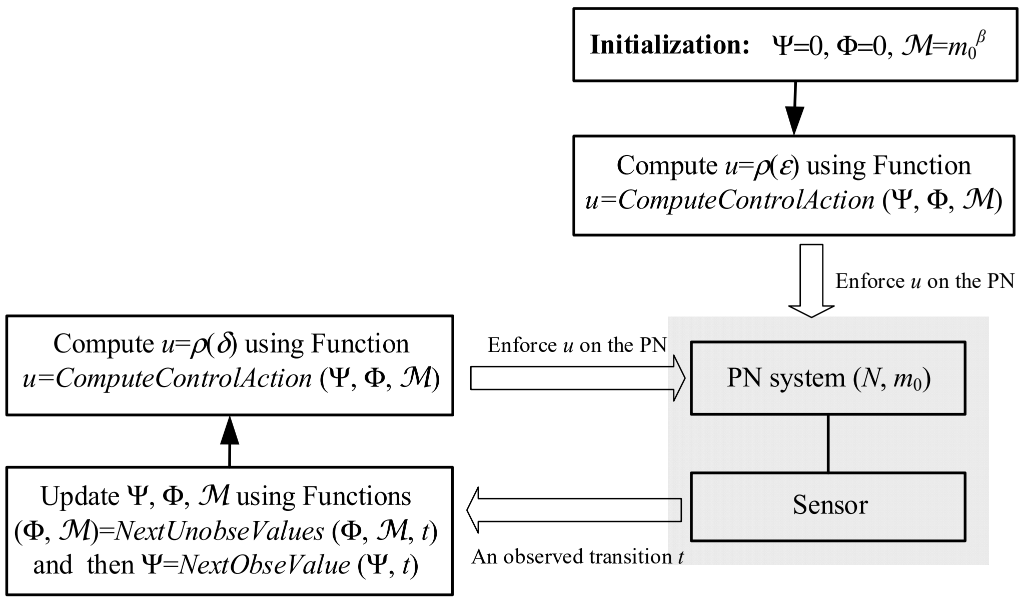

Initialization: Ψ=0, Φ=0, $\mathcal{M}=m_0^\beta$
Compute $u=\rho(\varepsilon)$ using Function $u=ComputeControlAction$ (Ψ, Φ, $\mathcal{M}$)
Enforce u on the PN
Compute $u=\rho(\delta)$ using Function $u=ComputeControlAction$ (Ψ, Φ, $\mathcal{M}$)
Enforce u on the PN
PN system (N, m_0)
Sensor
Update Ψ, Φ, $\mathcal{M}$ using Functions (Φ, $\mathcal{M}$)=$NextUnobseValues$ (Φ, $\mathcal{M}$, t) and then Ψ=$NextObseValue$ (Ψ, t)
An observed transition t

Fig. 3.8 Flow chart describing Policy 3.2

Theorem 3.6: Given a PN system (N, m_0) and an admissible GMEC (ω, k) such that the observation subnet N_ω^β is acyclic and BBF, Policy 3.2 is optimal.

Proof: See Appendix. ■

Remark 3.5: Considering that Function *NextUnobseValue* has polynomial complexity w.r.t. the number of unobservable transitions of the considered PN, it is trivial to see that Policy 3.2 is also of polynomial complexity w.r.t. the number of unobservable transitions. ♣

Recall Example 3.8: The PN in Fig. 3.6 is subject to the GMEC (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})+2m(p_{13})+2m(p_{12})\leq 8$. It can be verified that (ω, k) is an admissible GMEC and the observation subnet N_ω^β in Fig. 3.7 is acyclic and BBF. Hence, Policy 3.2 can be applied. Furthermore, it is optimal and has polynomial complexity.

3.7 Optimal Control Policy for Special Classes of PN Subject to a GMEC

In this section, we discuss how to develop an optimal policy for PNs subject to an arbitrary GMEC. We present the following theorem first.

Theorem 3.7: Given a PN system (N, m_0), two GMECs (ω_1, k_1) and (ω_2, k_2) such that $\mathcal{A}_{(\omega_1, k_1)}=\mathcal{A}_{(\omega_2, k_2)}$, and a control policy ρ, it holds that

$$\rho \text{ is optimal for } (\omega_1, k_1) \Leftrightarrow \rho \text{ is optimal for } (\omega_2, k_2).$$

Proof: Straightforward from Definition 3.4 and Theorems 3.2 and 3.3. ■

According to Theorem 3.7, given a PN system subject to an arbitrary GMEC (ω, k), if (ω, k) can be equivalently transformed into an admissible GMEC (ω', k'), the optimal control policy designed for (ω', k') is also optimal for (ω, k). (Note that an equivalent transformation from (ω, k) to (ω', k') implies $\mathcal{A}_{(\omega, k)}=\mathcal{A}_{(\omega', k')}$.) Moreover, if the observation subnet w.r.t. the admissible GMEC (ω', k') is acyclic and BBF, Policy 3.2 designed for (ω', k') is also an optimal policy for the PN system subject to (ω, k). Hence, what we need to consider now is how to equivalently transform an arbitrary GMEC into an admissible one. This problem has been extensively studied for PNs being entirely observable but partially controllable. It is worth noting that all proposed methods still work for PNs with unobservable transitions since unobservable transitions are assumed to be uncontrollable. Also, note that such a transformation is related to a subnet called *uncontrollable influence subnet* [38].

Based on the above considerations, we develop an optimal policy for a PN system (N, m_0) subject to an arbitrary GMEC (ω, k), which consists in the following two steps:

Stage 1: Compute the uncontrollable influence subnet w.r.t. (ω, k). Determine if there exists a method that can equivalently transform (ω, k) into an admissible GMEC (ω', k'). If so, compute the equivalent admissible GMEC (ω', k'); otherwise, stop.

Stage 2: Compute the observation subnet N_ω^β w.r.t. (ω', k'). Determine if N_ω^β is acyclic and BBF. If so, compute and enforce Policy 3.2 with (ω, k):=(ω', k') on the PN system; otherwise stop.

Clearly, the above method does not work for all arbitrary PNs subject to all arbitrary GMECs. The first reason is that not all GMECs can be equivalently transformed into a single admissible GMEC [103]. Fortunately, various efficient constraint transformation approaches, such as [60, 62, 100, 102, 123], have been proposed in the literature that are applicable to PNs with uncontrollable influence subnets having different structures. Therefore, in most of the practical cases of interest, depending on the given PN and the given GMEC, a different transformation method can be chosen in Stage 1. Furthermore, the observation subnet w.r.t. the computed admissible GMEC is required to be acyclic and BBF when Policy 3.2 is computed in Stage 2.

We finally notice that the computational complexity of the two stages depends on the computational complexity of Stage 1 since Policy 3.2 is of polynomial complexity. Hence, the whole method has polynomial complexity when the approach in Stage 1 also has polynomial complexity.

Example 3.9: Consider again the PN system (N, m_0) in Fig. 3.6. Now, let (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})\leq 8$ be the GMEC. We can easily verify that it is not an admissible GMEC. However, the approach in [123] can equivalently transform a GMEC into an admissible one with polynomial complexity provided that the uncontrollable influence subnet is *forward-concurrent-free* (i.e., each transition has only one input place). The uncontrollable influence subnet is reported in Fig. 3.9 and satisfies such a condition. Therefore, the approach can be applied in this case, resulting in an equivalent admissible GMEC (ω', k'): $m(p_5)+2m(p_3)+2m(p_{14})+2m(p_{13})+2m(p_{12})\leq 8$. As we analyzed in Example 3.8, the observation subnet w.r.t. (ω', k') is the net inside the dashed line in Fig. 3.7, which is acyclic and BBF. Hence, Policy 3.2 computed for the GMEC (ω', k') is exactly an optimal control policy for the PN system subject to (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})\leq 8$. Since the approach in [123] is of polynomial complexity, it is easy to see that the whole method has polynomial complexity. ♦

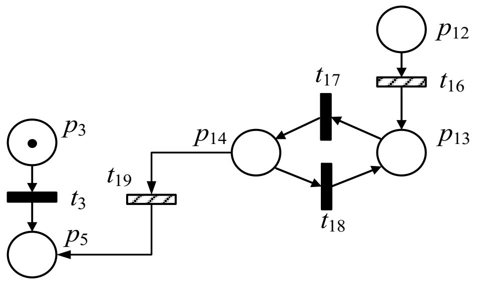


Fig. 3.9 Uncontrollable influence subnet of the PN system in Fig. 3.6 w.r.t. (ω, k): $m(p_5)+2m(p_3)+2m(p_{14})\leq 8$

3.8 Conclusions

This chapter focuses on the forbidden state problem of PNs with both uncontrollable and unobservable transitions, assuming that unobservable transitions are all uncontrollable. For ordinary PNs subject to an admissible GMEC, we propose an optimal policy with polynomial complexity, provided that a certain subnet of the original net, called the observation subnet, is acyclic and backward-conflict and backward-concurrent free. We also prove that for any two GMECs with the same admissible marking set, a control policy is optimal with respect to one GMEC if and only if it is optimal with respect to the other one. This allows us to apply the proposed approach to more general cases. Indeed, efficient transformations exist that allow us to transform an arbitrary GMEC to an equivalent admissible one.

The work of this chapter has been published in *Information Sciences*; see [124].

3.9 Appendices

3.9.1 Proof of Proposition 3.2

Since (ω, k) is an admissible GMEC, it holds that $\forall t\in T_{uc}$, $\varpi(t)\leq 0$. Thus, it follows that

$$\max_{\sigma\in\Lambda(\delta)} \varpi\cdot\vec{\sigma} = \max_{\sigma\in O_N^{-1}(\delta)} \varpi\cdot\vec{\sigma}. \quad (2)$$

It is clear that

$$\max_{\sigma\in O_N^{-1}(\delta)} \varpi\cdot\vec{\sigma} = \max_{\sigma\in O_N^{-1}(\delta)} \left(\sum\nolimits_{t\in T_o} \varpi(t)\cdot\vec{\sigma}(t) + \sum\nolimits_{t\in T_{uo}} \varpi(t)\cdot\vec{\sigma}(t)\right)$$
$$= \varpi\cdot\vec{\delta} + \max_{\sigma\in O_N^{-1}(\delta)} \sum\nolimits_{t\in T_{uo}} \varpi(t)\cdot\vec{\sigma}(t). \quad (3)$$

Since $T_{uo}\subseteq T_{uc}$, it holds that for any $\sigma\in O_N^{-1}(\delta)$, if a transition $t\in T_{uo}$ appears in σ, it is $\varpi(t)\leq 0$. Thus, it is

$$\max_{\sigma\in O_N^{-1}(\delta)} \sum\nolimits_{t\in T_{uo}} \varpi(t)\cdot\vec{\sigma}(t) = \max_{\sigma\in O_N^{-1}(\delta)} \sum\nolimits_{t\in T_{uo}\cap\mathcal{T}_\omega^-} \varpi(t)\cdot\vec{\sigma}(t)$$
$$= -\min_{\sigma\in O_N^{-1}(\delta)} \sum\nolimits_{t\in\mathcal{T}_\omega^\#} |\varpi(t)|\cdot\vec{\sigma}(t). \quad (4)$$

By (2)-(4), we have

$$\max_{\sigma\in\Lambda(\delta)} \varpi\cdot\vec{\sigma} = \varpi\cdot\vec{\delta} - \min_{\sigma\in O_N^{-1}(\delta)} \sum\nolimits_{t\in\mathcal{T}_\omega^\#} |\varpi(t)|\cdot\vec{\sigma}(t)$$
$$= \Psi_{\omega,\delta} - \Phi_{\omega,\delta}. \qquad \blacksquare$$

3.9.2 Proof of Theorem 3.5

We first introduce a new notion. Given a transition $t\in T_{uo}$, $t'\in T_{uo}$ is called a *downstream unobservable transition* w.r.t. t if t' can be reached from t following an unobservable path. The set of all downstream unobservable transitions w.r.t. $t\in T_{uo}$ is denoted by $T_{uo}^\vee(t)$. Note that transitions in $\mathcal{T}_{uo}^\beta$ can be divided into transitions inside and outside of N_ω^β. More precisely, $\mathcal{T}_{uo}^\beta = T_\omega^\beta \cup (\mathcal{T}_{uo}^\beta \cap \mathcal{I}(N_\omega^\beta))$.

Let us first consider $t\in\mathcal{T}_{uo}^\beta\cap\mathcal{I}(N_\omega^\beta)$. We observe that t cannot reach any observable transition following a path since otherwise t is a transition in N_ω^β. Hence, the firing times of t and all its downstream unobservable transitions are unrelated to the firing times of observable transitions. In other words, t and all its downstream unobservable transitions can remain unfired no matter how observable transitions fire.

Next, we consider $t \in T_o^\beta$. Since N_ω^β is BBF, t has a single output place in N_ω^β. Let $\{p\} = t^\bullet \cap P_\omega^\beta$. Moreover, p has a single input transition in N_ω^β since N_ω^β is BBF. Clearly, $\{t\} = {}^\bullet p \cap T_o^\beta$. Consider $p' \in t^\bullet \setminus \{p\}$. We can see p' cannot reach any observable transition following a path since otherwise p' is a place in N_ω^β. Hence, the firing times of unobservable transitions in $p'^\bullet$ and all of their downstream unobservable transitions are unrelated to the firing times of observable transitions. In other words, unobservable transitions in $p'^\bullet$ and all their downstream unobservable transitions can remain unfired no matter how observable transitions fire. Consider $t' \in {}^\bullet p \setminus \{t\}$. We can see t' is definitely an observable transition since otherwise t' is a transition in N_ω^β. Based on the above analysis, when a transition sequence δ is observed, t fires its least times only in the case that transitions in $p^\bullet \cap T_{uo}$ all fire their least times and transitions in $(t^\bullet \setminus \{p\})^\bullet$ all remain unfired.

Due to that N_ω^β is acyclic, we observe that, from bottom up in $\overline{N_\omega^\beta}$, it holds that $\forall t \in \mathcal{T}_{uo}^\beta$, $\exists \sigma \in O_N^{-1}(\delta)$ such that $\vec{\sigma}(t) = \varphi_\delta(t)$ and σ satisfies $\vec{\sigma}(t') = \varphi_\delta(t')$, $\forall t' \in T_{uo}^\vee(t)$. Trivially, $\exists \sigma \in O_N^{-1}(\delta)$ such that $\vec{\sigma}(t) = \varphi_\delta(t)$, $\forall t \in \mathcal{T}_{uo}^\beta$. By Property 3.2, $|\mathcal{B}_v^\beta(\delta)| = 1$. Thus, it also holds that $|\mathcal{B}_m^\beta(\delta)| = 1$ by Definition 3.13. ■

3.9.3 Proof of Proposition 3.3

We can see that Φ and $\mathcal{M}$ are initialized as $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$. Then, we have the following four cases related to t.

Case 1: $t \notin \mathcal{T}_o^\beta$. The firing of t does not require the firing of any transition in $\mathcal{T}_{uo}^\beta$ in advance. Hence, $\varphi_{\delta'}(t) = \varphi_\delta(t)$, $\forall t \in \mathcal{T}_{uo}^\beta$. Trivially, $\mathcal{M}_{\omega,\delta'} = \mathcal{M}_{\omega,\delta}$. Besides, $\Phi_{\omega,\delta'} = \Phi_{\omega,\delta}$ by Property 3.4 since $\mathcal{T}_\omega^\# \subseteq \mathcal{T}_{uo}^\beta$. Hence, the outputted Φ and $\mathcal{M}$ are exactly $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$.

Case 2: $t \in \mathcal{T}_o^\beta{}_{(out)} \setminus \mathcal{T}_o^\beta{}_{(in)}$. In this case, *BottomUpHandle*(t) is called. The execution of *BottomUpHandle*(t) is essentially a procedure that determining from bottom up which transitions in $N_\omega{}^\beta$ definitely fire more times than their least firing times induced by δ in the case that the firing of t is observed following δ. Such a determination relies on the basic partial marking $\mathcal{M}_{\omega,\delta}$. According to Definition 3.13, we can see that the firing of σ s.t. $\vec{\sigma}^\beta\in\mathcal{B}_v{}^\beta(\delta)$ leads to the distribution of tokens in $N_\omega{}^\beta$ being $\mathcal{M}_{\omega,\delta}$. In other words, when all transitions in $\mathcal{T}_{uo}{}^\beta$ fire their least firing times in the case that δ is observed, the distribution of tokens in $N_\omega{}^\beta$ is $\mathcal{M}_{\omega,\delta}$. Let p be a place in ${}^\bullet t\cap P_\omega{}^\beta$ and t' be the single input unobservable transition of p. There are two cases related to p. Subcase a): p contains tokens at $\mathcal{M}_{\omega,\delta}$. It implies that t' fires $\varphi_\delta(t')$ times that are enough to enable t regardless of the distribution of tokens in other input places of t. Hence, t' does not need to fire more than $\varphi_\delta(t')$ times when the firing of t is observed following δ. In this case, $\mathcal{M}$ needs to be updated by removing a token in p since t fires. Subcase b): p contains no token at $\mathcal{M}_{\omega,\delta}$. Now that the firing of t is observed following δ, t' at least fires one more time than $\varphi_\delta(t')$ times since otherwise t cannot fire. Hence, if $t'\in\mathcal{T}_\omega{}^\#$, Φ needs to be updated by adding $|\varpi(t')|$ to itself. Let p' be a place in ${}^\bullet t'\cap P_\omega{}^\beta$ and t'' be the single input unobservable transition of p'. In the case that t' at least fires one more time than $\varphi_\delta(t')$ times, it can be determined similarly whether t'' at least fires one more time than $\varphi_\delta(t'')$ times. Accordingly, Φ and $\mathcal{M}$ are updated. The determination is performed from bottom up similarly and Φ and $\mathcal{M}$ are updated similarly. Consequently, the outputted Φ and $\mathcal{M}$ are exactly $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$.

Case 3: $t\in\mathcal{T}_o{}^\beta{}_{(in)}\backslash\mathcal{T}_o{}^\beta{}_{(out)}$. The firing of t does not require the firing of any transition in $\mathcal{T}_{uo}{}^\beta$ in advance. Hence, $\varphi_{\delta'}(t)=\varphi_\delta(t)$, $\forall t\in\mathcal{T}_{uo}{}^\beta$. As a result, $\Phi_{\omega,\delta'}=\Phi_{\omega,\delta}$. We can see that the firing of t increases tokens in its output places in $N_\omega{}^\beta$. Hence, $\mathcal{M}$ is updated by adding one token to each output place of t in $N_\omega{}^\beta$. Therefore, the outputted Φ and $\mathcal{M}$ are exactly $\Phi_{\omega,\delta'}$ and $\mathcal{M}_{\omega,\delta'}$.

Case 4: $t\in \mathcal{T}_o^{\beta}(in)\cap\mathcal{T}_o^{\beta}(out)$. Based on the analysis in Cases 2 and 3, it can be concluded that the outputted Φ and $\mathcal{M}$ are exactly $\Phi_{\omega,\delta}$ and $\mathcal{M}_{\omega,\delta}$. ■

3.9.4 Proof of Theorem 3.6

Before proving it, we introduce the following lemma.

Lemma 3.1: Let (N, m_0) be a PN system subject to an admissible GMEC (ω, k), and δ_1, $\delta_2\in L_o(N, m_0)$. $\Psi_{\omega,\delta_2}-\Phi_{\omega,\delta_2}\leq\Psi_{\omega,\delta_1}-\Phi_{\omega,\delta_1}$ if $\exists\delta\in T_{uc}^{*}$ such that $\delta_2=\delta_1\delta$.

Proof: We can see that $\Lambda(\delta_2)\subseteq\Lambda(\delta_1)$ and thus $\max_{\sigma\in\Lambda(\delta_2)}\varpi\cdot\vec{\sigma}\leq\max_{\sigma\in\Lambda(\delta_1)}\varpi\cdot\vec{\sigma}$. Since (ω, k) is admissible, $\Psi_{\omega,\delta_2}-\Phi_{\omega,\delta_2}\leq\Psi_{\omega,\delta_1}-\Phi_{\omega,\delta_1}$ by Proposition 3.2. ■

In the following, we prove Theorem 3.6.

We use ρ_1 and ρ_2 to denote Policies 3.1 and 3.2 respectively. First, we prove $L_o(N, m_0)|_{\rho_2}\sqsubseteq L_o(N, m_0)|_{\rho_1}$. Let $\delta\in L_o(N, m_0)|_{\rho_2}$. Consider $\delta=\varepsilon$. It is assumed that $\varepsilon\in\mathcal{G}_{(\omega, k)}$, i.e., $\forall\sigma\in\Lambda(\varepsilon)$, $\omega\cdot m_\sigma\leq k$. Thus, $\Psi_{\omega,\varepsilon}-\Phi_{\omega,\varepsilon}\leq k-\omega\cdot m_0$. Consider $\delta=t_1t_2\ldots t_n\in L_o(N, m_0)|_{\rho_2}$. Clearly, $\forall\delta'\in\bar{\delta}$, $\delta'\in L_o(N, m_0)|_{\rho_2}$. Since $t_1\in L_o(N, m_0)|_{\rho_2}$, we have the following cases related to t_1 according to Policy 3.2.

Case 1: $t_1\in T_{uc}$. We can see $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq\Psi_{\omega,\varepsilon}-\Phi_{\omega,\varepsilon}$ by Lemma 3.1. Since $\Psi_{\omega,\varepsilon}-\Phi_{\omega,\varepsilon}\leq k-\omega\cdot m_0$, we have $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq k-\omega\cdot m_0$.

Case 2: $t_1\in T_c\backslash\mathcal{T}_\omega^{+}$. Since $\varpi(t_1)\leq 0$, $\Psi_{\omega,t_1}\leq\Psi_{\omega,\varepsilon}$. Besides, it is trivial to see that $\Phi_{\omega,t_1}\geq\Phi_{\omega,\varepsilon}$. Hence, $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq\Psi_{\omega,\varepsilon}-\Phi_{\omega,\varepsilon}$. Furthermore, we have $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq k-\omega\cdot m_0$.

Case 3: $t_1\in T_c\cap\mathcal{T}_\omega^{+}$ and $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq k-\omega\cdot m_0$.

Clearly, it always holds $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq k-\omega\cdot m_0$. Now, consider $t_1t_2\in L_o(N, m_0)|_{\rho_2}$. There are three similar cases related to t_2. By the similar analysis, we have $\Psi_{\omega,t_1t_2}-\Phi_{\omega,t_1t_2}\leq k-\omega\cdot m_0$ since $\Psi_{\omega,t_1}-\Phi_{\omega,t_1}\leq k-\omega\cdot m_0$. By repeating the above analysis, it can be concluded that $\Psi_{\omega,\delta}-\Phi_{\omega,\delta}\leq k-\omega\cdot m_0$. Furthermore, we can see $\forall\delta'\in\bar{\delta}$, $\Psi_{\omega,\delta'}-\Phi_{\omega,\delta'}\leq k-\omega\cdot m_0$. Observing Policy 3.1, it is $\delta\in L_o(N, m_0)|_{\rho_1}$. Hence, $L_o(N, m_0)|_{\rho_2}\sqsubseteq L_o(N, m_0)|_{\rho_1}$.

Next, we prove $L_o(N, m_0)|_{\rho_2}\supseteq L_o(N, m_0)|_{\rho_1}$. By contradiction, suppose that $L_o(N, m_0)|_{\rho_1}\not\subset L_o(N, m_0)|_{\rho_2}$. Hence, there exists $\delta\in L_o(N, m_0)|_{\rho_1}\backslash L_o(N, m_0)|_{\rho_2}$. Since $\delta\in L_o(N,$

$m_0)|_{\rho1}$, it is clear that $\delta\in L_o(N, m_0)$. Moreover, since $\delta\notin L_o(N, m_0)|_{\rho2}$, we can see $\exists\delta' t\in \bar{\delta}$ such that $\delta'\in L_o(N, m_0)|_{\rho2}$, $t\in T_c\cap\mathcal{T}_\omega^+$ and $\Psi_{\omega,\delta' t}-\Phi_{\omega,\delta' t}>k-\omega\cdot m_0$. Observing Policy 3.1, we have $\delta' t\notin L_o(N, m_0)|_{\rho1}$. Thus, $\delta\notin L_o(N, m_0)|_{\rho1}$, which contradicts the fact that $\delta\in L_o(N, m_0)|_{\rho1}$. Hence, $L_o(N, m_0)|_{\rho2}\supseteq L_o(N, m_0)|_{\rho1}$. Consequently, $L_o(N, m_0)|_{\rho2}=L_o(N, m_0)|_{\rho1}$. According to Theorem 3.2, Policy 3.2 is optimal since Policy 3.1 is optimal. ■

CHAPTER IV

Forbidden State Problem of DES Vulnerable to Network Attacks

4.1 Introduction

In this chapter, we address the forbidden state problem of DES assuming that events are all controllable and observable but the system is vulnerable to network attacks. We consider the so-called *sensor-reading disguising attacks* (*SD-attacks* for short) that may disguise the occurrence of an event as another by tampering with the sensor-readings in sensor communication channels. In particular, we use PNs as a reference formalism to model a plant that is allowed to be bounded or unbounded and assume a control specification in terms of a GMEC. We propose three different methods to derive on-line control policies. The first two lead to an optimal policy but are computationally inefficient when applied to large-size systems. On the contrary, the third method computes a policy with timely response even for large-size systems but at the expense of optimality.

This chapter is organized as follows. Section 4.2 introduces the considered attacks and formalizes the problem statement. Section 4.3 presents a method that computes an optimal control policy based on the enumeration of markings consistent with the current observation. In Section 4.4, we develop two control policies based on constructing a *monitor-controlled* PN system. One is an optimal policy that still requires the marking enumeration and the other is computationally more efficient even for large-size systems but at the expense of optimality. Section 4.5 draws conclusions of this chapter.

We notice that the work of this chapter is now under review by *IEEE Transactions on Automatic Control*.

4.2 Problem Statement

In this section, we first introduce the type of attacks considered in this chapter and then formalize the problem statement. We note that in the next chapter we consider the same type of attacks as in this chapter but deal with a different control problem.

4.2.1 Sensor-reading Disguising Attacks (SD-attacks)

We consider a type of attacks that we call *sensor-reading disguising attacks* (*SD-attacks* for short). In an SD-attack scenario, the intruder has the ability to disguise the occurrence of an event as the occurrence of another event by modifying sensor-readings in vulnerable sensor communication channels. Formally, we define an SD-attack as $\mathcal{A} \in 2^{(T\times T)\setminus\{(t,t)|t\in T\}}$, i.e., as a set of ordered transition-pairs excluding those with identical transitions. Each pair in $\mathcal{A}$, e.g., (t, t'), denotes the possible action taken by the intruder disguising the firing of transition t as the firing of transition t'. In other words, $\mathcal{A}$ characterizes the capability of an intruder on disguising transitions.

We assume that we have prior knowledge of $\mathcal{A}$, i.e., it is a given set in our problem statement. How to derive $\mathcal{A}$ is application-dependent. It may be derived based on analyzing the vulnerability of sensor channels, the potential goal of an intruder, the similarity of sensor-readings, and so on. The focus of our work is enforcing a control specification to the considered system when given such a potential intruder. We notice that a similar assumption is made in [97], where the authors assume that they do not know exactly which intruder they are facing but they have prior knowledge of all the possible intruders, namely, multiple SD-attacks are given. In addition, we note that $\mathcal{A}$ is assumed to be not empty since the case that $\mathcal{A}$ is empty implies that no attack threat exists, which is not the concern of our work.

In what follows, we introduce notation related to SD-attacks.

Definition 4.1: Given an SD-attack $\mathcal{A}$, the set of *vulnerable transitions* is $T_v=\{t \mid (t, t')\in \mathcal{A}\}$ and the set of *uncertain transitions* is $T_u=\{t' \mid (t, t')\in \mathcal{A}\}$. Moreover, given a transition t, we denote

$$A(t)= \{t\}\cup\{t' \mid (t, t')\in \mathcal{A}\} \text{ and } A^{-1}(t)= \{t\}\cup\{t' \mid (t', t)\in \mathcal{A}\}.$$

The set $A(t)$ enumerates all those transitions that could appear firing when transition t actually has fired, while $A^{-1}(t)$ is the set of transitions that may have actually fired when transition t is observed. We can see that when a vulnerable transition t fires, its observation is not necessarily transition t; when the firing of an uncertain transition t is observed, the transition that has actually fired is not necessarily transition t.

Example 4.1: Consider a PN with $T=\{t_1\text{-}t_5\}$ and an SD-attack $\mathcal{A}=\{(t_1, t_2), (t_2, t_3), (t_2, t_4), (t_3, t_4)\}$. It is $T_v=\{t_1, t_2, t_3\}$ and $T_u=\{t_2, t_3, t_4\}$. It holds: $A(t_1)=\{t_1, t_2\}$, $A(t_2)=\{t_2, t_3, t_4\}$, $A(t_3)=\{t_3, t_4\}$, $A(t_4)=\{t_4\}$, $A(t_5)=\{t_5\}$. Moreover, $A^{-1}(t_1)=\{t_1\}$, $A^{-1}(t_2)=\{t_1, t_2\}$, $A^{-1}(t_3)=\{t_2, t_3\}$, $A^{-1}(t_4)=\{t_2, t_3, t_4\}$, and $A^{-1}(t_5)=\{t_5\}$. ♦

Furthermore, we extend the notations $A(\cdot)$ and $A^{-1}(\cdot)$ to a transition sequence, respectively. The extension of $A(\cdot)$ to a transition sequence $\sigma\in T^*$ is defined recursively such that

1) $A(\varepsilon)=\varepsilon$; and
2) $A(\sigma)=A(\sigma')A(t)$, where $\sigma'\in T^*$ and $t\in T$ s.t. $\sigma=\sigma't$.

Note that $A(\sigma')A(t)$ denotes the concatenation of $A(\sigma')$ and $A(t)$. The extension of $A^{-1}(\cdot)$ to a transition sequence is similarly defined.

We can see that $A(\sigma)$ is the set of *possible observations* when the sequence $\sigma\in T^*$ actually fires, whereas $A^{-1}(\sigma)$ is the set of sequences that may produce the observation $\sigma\in T^*$. In general, not all sequences in $A^{-1}(\sigma)$ are enabled at the initial marking of the considered system.

In the following, we introduce the *observation language* of a PN system under an SD-attack and then reformulate a control policy in the presence of such an attack.

Definition 4.2: Let (N, m_0) be a PN system vulnerable to an SD-attack $\mathcal{A}$. The *observation language* of (N, m_0) under the attack $\mathcal{A}$ is

$$L_o^{\mathcal{A}}(N, m_0)=\bigcup_{\sigma\in L(N,m_0)} A(\sigma).$$

Example 4.2: Consider the PN system (N, m_0) in Fig. 4.1 vulnerable to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$ and the sequence $t_2t_3\in L(N, m_0)$. It is $A(t_2t_3)=A(t_2)A(t_3)=\{t_2t_3, t_3t_3\}$. Moreover, consider the sequence $t_2t_3t_4\in L_o^{\mathcal{A}}(N, m_0)$. It is $A^{-1}(t_2t_3t_4)= A^{-1}(t_2)A^{-1}(t_3)A^{-1}(t_4)=\{t_2t_3t_4, t_2t_2t_4\}$. Note that when the sequence $t_2t_3t_4$ is observed, we know for sure that $t_2t_3t_4$ has fired since $t_2t_2t_4$ is not enabled at the initial marking. ♦

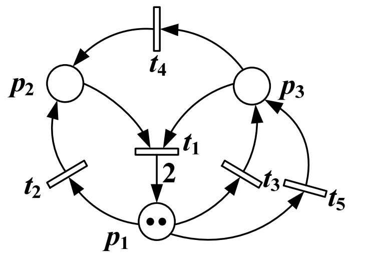


Fig. 4.1 PN system (N, m_0) in Examples 4.2-4.6

As discussed in Section 2.4.2, a control policy (or supervisor) of a PN system (N, m_0) is a function ρ: $L_o(N, m_0)\to 2^{T_c}$. Clearly, it is reduced as ρ: $L(N, m_0)\to 2^T$ in the case that all transitions are observable and controllable. Now, although all transitions are observable and controllable, considering the existence of an SD-attack $\mathcal{A}$, the observed language of the system should be $L_o^{\mathcal{A}}(N, m_0)$ rather than $L(N, m_0)$. Thus, a control policy (or supervisor) of a PN system (N, m_0) in the presence of an SD-attack $\mathcal{A}$ is

$$\rho: L_o^{\mathcal{A}}(N, m_0)\to 2^T.$$

4.2.2 Problem Statement

In this chapter we study the following control problem under the assumption that all transitions are controllable and observable.

***Problem* 4.1**: Given a PN system (N, m_0) vulnerable to an SD-attack $\mathcal{A}$ and a GMEC (ω, k), design a control policy ρ: $L_o^{\mathcal{A}}(N, m_0)\rightarrow 2^T$ that enforces GMEC (ω, k).

The control policy to be designed in Problem 4.1 is actually a tolerant control policy w.r.t. the considered attack. The "tolerant control problem" is often discussed in the scenario where a system suffers from faults, referring to the design of a control method that enables the system to continue operating properly even in the case that some faults occur. Likewise, in the context of attack issues, a control policy (or supervisor) tolerant to an attack indicates that the control policy enforces a control specification to the considered system even in the presence of the attack.

Like many studies dealing with the supervisory control problem, we assume that the initial marking m_0 is legal, i.e., $\omega\cdot m_0\leq k$.

In this chapter, initially, Problem 4.1 is investigated with the goal of determining an optimal control policy. Then, such a requirement is relaxed, in order to provide a more efficient approach in terms of computational complexity.

4.3 Optimal Control Policy

In this section, we present an on-line control policy for Problem 4.1. To this end, we introduce the notion of *violating transitions* at a given set of markings. Recall that $En(m)$ denotes the set of transitions enabled at marking m. Now, we extend the notation to a marking set $\mathcal{M}$ such that $En(\mathcal{M})=\bigcup_{m\in\mathcal{M}}En(m)$. Besides, in this chapter, given a marking m and a transition $t\in En(m)$, we denote m_t the marking reached by firing t at m; given a transition sequence α enabled at marking m, we denote m_α the marking reached by firing α at m. Note that the notation is defined differently from that in Chapter III.

Definition 4.3: Given a PN N, a GMEC (ω, k) and a set of markings $\mathcal{M}$, the set of *violating transitions* at $\mathcal{M}$ is

$$\Gamma_{(\omega,k)}(\mathcal{M})=\{t\in En(\mathcal{M}) \mid \exists m\in\mathcal{M}, \text{ s.t. } \omega\cdot m_t > k\}.$$

In words, $\Gamma_{(\omega,k)}(\mathcal{M})$ includes all those transitions whose firing leads a marking in $\mathcal{M}$ to a marking violating the GMEC (ω, k).

Now, we compute an on-line control policy by Method 4.1, where δ denotes the current observation and $\mathcal{M}$ contains all possible markings consistent with δ. When nothing is observed, δ is initialized at ε that denotes the empty sequence and $\mathcal{M}$ is initialized at $\{m_0\}$. Accordingly, the disabled set $\rho(\varepsilon)$ is computed and enforced on the plant, which is exactly the set of violating transitions at $\{m_0\}$, namely, $\Gamma_{(\omega,k)}(\{m_0\})$. Then, every time a new transition t is observed, we update the set $\mathcal{M}$ of markings consistent with the observation and compute the new set $\Gamma_{(\omega,k)}(\mathcal{M})$ of violating transitions. In particular, the updating of $\mathcal{M}$ is done in Step 4, where the new set $\mathcal{M}$ contains the markings reached from a marking in the old set $\mathcal{M}$ by firing a transition that possibly produces the observation t and is not a disabled transition.

Method 4.1: On-line computation of an optimal policy

Input: 1) plant: a PN system (N, m_0) vulnerable to an SD-attack $\mathcal{A}$; and
2) specification: a GMEC (ω, k).

Output: A control policy ρ.

```
1. Initialization: $\delta\leftarrow\varepsilon$; $\mathcal{M}\leftarrow\{m_0\}$;
2. $\rho(\delta)\leftarrow\Gamma_{(\omega,k)}(\mathcal{M})$;
3. while there is an observation $t\in T$ do
4.     $\mathcal{M}\leftarrow\{m_{t'} \mid t'$ is enabled at $m\in\mathcal{M}, t'\in A^{-1}(t)\setminus\rho(\delta)\}$;
5.     $\delta\leftarrow\delta t$;
6.     $\rho(\delta)\leftarrow\Gamma_{(\omega,k)}(\mathcal{M})$;
7. end while
```

Example 4.3: Consider the PN system (N, m_0) in Fig. 4.1 with specification (ω, k): $m(p_2)\leq 1$, and assume that the system is vulnerable to an SD-attack $\mathcal{A}=\{(t_2, t_3)\}$. Method 4.1 computes the control policy as follows.

1) Nothing is observed, i.e., $\delta=\varepsilon$. It is $\mathcal{M}=\{m_0\}=\{[2, 0, 0]^T\}$ and thus $\rho(\varepsilon)=\Gamma_{(\omega, k)}(\mathcal{M})=\emptyset$;

2) Suppose that t_3 is observed. $\mathcal{M}$ is updated to $\{[1, 1, 0]^T, [1, 0, 1]^T\}$ since $A^{-1}(t_3)=\{t_2, t_3\}$ and $\rho(\varepsilon)=\emptyset$. Accordingly, it is $\rho(t_3)=\Gamma_{(\omega, k)}(\mathcal{M})=\{t_2\}$;

3) Suppose that t_3 is observed again. It is $A^{-1}(t_3)\backslash\rho(t_3)=\{t_3\}$. Hence, $\mathcal{M}$ is updated to $\{[0, 1, 1]^T, [0, 0, 2]^T\}$ and $\rho(t_3t_3)=\Gamma_{(\omega k)}(\mathcal{M})=\{t_4\}$.

Similarly, every time a new transition is observed, the disabled set is recomputed.

Theorem 4.1: Let (N, m_0) be a PN system vulnerable to an SD-attack $\mathcal{A}$ and (ω, k) be the imposed GMEC. The policy computed by Method 4.1 is optimal.

Proof: Let ρ be the policy computed by Method 4.1. It is trivial to see that ρ is acceptable, i.e., $R(N, m_0)|_{\rho}\subseteq \mathcal{L}_{(\omega, k)}$. Let ρ' be a policy more permissive than ρ. It means that there exists an observed sequence $\delta\in T^*$ generated by the plant such that 1) $\rho'(\delta')=\rho(\delta')$, $\forall\delta'\in \bar{\delta}\backslash\{\delta\}$; and 2) $\rho'(\delta)\subset\rho(\delta)$. Let $\mathcal{M}$ be the set of markings consistent with the observation obtained before the last transition in δ is observed. Clearly, it is $\mathcal{M}\subseteq\mathcal{L}_{(\omega, k)}$. Since $\rho'(\delta)\subset\rho(\delta)$, there exists a violating transition associated with $\mathcal{M}$ that is not included in $\rho'(\delta)$. Hence, the system possibly reaches an illegal marking by firing such a transition. In other words, $R(N, m_0)|_{\rho'}\not\subset \mathcal{L}_{(\omega, k)}$. This proves that ρ is optimal. ♦

Remark 4.1: Method 4.1 computes a control action every time the firing of a transition is observed. Thus, as long as the considered net system may keep running, the computation of control actions is always needed and will not terminate. This is why we describe the proposed control policy as "method" rather than "algorithm" since an algorithm should be able to terminate. Besides, we notice that Method 4.1 is not limited to GMECs. It is actually applicable to any control specification that requires a system to evolve within a marking set. ♣

Remark 4.2: We analyze the computational complexity of Method 4.1. In Method 4.1, we record a set $\mathcal{M}$ of markings consistent with an observation. At the beginning, it contains the initial marking only. When we observe the first transition, we update the set $\mathcal{M}$. Since, in the worst case, all the transitions in the net may have fired, the updated set $\mathcal{M}$ contains at most $|T|$ markings. Then, we compute $\Gamma_{(\omega\ k)}(\mathcal{M})$ that is the disabled set relative to the current observation. By its definition, we should check for each marking in $\mathcal{M}$ and each enabled transition. Thus, the complexity of computing such a control action is $O(|T|^2)$. When we observe the second transition, we update again the set $\mathcal{M}$. Similarly, considering the worst case, the updated set $\mathcal{M}$ contains at most $|T|^2$ markings. Then, we compute again $\Gamma_{(\omega\ k)}(\mathcal{M})$ and its complexity is $O(|T|^3)$. By repeating the above reasoning, when we observe the n^{th} transition, the updated set $\mathcal{M}$ contains at most $|T|^n$ markings and the complexity of computing a control action is $O(|T|^{n+1})$. We notice that the set $\mathcal{M}$ is always bounded although its size grows exponentially with the length of the observed sequence. Nevertheless, since the size of $\mathcal{M}$ grows exponentially, the computation of a control action might become inefficient particularly in large-size systems. Consequently, we look for control policies with a smaller computational complexity. ♣

4.4 Control Policies Based on a Monitor-controlled PN System

In this section, we develop two control policies based on the off-line construction of a *monitor-controlled PN system*, which is an auxiliary supervisory system allowing us to establish, more efficiently than Method 4.1, which transitions should be disabled corresponding to an observation.

4.4.1 Monitor-controlled PN System

In this subsection, we introduce a *monitor-controlled PN system* and define its related notations.

Given a plant (N, m_0) and a GMEC (ω, k), the *monitor-controlled PN system*, denoted as (N^c, m_0^c), is an augmented system obtained by adding, to the net system to be controlled, monitor p_c that enforces GMEC (ω, k) via the *place-invariant method* [109]. In more detail, it is

$$[N^c]=\begin{bmatrix}[N]\\ -\omega\cdot[N]\end{bmatrix}=\begin{bmatrix}[N]\\ -\varpi\end{bmatrix} \quad \text{and} \quad m_0^c=\begin{bmatrix}m_0\\ k-\omega\cdot m_0\end{bmatrix},$$

where [N^c] and [N] are the incidence matrices of N^c and N, respectively. Note that the last rows of [N^c] and m_0^c correspond to monitor p_c.

Example 4.4: Consider the PN system (N, m_0) in Fig. 4.1 and the GMEC (ω, k): $m(p_2)\leq 1$. The corresponding monitor-controlled PN system (N^c, m_0^c) is shown in Fig. 4.2. ♦

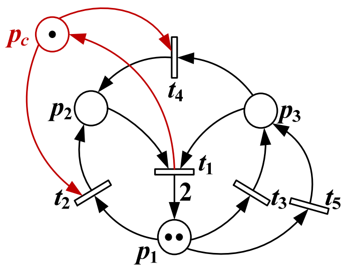


Fig. 4.2 Monitor-controlled PN system (N^c, m_0^c) relative to the PN system in Fig. 4.1 and the GMEC (ω, k): $m(p_2)\leq 1$

In the following, we use m^c to indicate the marking of the monitor-controlled PN N^c. By default, m^c is divided into two parts, that is,

$$m^c=[m^{\mathrm{T}} \quad m_{pc}{}^{\mathrm{T}}]^{\mathrm{T}},$$

where m denotes the marking corresponding to the original PN N and m_{pc} denotes the marking of the monitor place p_c.

We now introduce some notation to be used in the rest of the chapter.

Definition 4.4: Given a monitor-controlled PN N^c=(P^c, T^c, F^c, W^c) and a marking m_{pc}

of the monitor place p_c, we define the set of *monitor-disabled transitions* as

$$Dis(m_{pc})=\{t\in p_c^{\bullet} \mid m_{pc}<W^c(p_c, t)\}.$$

Definition 4.5: Given a monitor-controlled PN $N^c=(P^c, T^c, F^c, W^c)$ and a marking $m^c=[m^T \quad m_{pc}^T]^T$, we define:

$$\Psi(m^c)=Dis(m_{pc})\cap En(m).$$

In other words, $\Psi(m^c)$ is the set of transitions that are enabled in the original net system but are disabled by monitor p_c. Furthermore, given a set of markings $\mathcal{M}^c$ of N^c, we define:

$$\Psi(\mathcal{M}^c)=\bigcup_{m^c\in\mathcal{M}^c}\Psi(m^c).$$

Example 4.5: Consider the monitor-controlled PN N^c in Fig. 4.3 at marking $m^c=[1, 1, 0, 0]^T$. It is $m=[1, 1, 0]^T$ and $m_{pc}=0$. Hence, $Dis(m_{pc})=\{t_2, t_4\}$ and $\Psi(m^c)=\{t_2\}$ by Definitions 4.4 and 4.5. ♦

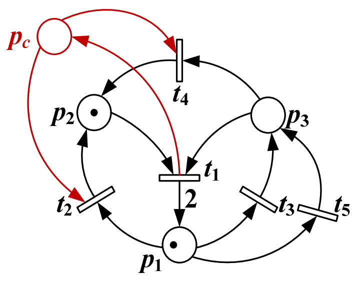


Fig. 4.3 Monitor-controlled PN N^c at marking $m^c=[1, 1, 0, 1]^T$

4.4.2 Optimal Control Policy

In this subsection, we propose the second approach, that we call Method 4.2, to derive a control policy. The procedure of Method 4.2 is similar to Method 4.1. The main difference is that Method 4.2 constructs a monitor-controlled PN system (N^c, m_0^c) off-line and the policy is computed on-line based on (N^c, m_0^c). Every time a transition is observed, Method 4.2 computes the set $\mathcal{M}^c$ of all possible reachable markings of $(N^c$,

m_0^c) and determines the disabled set based on $\mathcal{M}^c$. Actually, the policy computed by Method 4.2 is the same as that computed by Method 4.1. It means that Method 4.2 also derives an optimal solution, which is formally presented and proven as follows.

```
Method 4.2: Computation of an optimal policy based on monitors
Input: 1) plant: a PN system $(N, m_0)$ vulnerable to an SD-attack $\mathcal{A}$; and
       2) specification: a GMEC $(\omega, k)$.
Output: A control policy $\rho$.
Off-line computation:
1. Compute the monitor-controlled PN system $(N^c, m_0^c)$ with respect to the
   net $(N, m_0)$ and the GMEC $(\omega, k)$.
On-line computation and control:
1. Initialization: $\delta \leftarrow \varepsilon$; $\mathcal{M}^c \leftarrow \{m_0^c\}$;
2. $\rho(\delta) \leftarrow \Psi(\mathcal{M}^c)$ ;
3. while there is an observation $t \in T$ do
4.     $\mathcal{M}^c \leftarrow \{m^c_{t'} \mid t'$ is enabled at $m^c \in \mathcal{M}^c$, $t' \in A^{-1}(t) \setminus \rho(\delta)\}$;
5.     $\delta \leftarrow \delta t$;
6.     $\rho(\delta) \leftarrow \Psi(\mathcal{M}^c)$;
7. end while
```

The following results hold.

Property 4.1 [109]: Let (N, m_0) be a PN system, (ω, k) be a GMEC, and (N^c, m_0^c) be the corresponding monitor-controlled PN system. Given a marking $m^c \in R(N^c, m_0^c)$, it holds that

1) $\omega \cdot m + m_{pc} = k$; and

2) $Dis(m_{pc}) = \{t \in T \mid \omega \cdot m + \varpi(t) > k\}$.

Property 4.2: Let (N, m_0) be a PN system, (ω, k) be a GMEC, and (N^c, m_0^c) be the corresponding monitor-controlled PN system.

1) Given a marking $m^c \in R(N^c, m_0^c)$, it holds that $\Psi(m^c) = \{t \in En(m) \mid \omega \cdot m_t > k\}$;

2) Given a set of markings $\mathcal{M}^c \subseteq R(N^c, m_0^c)$, it holds that $\Psi(\mathcal{M}^c)=\Gamma_{(\omega, k)}(\mathcal{M})$, where $\mathcal{M}$ is the set of markings derived by restricting markings in $\mathcal{M}^c$ to the net N.

Proof: Straightforward from Property 4.1 and Definitions 4.3 and 4.5. ■

Theorem 4.2: Let (N, m_0) be a PN system vulnerable to an SD-attack $\mathcal{A}$ and (ω, k) be a GMEC. The policy computed by Method 4.2 is optimal.

Proof: Let ρ_2 and ρ_1 be policies computed by Methods 4.2 and 4.1, respectively. It is trivial to see that $\rho_2=\rho_1$ by Property 4.2. As a result, ρ_2 is optimal since ρ_1 is optimal. ■

Example 4.6: Consider again the PN system (N, m_0) in Fig. 4.1 subject to GMEC (ω, k): $m(p_2)\leq 1$ and vulnerable to an SD-attack $\mathcal{A}=\{(t_2, t_3)\}$. Now, we use Method 4.2 to compute a control policy. First, a monitor-controlled PN system (N^c, m_0^c) is constructed off-line, as shown in Fig. 4.2. Next, the on-line computation and control is performed based on (N^c, m_0^c). We show a part of the procedure as follows.

1) Nothing is observed, i.e., $\delta=\varepsilon$. It is $\mathcal{M}^c=\{m_0^c\}=\{[2, 0, 0, 1]^T\}$ and thus $\rho(\varepsilon)=\Psi(\mathcal{M}^c)=\varnothing$;

2) Suppose that t_3 is observed. Since $A^{-1}(t_3)=\{t_2, t_3\}$ and $\rho(\varepsilon)=\varnothing$, either t_2 or t_3 may have actually fired, resulting in $\mathcal{M}^c=\{m_1^c, m_2^c\}=\{[1, 1, 0, 0]^T, [1, 0, 1, 1]^T\}$. For the sake of clarity, markings m_1^c and m_2^c are both shown in Fig. 4.4(a) in blue and black color, respectively. We can see that $\rho(t_3)=\Psi(\mathcal{M}^c) =\Psi(m_1^c)\cup\Psi(m_2^c)=\{t_2\}\cup\varnothing=\{t_2\}$;

3) Suppose that t_3 is observed again. It is $A^{-1}(t_3)\setminus\rho(t_3)=\{t_3\}$, i.e., t_3 has fired for sure. Hence, $\mathcal{M}^c$ is updated at $\mathcal{M}^c=\{m_3^c, m_4^c\}=\{[0, 1, 1, 0]^T, [0, 0, 2, 1]^T\}$, which is shown in Fig. 4.4(b) with m_3^c in blue and m_4^c in black. We can see that $\rho(t_3t_3)=\Psi(\mathcal{M}^c)=\Psi(m_3^c)\cup\Psi(m_4^c)=\{t_4\}\cup\varnothing=\{t_4\}$. ♦

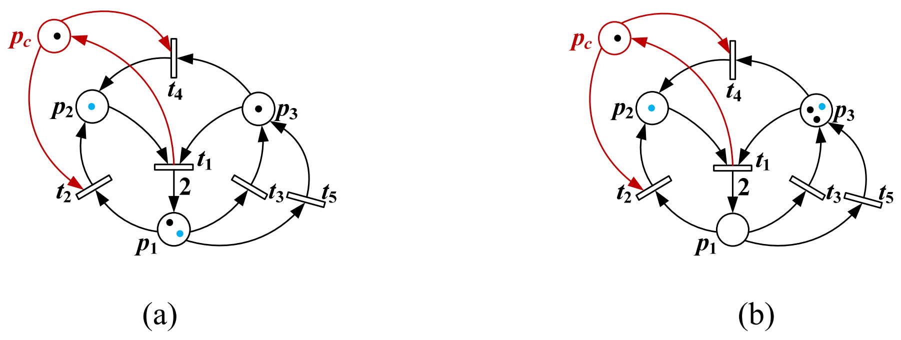


Fig. 4.4 (a) Marking set $\mathcal{M}^c=\{m_1^c, m_2^c\}$ consistent with $\delta=t_3$; and (b) marking set $\mathcal{M}^c=\{m_3^c, m_4^c\}$ consistent with $\delta=t_3t_3$.

Remark 4.3: Method 4.2 provides an alternative way to compute an optimal control policy. It consists of both off-line and on-line computations. Regarding the off-line computations, the complexity is proportional to the number of GMECs. The on-line computations are similar to Method 4.1. The differences lie in that when implementing Method 4.2 we record the set $\mathcal{M}^c$ of markings of the monitor-controlled system consistent with an observation and based on that, we compute the disabling set $\Psi(\mathcal{M}^c)$. Thus, as in Method 4.1, when we observe the n^{th} transition, the updated set $\mathcal{M}^c$ contains no more than $|T|^n$ markings. Then, the complexity of computing $\Psi(\mathcal{M}^c)$ is $O(|T|^n)$. Clearly, compared with Method 4.1, computing a control action using Method 4.2 is faster. However, the applicability of Method 4.2 could still become prohibitive for long evolutions of the system. ♣

4.4.3 Non-optimal Control Policy with Improved Computational Efficiency

In this subsection, we present a more efficient method (Method 4.3) to compute a control policy, which is still based on constructing the monitor-controlled PN system. The derived policy is acceptable but not necessarily optimal.

Method 4.3: Computation of a non-optimal policy based on monitors

Input: 1) plant: a PN system (N, m_0) vulnerable to an SD-attack $\mathcal{A}$; and

2) specification: a GMEC (ω, k).

Output: A control policy ρ.

```
Off-line computation:
1. Compute the monitor-controlled PN system ($N^c$, $m_0^c$) with respect to the
   net ($N$, $m_0$) and the GMEC ($\omega$, $k$).
On-line computation and control:
1. Initialization: $\delta\leftarrow\varepsilon$; $m^c\leftarrow m_0^c$; Flag←True;
2. $\rho(\delta)\leftarrow Dis(m_{pc})$;
3. while there is an observation $t\in T$ do
4.     if Flag=True then
5.         $T_{real}\leftarrow A^{-1}(t)\cap En(m^c)$;
6.         if $T_{real}$ is a singleton and $t'\in T_{real}$ then
7.             $m^c\leftarrow m^c+[N^c](\cdot, t')$;
8.         else
9.             $m_{pc}\leftarrow m_{pc}+\min_{t'\in T_{real}}[N^c](p_c,t')$;
10.            Flag←False;
11.        end if
12.    else
13.        $T_{real}\leftarrow A^{-1}(t)\backslash\rho(\delta)$;
14.        $m_{pc}\leftarrow m_{pc}+\min_{t'\in T_{real}}[N^c](p_c,t')$;
15.    end if
16.    $\delta\leftarrow\delta t$;
17.    $\rho(\delta)\leftarrow Dis(m_{pc})$;
18. end while
```

We explain Method 4.3 as follows. As in Method 4.2, we construct off-line the monitor-controlled PN system (N^c, m_0^c) and then perform the on-line computation and control based on it when the plant starts its evolution.

We introduce a variable Flag$\in$ {True, False}. In particular, Flag=True indicates that we know with certainty the marking of the considered system before making a new observation, while Flag=False means the opposite, i.e., we do not know with certainty at which marking the system is when a new observation is done. Hence, Flag is initialized at "True" since we know the system is at marking m_0^c when we start our observation. Besides, m^c is initialized at m_0^c, representing the system marking and the disabled set $\rho(\varepsilon)$ enforced on the plant is the set of monitor-disabled transitions computed based on the initial token-count of p_c. Whenever a new observation is produced, the token-count of monitor p_c is updated and the disabled set enforced on the plant is updated accordingly based on the token-count of p_c. In what follows, we focus on the way the token-count of p_c is updated when a new observation is produced. In particular, two different cases are considered:

Case 1: We know with certainty in which marking the system is before such an observation (i.e., Flag=True); and

Case 2: We do not know with certainty in which marking the system is before such an observation (i.e., Flag=False).

For the former case, we compute the set of transitions that may have fired (stored in T_{real}) based on the known current marking and the observed transition as in Step 5. Clearly, if T_{real} contains only one transition, the new marking can be uniquely determined including the token-count in p_c. Otherwise, as it typically occurs, we do not know the new marking of the system. In this case, Method 4.3 does not enumerate possible new markings of the system but only updates the token-count in p_c considering the firing effect of each transition in T_{real} on p_c. More precisely, the token-count in p_c is updated considering the firing of a transition in T_{real} such that p_c contains the fewest tokens. In such a case, Flag is updated to "False" and remains like that until the end of the system observation.

When Flag=False, i.e., the current marking of the system is not known, we do not reconstruct the marking of the system but update the token-count in p_c only. In this case, we compute the set of transitions that may have fired only based on the observed transition and the enforced disabled set as in Step 13. Then, we update again the token-count in p_c considering the firing of a transition in T_{real} such that p_c contains the fewest tokens.

Theorem 4.3: Let (N, m_0) be a PN system vulnerable to an SD-attack $\mathcal{A}$ and (ω, k) be a GMEC. The policy computed by Method 4.3 is acceptable.

Proof: Let (N^c, m_0^c) be the monitor-controlled PN system w.r.t. (N, m_0) and (ω, k). By Property 4.1, $\forall m^c \in R(N^c, m_0^c)$, $\omega \cdot m + m_{pc} = k$ and $Dis(m_{pc}) = \{t \in T \mid \omega \cdot m + \varpi(t) > k\}$.

Let ρ be the policy computed by Method 4.3. By assumption, m_0 is legal. Let m_{pc0} be the marking of monitor p_c such that $m_0^c = [m_0, m_{pc0}]$. Since $\rho(\varepsilon) = Dis(m_{pc0})$, any firable transition t at m_0 satisfies the inequality: $\omega \cdot m_0 + \varpi(t) \le k$, i.e., any marking reachable from m_0 by firing a transition is legal.

Let t_1 be the first observed transition. Suppose that t_1' is the really fired transition producing observation t_1. Let m_1 and m_1^c be the markings reached by N and N^c after firing t_1' from m_0 and m_0^c, respectively. It is $m_1^c = [m_1, m_{pc1}]$, where $m_{pc1} = k - \omega \cdot m_1$ and $m_{pc1} = m_{pc0} + [N^c](p_c, t_1')$. Let $m_{pc1}^* = m_{pc0} + \min_{t' \in T_{real}} [N^c](p_c, t')$, where $T_{real} = A^{-1}(t_1) \cap En(m_0^c)$ since Flag=True. We can see that $m_{pc1}^* \le m_{pc1}$ since $t_1' \in T_{real}$. Hence, $Dis(m_{pc1}^*) \supseteq Dis(m_{pc1})$. We observe that $\omega \cdot m_1 \le k$ and $Dis(m_{pc1}) = \{t \in T \mid \omega \cdot m_1 + \varpi(t) > k\}$. Since $\rho(t_1) = Dis(m_{pc1}^*)$, any firable transition t at m_1 satisfies the condition: $\omega \cdot m_1 + \varpi(t) \le k$, i.e., any marking reachable from m_1 by firing a transition is legal.

If Flag is still "True", when the next transition is observed, by repeating the above reasoning, the reachable markings in the next step are all legal. Let us consider now the case that Flag is changed to "False". Let t_2 be the next observed transition. Suppose that t_2' is the really fired transition producing observation t_2. Let m_2 and m_2^c be the markings reached by N and N^c after firing t_2' from m_1 and m_1^c, respectively. Clearly, $m_2^c = [m_2, m_{pc2}]$, where $m_{pc2} = k - \omega \cdot m_2$ and $m_{pc2} = m_{pc1} + [N^c](p_c, t_2')$. Let $m_{pc2}^* = m_{pc1}^* + \min_{t' \in T_{real}} [N^c](p_c, t')$, where $T_{real} = A^{-1}(t_2) \setminus \rho(t_1)$ since Flag=False. Since $m_{pc1}^* \le m_{pc1}$ and $t_2' \in T_{real}$, it holds that $m_{pc2}^* \le m_{pc2}$. Hence, $Dis(m_{pc2}^*) \supseteq Dis(m_{pc2})$. We observe that $\omega \cdot m_2 \le k$ and $Dis(m_{pc2}) = \{t \in T \mid \omega \cdot m_2 + \varpi(t) > k\}$. Since $\rho(t_1 t_2) = Dis(m_{pc2}^*)$, any firable transition t at m_2 satisfies the condition: $\omega \cdot m_2 + \varpi(t) \le k$, i.e., any marking reachable from m_1 by firing a transition is legal.

By repeating the above reasoning, every time we observe a transition, the computed disabled set guarantees that the reachable markings in the next step are all legal. As a result, $R(N, m_0)|_\rho \subseteq \mathcal{L}_{(\omega, k)}$, i.e., the policy ρ is acceptable. ■

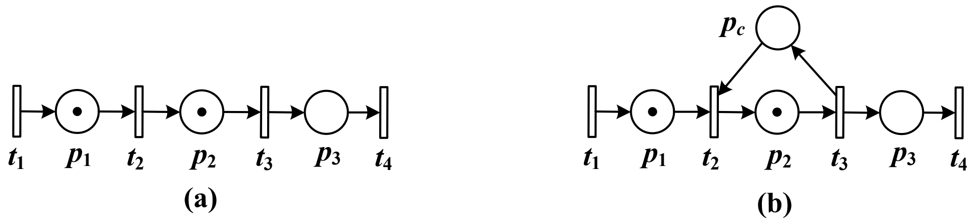


Fig. 4.5 (a) Plant: a PN system (N, m_0); and
(b) Monitor-controlled PN system (N^c, m_0^c) associated with GMEC (ω, k): $m(p_2)\le 1$

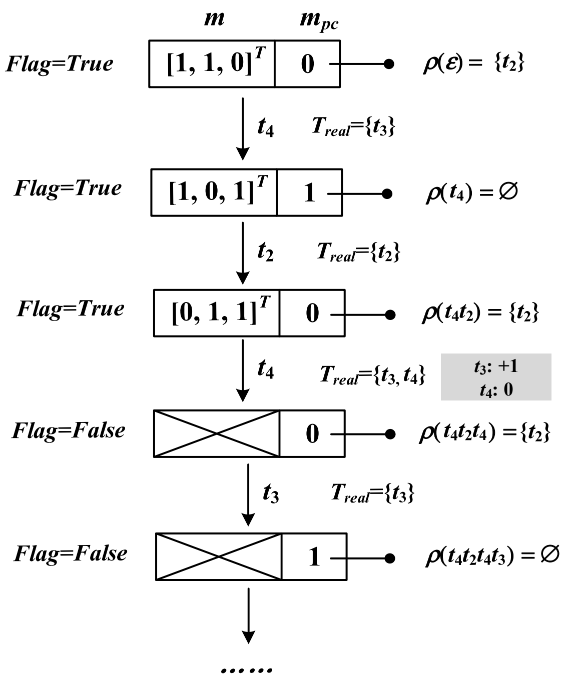


Fig. 4.6 Illustration of the on-line control in Example 4.7

Example 4.7: Consider the PN system (N, m_0) in Fig. 4.5(a) subject to a GMEC (ω, k): $m(p_2)\le 1$ and vulnerable to an SD-attack $\mathcal{A}=\{(t_3, t_4)\}$. Let us see how Method 4.3 works. First, the monitor-controlled PN system (N^c, m_0^c) is constructed off-line, as shown in Fig. 4.5(b). Next, the on-line computation and control is performed based on (N^c, m_0^c). We explain a part of the procedure as follows, which is also shown in Fig. 4.6 for the sake of clarity.

1) Nothing is observed, i.e., $\delta=\varepsilon$. We initialize Flag=True and $m^c=m_0^c=[1, 1, 0, 0]^T$. Note that the last entry of m^c indicates the token-count of p_c. Thus, $\rho(\varepsilon)=Dis(m_{pc})=\{t_2\}$.

2) Suppose that t_4 is observed. Since Flag=True, it is $T_{real}=A^{-1}(t_4)\cap En(m^c)=\{t_3\}$. Hence, m^c is updated by firing t_3 instead of t_4, resulting in $m^c=[1, 0, 1, 1]^T$. Hence, $\rho(t_4)=Dis(m_{pc})=\varnothing$.

3) Suppose that t_2 is observed. Since Flag=True, $T_{real}=A^{-1}(t_2)\cap En(m^c)=\{t_2\}$. Hence, m^c is updated at $m^c=[0, 1, 1, 0]^T$ by firing t_2. Thus, $\rho(t_4t_2)=Dis(m_{pc})=\{t_2\}$.

4) Suppose that t_4 is observed. Since Flag is still "True", it is $T_{real}=A^{-1}(t_4)\cap En(m^c)=\{t_3, t_4\}$. Now, $|T_{real}|=2$. Thus, we only update the token-count of p_c. Since $[N^c](p_c, t_3)=1$ and $[N^c](p_c, t_4)=0$, it follows that $m_{pc}=m_{pc}+\min\{[N^c](p_c, t_3), [N^c](p_c, t_4)\}=0$. Hence, $\rho(t_4t_2t_4)=Dis(m_{pc})=\{t_2\}$. Finally, Flag is updated to False.

5) Suppose that t_3 is observed. Since Flag=False, it is $T_{real}=A^{-1}(t_3)\backslash\rho(t_4t_2t_4)=\{t_3\}$. Hence, $m_{pc}= m_{pc}+[N^c](p_c, t_3)=1$. Accordingly, $\rho(t_4t_2t_4t_3)=Dis(m_{pc})=\varnothing$. ♦

Example 4.8: Consider again the problem in Example 4.6. Now, we use Method 4.3 to compute a control policy. The monitor-controlled PN system (N^c, m_0^c) is still the system in Fig. 4.2. Let us see a part of the on-line computation procedure.

1) Nothing is observed, i.e., $\delta=\varepsilon$. It is Flag=True and $m^c=m_0^c=[2, 0, 0, 1]^T$. Thus, $\rho(\varepsilon)=Dis(m_{pc})=\varnothing$;

2) Suppose that t_3 is observed. Since Flag=True, it is $T_{real}=A^{-1}(t_3)\cap En(m^c)=\{t_2, t_3\}$. Hence, we update m_{pc} such that $m_{pc}=m_{pc}+\min\{[N^c](p_c, t_2), [N^c](p_c, t_3)\}=1+\min\{-1, 0\}=0$. Then, $\rho(t_3)=Dis(m_{pc})=\{t_2, t_4\}$ and Flag is updated to False.

3) Suppose that t_3 is observed again. Since Flag=False, $T_{real}= A^{-1}(t_3)\backslash\rho(t_3)=\{t_3\}$. Hence, m_{pc} is updated at $m_{pc}= m_{pc}+[N^c](p_c, t_3)=0$. Accordingly, $\rho(t_3t_3)=Dis(m_{pc})=\{t_2, t_4\}$.

We observe that compared with the policy computed by Method 4.2, which is shown in Example 4.6, the policy computed by Method 4.3 is more restrictive. ♦

Remark 4.4: The non-optimality of Method 4.3 can be explained as follows. Method 4.3 does not enumerate the set of markings in which the system can be when such a set is not a singleton. Indeed, in such a case, it only updates the token-count in monitor p_c

to make it restrictive enough to prevent the system from entering an illegal marking. Since the enumeration of markings is avoided, the computation is much more efficient. However, it results in a control policy not necessarily optimal. In particular, the set T_{real} of possible really fired transitions computed at Flag=False may include non-enabled transitions since we do not have the information on the marking of the system. As a result, the set of transitions disabled by the computed monitor p_c can be larger than it needs to be, which turns out a more restrictive control policy. Such an effect could increase when the evolution of the system proceeds. ♣

Remark 4.5: As Method 4.2, the complexity of the off-line computations in Method 4.3 is proportional to the number of GMECs. Regarding the on-line computations of Method 4.3, the complexity of computing a control action is always $O(|T|)$ since we always record one marking only and every time we observe a transition, no more than $|T|$ transitions should be enumerated. As a result, Method 4.3 is also applicable to large-size nets. ♣

In what follows, we show that the optimality of the policy computed by Method 4.3 is guaranteed when a certain condition on the SD-attack is satisfied. The condition can be interpreted as that the variation of the token-count in the monitor p_c is identical for all transitions that could have fired corresponding to an observed transition under the SD-attack.

Theorem 4.4: Let (N, m_0) be a PN system vulnerable to an SD-attack $\mathcal{A}$ and (ω, k) be a GMEC. The policy computed by Method 4.3 is optimal if $\forall t\in T$, it holds that $\overline{\omega}(t_1)=\overline{\omega}(t_2)=\ldots=\overline{\omega}(t_n)$, where $\{t_1, t_2, \ldots, t_n\}=A^{-1}(t)$.

Proof: Let ρ_3 and ρ_2 be the policies computed by Methods 4.3 and 4.2, respectively. We prove that ρ_3 is as permissive as ρ_2, i.e., $L(N, m_0)|_{\rho_3}=L(N, m_0)|_{\rho_2}$.

We preliminarily introduce a new notation. Given a policy ρ and $\delta\in L_o(N, m_0)$, we denote $Next_\rho(\delta)=\{t\in T\,|\,\sigma t\in L(N, m_0)|_\rho$, where $\sigma\in A^{-1}(\delta)\}$, i.e., the set of transitions that are firable in the next step under the control policy ρ after observing δ.

First, consider $\delta=\varepsilon$. It is clear that $\rho_2(\varepsilon)=\Psi(m_0^c)=Dis(m_{pc0})\cap En(m_0)$, while $\rho_3(\varepsilon)=Dis(m_{pc0})$, where m_{pc0} is the marking of monitor p_c s.t. $m_0^c=[m_0\ m_{pc0}]^T$. Note that

$Next_{\rho2}(\varepsilon)=En(m_0)\backslash\rho_2(\varepsilon)$ and $Next_{\rho3}(\varepsilon)=En(m_0)\backslash\rho_3(\varepsilon)$. Hence, it obviously holds that $Next_{\rho2}(\varepsilon)=Next_{\rho3}(\varepsilon)$.

Next, let t_1 be the first observed transition, i.e., $\delta=t_1$. Consider Method 4.2. Let $\mathcal{M}^c$ be the set of possible reached markings of (N^c, m_0^c) consistent with t_1 under control policy ρ_2. We observe that, $\forall m^c\in\mathcal{M}^c$, $m_{pc}=m_{pc0}-\varpi(t_1')$, where $t_1'\in A^{-1}(t_1)$. Since $\varpi(t_{11})=\varpi(t_{12})=\ldots=\varpi(t_{1n})$, where $\{t_{11}, t_{12}, \ldots, t_{1n}\}=A^{-1}(t_1)$, all the markings in $\mathcal{M}^c$ have the identical token-count in monitor p_c. Let a be such a number. It holds that $\Psi(\mathcal{M}^c)=\bigcup_{m^c\in\mathcal{M}^c}\Psi(m^c)=\bigcup_{m^c\in\mathcal{M}^c}(Dis(m_{pc})\cap En(m))$. As a result, $\rho_2(t_1)=\Psi(\mathcal{M}^c)=En(\mathcal{M})\cap Dis(a)$, where $\mathcal{M}$ is the set of markings by restricting markings in $\mathcal{M}^c$ to the net N, which is exactly the set of possible reached markings of (N, m_0) consistent with t_1 under control policy ρ_2. Hence, $Next_{\rho2}(t_1)=En(\mathcal{M})\backslash\rho_2(t_1)=En(\mathcal{M})\backslash Dis(a)$. Consider Method 4.3. $\rho_3(t_1)=Dis(m_{pc}^*)$, where $m_{pc}^*=m_{pc0}+\min_{t'\in T_{real}}(-\varpi(t'))$ and $T_{real}\subseteq A^{-1}(t_1)$. Hence, $m_{pc}^*=a$. Since $Next_{\rho2}(\varepsilon)=Next_{\rho3}(\varepsilon)$, the set of possible reached markings of (N, m_0) consistent with t_1 under the control of ρ_3 is also $\mathcal{M}$. Hence, $Next_{\rho3}(t_1)=En(\mathcal{M})\backslash\rho_3(t_1)=En(\mathcal{M})\backslash Dis(a)$. Clearly, $Next_{\rho2}(t_1)=Next_{\rho3}(t_1)$.

By repeating the same procedure, we can see $Next_{\rho2}(\delta)=Next_{\rho3}(\delta)$, $\forall\delta\in L_o(N, m_0)$. This implies that $L(N, m_0)|_{\rho3}=L(N, m_0)|_{\rho2}$. Thus we conclude that, since ρ_2 is optimal, ρ_3 is also optimal. ■

If the condition in Theorem 4.4 is satisfied, Method 4.3 can be simplified by reducing Steps 4-15 to the following two steps:

1. wait for an observation $t\in T$ from the plant;
2. $m_{pc}\leftarrow m_{pc}+[N^c](p_c,t)$; /*Equivalently, $m_{pc}\leftarrow m_{pc}-\varpi(t)$;*/

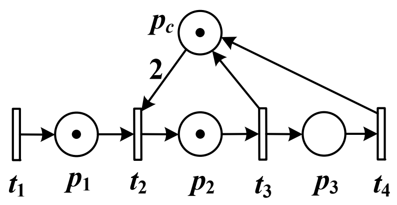


Fig. 4.7 Monitor-controlled PN system $(N^{c\prime}, m_0^{c\prime})$ associated with the plant (N, m_0) in Fig. 4.5 (a) and the GMEC (ω', k'): $2m(p_2)+m(p_3)\leq 3$

Remark 4.6: Inspired by Theorem 4.4, we have another idea to compute a control policy with timely response. That is, we may transform the given GMEC into a more restrictive GMEC that satisfies the condition in Theorem 4.4 and then perform the simplified Method 4.3 to compute a control policy. Clearly, the optimality is sacrificed to enhance the computational efficiency. For instance, GMEC (ω, k): $m(p_2)\leq 1$ in Example 4.7 can be transformed into a more restrictive GMEC (ω', k'): $2m(p_2)+m(p_3)\leq 3$ such that $\varpi'(t_3)=\varpi'(t_4)=-1$. The corresponding monitor-controlled PN system $(N^{c\prime}, m_0^{c\prime})$ is shown in Fig. 4.7. We thus can compute an on-line efficient control policy by the simplified Method 4.3, which is optimal for GMEC (ω', k'): $2m(p_2)+m(p_3)\leq 3$ but only acceptable for the original GMEC (ω, k): $m(p_2)\leq 1$. Normally, we want to get a transformed GMEC as permissive as possible. How to do systematically such GMEC transformation is left as future work. ♣

We finally note that the proposed methods can be easily extended to a control specification that is a conjunction of multiple GMECs. Suppose that a set of GMECs is $W=\{(\omega_1, k_1), (\omega_2, k_2), \ldots, (\omega_i, k_i)\}$. Let us see how the proposed methods are modified to handle the specification $\wedge W$.

Consider Method 4.1. In this case, we only need to extend the set of violating transitions $\Gamma_{(\omega, k)}(\mathcal{M})$ to W such that

$$\Gamma_W(\mathcal{M})=\{t\in En(\mathcal{M}) \mid \exists m\in \mathcal{M}, \exists(\omega, k)\in W, \text{ s.t. } \omega m_t > k\}.$$

Consider Method 4.2. The monitor-controlled PN system (N^c, m_0^c) is now an augmented system by adding monitors $p_{c1}, p_{c2}, \ldots, p_{ci}$ to (N, m_0), each monitor enforcing a GMEC in W by the place-invariant method. Specifically, it is

$$[N^c]=[[N],\quad -\varpi_1,\quad -\varpi_2,\quad \ldots,\quad -\varpi_i]^{\mathrm{T}};$$

$$m_0^c=[m_0,\quad k-\omega_1\cdot m_0,\quad k-\omega_2\cdot m_0,\quad \ldots,\quad k-\omega_i\cdot m_0]^{\mathrm{T}},$$

where the last i rows of $[N^c]$ and m_0^c correspond to monitors p_{c1}, p_{c2}, …, and p_{ci}, respectively. We use P_c to denote the set of monitors, i.e., $P_c=\{p_{c1}, p_{c2}, \ldots, p_{ci}\}$. Thus, given a marking m^c of the monitor-controlled PN N^c, it can be divided into two parts, that is, $m^c=[m \quad m_{Pc}]^T$, where m and m_{Pc} denote markings by restricting m^c within N and P_c, respectively. In addition, the set of monitor-disabled transitions is extended to marking m_{Pc} of P_c such that $Dis(m_{Pc})=\bigcup_{pc\in Pc} Dis(m_{pc})$. Accordingly, given a marking m^c of a monitor-controlled PN N^c, $\Psi(m^c)$ is re-defined as $\Psi(m^c)=Dis(m_{Pc})\cap En(m)$.

It is trivial to see that Methods 4.1 and 4.2, appropriately modified as mentioned above, both result in an optimal control policy when handling a conjunction of GMECs.

Consider Method 4.3. In addition to computing a monitor-controlled PN system (N^c, m_0^c) w.r.t. the conjunction of GMECs $\wedge W$, Steps 9 and 14 are replaced by the step:

$$\forall p_c\in P_c,\ m_{pc}\leftarrow m_{pc}+\min_{t'\in T_{real}}[N^c](p_c,t')$$

and Steps 2 and 17 are replaced by $\rho(\delta)\leftarrow Dis(m_{Pc})$. In this way, the modified Method 4.3 leads to an acceptable control policy with timely response. Furthermore, it is optimal if $\forall t\in T$, $\forall(\omega, k)\in W$, $\varpi(t_1)=\varpi(t_2)=\ldots=\varpi(t_n)$, where $\{t_1, t_2, \ldots, t_n\}=A^{-1}(t)$.

4.5 Study Case

We consider a tourist attraction consisting of four areas A-D, as shown in Fig. 4.8. The entrance and exit of the tourist attraction are located in area A and there are several one-way gates between areas. A PN system modelling the flow of visitors in the tourist attraction is depicted in Fig. 4.9. In more detail, places p_1-p_4 model areas A-D, respectively. Each transition models the transit of one visitor in the corresponding gate, which is physically detected by a sensor installed on the gates. Moreover, each token models one visitor. Initially, the PN system is in a state where each area contains one visitor.

Suppose that there is a restriction on the number of visitors in area C (modeled by place p_3) due to safety constraints. Assume that a malicious attacker wants to interfere with the control system with the goal of compromising its safety. Here, we consider a practical scenario in which the control center communicates with sensors/actuators related to all the gates via a communication network. In terms of PNs, this means that a control policy to be designed works by observing the firing of transitions and controlling transitions according to the current observation. Now, suppose that the controller knows that the communication channel related to sensors installed with gate g_{BC} is vulnerable to attacks and its sensor signals are prone to be disguised as the sensor signals produced by gate g_{BD}. In this case, when designing a control policy, we need to take into account the SD-attack $\mathcal{A}=\{(t_3, t_4)\}$.

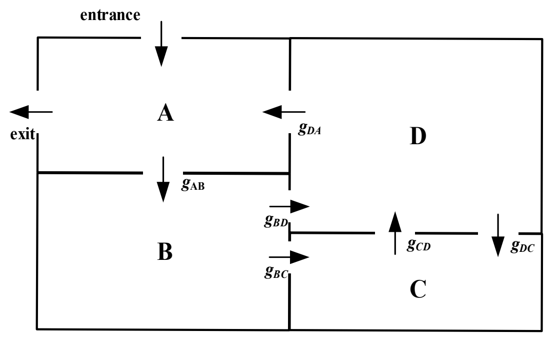


Fig. 4.8 Sketch map of a tourist attraction

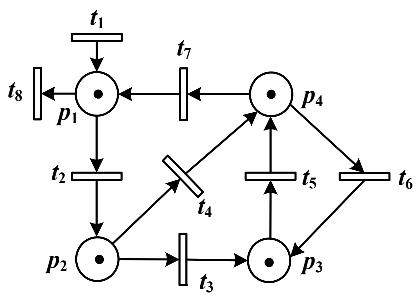


Fig. 4.9 PN model of the tourist attraction in Fig. 4.8 vulnerable to the SD-attack $\mathcal{A}=\{(t_3, t_4)\}$

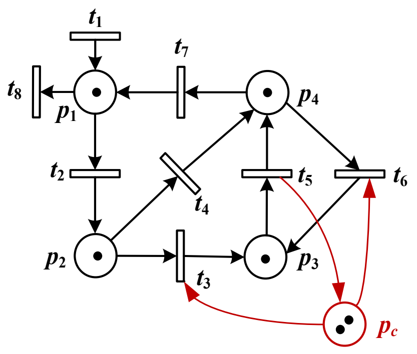


Fig. 4.10 Monitor-controlled PN system relative to the PN system in Fig. 4.9 and the GMEC (ω, k): $m(p_3)\leq 3$

In what follows, we apply Methods 4.1-4.3 to control the system. We assume that the number of visitors in area C cannot be more than three, i.e., we enforce the GMEC (ω, k): $m(p_3)\leq 3$. The monitor-controlled PN system relative to the GMEC (ω, k): $m(p_3)\leq 3$ is shown in Fig. 4.10. Table 4.1 illustrates the application of Methods 4.1-4.3 for a possible system evolution. In more detail, the first column shows the observed transitions, the second column records the set $\mathcal{M}$ of markings consistent with the current observation, which is computed in Method 4.1, the third column records the set $\mathcal{M}^c$ of markings of the monitor-controlled system consistent with the current observation, which is computed in Method 4.2, and the forth column provides the disabled set computed by Methods 4.1 and 4.2. The last three columns refer to Method 4.3 and contain the marking, Flag, and the disabled set computed in Method 4.3, respectively. Note that, for sake of clarity, the number of tokens in the monitor place p_c are highlighted in bold in the table. Besides, we write "×" to indicate that we do not record the token count in the corresponding place.

For this example, it can be verified that the condition of Theorem 4.4 is not satisfied. In more detail, we observe that $A^{-1}(t_4)=\{t_3, t_4\}$ but $\varpi(t_3)\neq\varpi(t_4)$ since $\varpi(t_3)=1$ and $\varpi(t_4)=0$. Thus, the policy computed by Method 4.3 is not guaranteed to be optimal. Indeed, from Table 4.1, we can see that the policy computed by Method 4.3 is more restrictive than those computed by Methods 4.1 and 4.2. Nevertheless, we note that Method 4.3 records one marking only while Methods 4.1 and 4.2 both record multiple markings.

Table 4.1 Application of Methods 4.1-4.3

	Method 4.1&4.2			Method 4.3		
t_i	$\mathcal{M}$	$\mathcal{M}^c$	$\rho(\delta)$	m^c	Flag	$\rho(\delta)$
ε	$[1, 1, 1, 1]^T$	$[1, 1, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[1, 1, 1, 1, \mathbf{2}]^T$	True	$\varnothing$
t_1	$[2, 1, 1, 1]^T$	$[2, 1, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[2, 1, 1, 1, \mathbf{2}]^T$	True	$\varnothing$
t_1	$[3, 1, 1, 1]^T$	$[3, 1, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[3, 1, 1, 1, \mathbf{2}]^T$	True	$\varnothing$
t_2	$[2, 2, 1, 1]^T$	$[2, 2, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[2, 2, 1, 1, \mathbf{2}]^T$	True	$\varnothing$
t_4	$[2, 1, 2, 1]^T$; $[2, 1, 1, 2]^T$	$[2, 1, 2, 1, \mathbf{1}]^T$; $[2, 1, 1, 2, \mathbf{2}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_7	$[3, 1, 2, 0]^T$; $[3, 1, 1, 1]^T$	$[3, 1, 2, 0, \mathbf{1}]^T$; $[3, 1, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_2	$[2, 2, 2, 0]^T$; $[2, 2, 1, 1]^T$	$[2, 2, 2, 0, \mathbf{1}]^T$; $[2, 2, 1, 1, \mathbf{2}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_4	$[2, 1, 3, 0]^T$; $[2, 1, 2, 1]^T$; $[2, 1, 1, 2]^T$	$[2, 1, 3, 0, \mathbf{0}]^T$; $[2, 1, 2, 1, \mathbf{1}]^T$; $[2, 1, 1, 2, \mathbf{2}]^T$	$\{t_3\}$	$[\times,\times,\times,\times, \mathbf{0}]^T$	False	$\{t_3, t_6\}$
t_4	$[2, 0, 3, 1]^T$; $[2, 0, 2, 2]^T$; $[2, 0, 1, 3]^T$	$[2, 0, 3, 1, \mathbf{0}]^T$; $[2, 0, 2, 2, \mathbf{1}]^T$; $[2, 0, 1, 3, \mathbf{2}]^T$	$\{t_6\}$	$[\times,\times,\times,\times, \mathbf{0}]^T$	False	$\{t_3, t_6\}$
t_2	$[1, 1, 3, 1]^T$; $[1, 1, 2, 2]^T$; $[1, 1, 1, 3]^T$	$[1, 1, 3, 1, \mathbf{0}]^T$; $[1, 1, 2, 2, \mathbf{1}]^T$; $[1, 1, 1, 3, \mathbf{2}]^T$	$\{t_3, t_6\}$	$[\times,\times,\times,\times, \mathbf{0}]^T$	False	$\{t_3, t_6\}$
t_5	$[1, 1, 2, 2]^T$; $[1, 1, 1, 3]^T$; $[1, 1, 0, 4]^T$	$[1, 1, 2, 2, \mathbf{1}]^T$; $[1, 1, 1, 3, \mathbf{2}]^T$; $[1, 1, 0, 4, \mathbf{3}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_2	$[0, 2, 2, 2]^T$; $[0, 2, 1, 3]^T$; $[0, 2, 0, 4]^T$	$[0, 2, 2, 2, \mathbf{1}]^T$; $[0, 2, 1, 3, \mathbf{2}]^T$; $[0, 2, 0, 4, \mathbf{3}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_7	$[1, 2, 2, 1]^T$; $[1, 2, 1, 2]^T$; $[1, 2, 0, 3]^T$	$[1, 2, 2, 1, \mathbf{1}]^T$; $[1, 2, 1, 2, \mathbf{2}]^T$; $[1, 2, 0, 3, \mathbf{3}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
t_6	$[1, 2, 3, 0]^T$; $[1, 2, 2, 1]^T$; $[1, 2, 1, 2]^T$	$[1, 2, 3, 0, \mathbf{0}]^T$; $[1, 2, 2, 1, \mathbf{1}]^T$; $[1, 2, 1, 2, \mathbf{2}]^T$	$\{t_3\}$	$[\times,\times,\times,\times, \mathbf{0}]^T$	False	$\{t_3, t_6\}$
t_5	$[1, 2, 2, 1]^T$; $[1, 2, 1, 2]^T$; $[1, 2, 0, 3]^T$	$[1, 2, 2, 1, \mathbf{1}]^T$; $[1, 2, 1, 2, \mathbf{2}]^T$; $[1, 2, 0, 3, \mathbf{3}]^T$	$\varnothing$	$[\times,\times,\times,\times, \mathbf{1}]^T$	False	$\varnothing$
…	…	…	…	…	…	…

4.6 Conclusions

In this chapter, we investigate the tolerant control problem for PN systems vulnerable to SD-attacks considering the control specification in terms of a GMEC. Three methods are proposed to obtain on-line control policies. Method 4.1 derives an optimal policy requiring marking enumeration; so it is limited by the size of the considered system and the number of alterations caused by the SD-attack. Method 4.2 also provides an optimal policy. It introduces monitors to the plant by a place-invariant method but still requires marking enumeration and thus suffers from similar problems as Method 4.1 does. To improve the on-line computational efficiency, we propose Method 4.3 that is modified based on Method 4.2 and computes a control policy by avoiding marking enumeration. Consequently, Method 4.3 ensures the timely response of the computed policy but at the expense of its optimality.

The work of this chapter is now under review by *IEEE Transactions on Automatic Control*.

CHAPTER V

Liveness Enforcement on DES Vulnerable to Network Attacks

5.1 Introduction

In this chapter, we study the problem of liveness enforcement on DES still assuming that events are all controllable and observable but the system is vulnerable to SD-attacks. Specifically, we consider the plant modelled as a bounded PN and the control specification consisting in liveness enforcing. Based on repeatedly computing a more restrictive liveness-enforcing supervisor under no attack and constructing a so-called *basic supervisor*, an off-line method that synthesizes a liveness-enforcing supervisor tolerant to an SD-attack is proposed.

This chapter is organized as follows. Section 5.2 formalizes the problem statement. In Section 5.3, we introduce a so-called supervisor graph that is used to intuitively represent supervisors. In Section 5.4, we show how to enforce liveness on a bounded PN system in the presence of SD-attacks. Study cases are presented in Section 5.5. Conclusions are finally discussed in Section 5.6.

We notice that the work of this chapter is now accepted by *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

5.2 Problem Statement

In this section, we formalize the problem studied in this chapter. As in the last chapter, we deal with a plant vulnerable to an SD-attack, assuming that events are all controllable and observable. Differently from the last chapter where the state specification characterized by GMEC is considered, we consider in this chapter the control specification that is liveness enforcing. Besides, the plant we consider in this chapter is a bounded system, which differs from the plant in the last chapter that can be bounded or unbounded.

***Problem* 5.1**: Given a bounded PN system (N, m_0) vulnerable to an SD-attack $\mathcal{A}$, design a liveness-enforcing supervisor that is tolerant to the SD-attack $\mathcal{A}$.

A liveness-enforcing supervisor tolerant to an SD-attack indicates that the supervisor guarantees the liveness of the controlled system even in the presence of the SD-attack. We recall that a supervisor in the presence of an SD-attack $\mathcal{A}$ is ρ: $L_o^{\mathcal{A}}(N, m_0) \rightarrow 2^T$. Furthermore, a liveness-enforcing supervisor tolerant to an SD-attack $\mathcal{A}$ is $\rho : L_o^{\mathcal{A}}(N, m_0) \rightarrow 2^T$, such that the controlled system $(N, m_0)|_\rho$ is live.

We note that, differently from the last chapter that computes on-line control policies, the focus of this chapter is to synthesize a supervisor off-line. Once the supervisor is synthesized off-line, little on-line computation is required. In Problem 5.1, the restriction of the considered PN system to be bounded allows us to synthesize such a supervisor based on constructing its *reachability graph*, which leads to our solution to Problem 5.1 inevitably exponential with the PN size.

We recall that a *reachability graph* (RG) [69] is a graph representation of the evolution of a bounded PN system (N, m_0), where each node represents a unique reachable marking $m \in R(N, m_0)$ and arcs represent the firing of transitions, which lead from one marking to another one (which may also be coincident with the previous one).

Example 5.1: Consider the PN system (N, m_0) in Fig. 5.1. Its RG is shown in Fig. 5.2. ♦

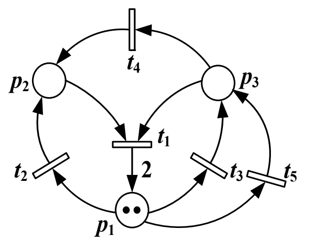


Fig. 5.1 PN system (N, m_0)

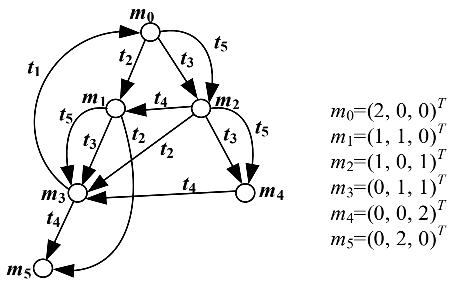


Fig. 5.2 RG of the PN system in Fig. 5.1

5.3 Supervisor Graph

In this section, we propose a structure called *supervisor graph*, which is used throughout this chapter to intuitively represent a supervisor.

Definition 5.1: Given a PN system (N, m_0), a *supervisor graph* is a 5-tuple $\mathcal{G}_s=(X, E, x_0, l_e, Fb)$, where

- (X, E) is a directed graph with set of nodes X and set of ordered pairs of nodes E, called *directed arcs*;
- x_0 is a node in X called the *initial node*;
- l_e: $E\rightarrow 2^T$ is the *arc labelling function* that labels arcs with sets of transitions;
- Fb: $X\rightarrow 2^T$ is the *forbidden function* that associates to a node a set of transitions that are forbidden at such a node.

In simple words, a supervisor graph is a directed graph with an initial node, where each arc is labeled with a set of transitions and each node is associated with a set of transitions that are forbidden to fire by the supervisor, which is called the *forbidden set*. Graphically, an arc labelled by a set of transitions can also be depicted as a set of arcs, each one labeled with a single transition. Besides, given a node x in a supervisor graph, each transition $t\in Fb(x)$ is depicted as "$x\bigcirc\!\!-\!\!\!\not\!\xrightarrow{t}$".

We note that a control policy requires that the control action corresponding to an observed sequence is uniquely determined. Hence, a supervisor graph representing a control policy should be *deterministic*. In other words, given a node x and a sequence $\sigma\in T^*$, the node accessible from x via sequential arcs labeled σ is uniquely determined.

We use $ac(x, \sigma)$ to denote such a node in this work. Note that $ac(x, \sigma)$ is not defined if this node does not exist. Besides, we simply write $ac(x, \sigma)$ as $ac(\sigma)$ if x is the initial node x_0. Specifically, given a supervisor graph, when a sequence σ is observed, the control action is forbidding the firing of transitions in $Fb(x)$, where $x=ac(\sigma)$.

Example 5.2: The graph in Fig. 5.3 is a supervisor graph with initial node x_0. We can see that $Fb(x_4)=\{t_4\}$ and $Fb(x')=\emptyset$, $\forall x'\in\{x_0\text{-}x_3, x_5, x_6\}$. Obviously, it is deterministic. As an example, consider a sequence $\sigma=t_2t_3$. Looking at the supervisor graph we know that it is $ac(\sigma)=x_4$ and we should forbid the firing of transition t_4 after observing σ. ♦

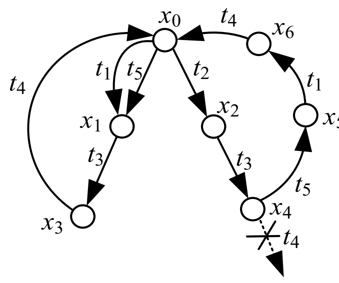


Fig. 5.3 Supervisor graph

An RG can be regarded as a supervisor graph with the initial marking m_0 as the initial node and $Fb(x)=\emptyset$ for each node x, i.e., the firing of no transition is forbidden by the supervisor during the evolution of the net system (N, m_0).

Finally, we partition nodes in a supervisor graph into two categories as follows.

Definition 5.2: Given a supervisor graph $\mathcal{G}_s=(X, E, x_0, l_e, Fb)$ and a node $x\in X$, we define $S(x)$ the set of sequences that lead to node x from the initial node x_0, i.e., $S(x)=\{\sigma\in T^*|\ ac(\sigma)=x\}$. Moreover, given a sequence $\sigma\in S(x)$, we call σ an *elementary sequence* of x if $\nexists\sigma'\in S(x)\setminus\{\sigma\}$ such that σ' is the prefix of σ, i.e., $\sigma=\sigma'\sigma''$, where $\sigma''\in T^*$. We use $S_e(x)$ to denote the set of elementary sequences of x. Furthermore, x is said a *multi-sequence* node if $|S_e(x)|\geq 2$, otherwise it is said a *single-sequence* node.

Equivalently, an elementary sequence of a node x is a transition sequence associated with an elementary path (excluding cycles) from the initial node x_0 to node x. Clearly, the initial node x_0 is a single-sequence node with $S_e(x)=\{\varepsilon\}$.

Example 5.3: Consider again the supervisor graph in Fig. 5.3. We can see that x_1 and x_3 are multi-sequence nodes, while the other nodes are single-sequence nodes. Specifically, $S_e(x_1)=\{t_1, t_5\}$ and $S_e(x_3)=\{t_1t_3, t_5t_3\}$. ♦

5.4 Liveness Enforcement Under SD-attacks

In this section, we investigate how to solve Problem 5.1. Firstly, we propose a way to synthesize a maximally permissive liveness-enforcing supervisor under no attack. Secondly, we introduce the construction of the so-called "basic supervisor" $\mathcal{G}_{s|\mathcal{A}}$ relative to a supervisor $\mathcal{G}_s$ under no attack and a given SD-attack $\mathcal{A}$. Thirdly, an approach is proposed to compute a liveness-enforcing supervisor tolerant to the given SD-attack.

5.4.1 Maximally Permissive Liveness-enforcing Supervisor Under No Attack

In this subsection, we introduce a function called *MaxLiveEnforce*. Given an arbitrary supervisor $\mathcal{G}_s$ and the set of transitions of the considered PN as inputs, it computes a liveness-enforcing supervisor $\mathcal{G}_s'$ that is as permissive as possible but no more permissive than $\mathcal{G}_s$. When $\mathcal{G}_s$ coincides with the RG of the PN system, the function returns a maximally permissive liveness-enforcing supervisor. Note that supervisors in this chapter are provided in the form of supervisor graphs. Besides, given an SCC ϕ in the supervisor graph, we use $T(\phi)$ to denote the set of transitions labeling the arcs in ϕ.

In words, given a supervisor $\mathcal{G}_s$ and the transition set T of a PN as inputs, Function *MaxLiveEnforce* works as follows: First, it computes all SCCs in $\mathcal{G}_s$, excluding those that are exactly a single node. Next, for every SCC not containing all transitions in T, it merges all nodes and arcs in the SCC into a node. Then, it repeatedly deletes sink nodes and updates accordingly the forbidden sets of the input nodes of sink nodes. When no sink node exists, merged nodes, if any, are unfolded into their original SCCs and the resulting graph is the supervisor $\mathcal{G}_s'$.

Function $\mathcal{G}_s'$=MaxLiveEnforce($\mathcal{G}_s$, T)

```
Input: A supervisor $\mathcal{G}_s=(X, E, x_0, l_e, Fb)$ and the transition set $T$ of a PN;
Output: A supervisor $\mathcal{G}_s'=(X', E', x_0', l_e', Fb')$.
1) $\Phi:=Tarjan(\mathcal{G}_s)$; /*Function Tarjan [91] here returns the set of SCCs excluding those that are
   exactly a single node in $\mathcal{G}_s$ */
2) for each $\phi\in\Phi$ do
3)     if $T\not\subset T(\phi)$ then
4)         merge all nodes and arcs in the SCC $\phi$ into a single node denoted as $x_\phi$;
5)     end if
6) end for
7) while there exists a sink node $x$ in $\mathcal{G}_s$ do
8)     for each $x'\in{}^{\bullet}x$ do
9)         $Fb(x'):= Fb(x')\cup T'$, where $T'$ is the set of transitions labeling the arcs
           from $x'$ to $x$;
10)    end for
11)    delete $x$ and its input arcs from $\mathcal{G}_s$;
12) end while
13) unfold merged nodes (if any) into the original SCCs;
14) let $\mathcal{G}_s':=\mathcal{G}_s$;
15) output: $\mathcal{G}_s'$.
```

An example is given next to illustrate Function *MaxLiveEnforce* more intuitively. Note that we do not present the PN system for saving space but simply give its transition set T.

Example 5.4: Consider the supervisor $\mathcal{G}_s$ in Fig. 5.4(a) for a PN system with $T=\{t_1$-$t_5\}$. Let us see how *MaxLiveEnforce*($\mathcal{G}_s$, T) works. First, all SCCs excluding single nodes in $\mathcal{G}_s$ are computed, which are shown in Fig. 5.4(b) denoted as A, B_1, B_2, respectively. Next, since SCCs B_1 and B_2 do not contain all transitions in T, nodes and arcs in them are merged into nodes B_1 and B_2, respectively. The resulting graph is depicted in Fig. 5.4(c). Now, we repeatedly delete sink nodes and update the forbidden sets. Nodes m_9, m_{10} and B_2 are thereby deleted. Finally, after unfolding node B_1, we get the supervisor $\mathcal{G}_s'$ in Fig. 5.4(d) that is the output of Function *MaxLiveEnforce*($\mathcal{G}_s$, T). ♦

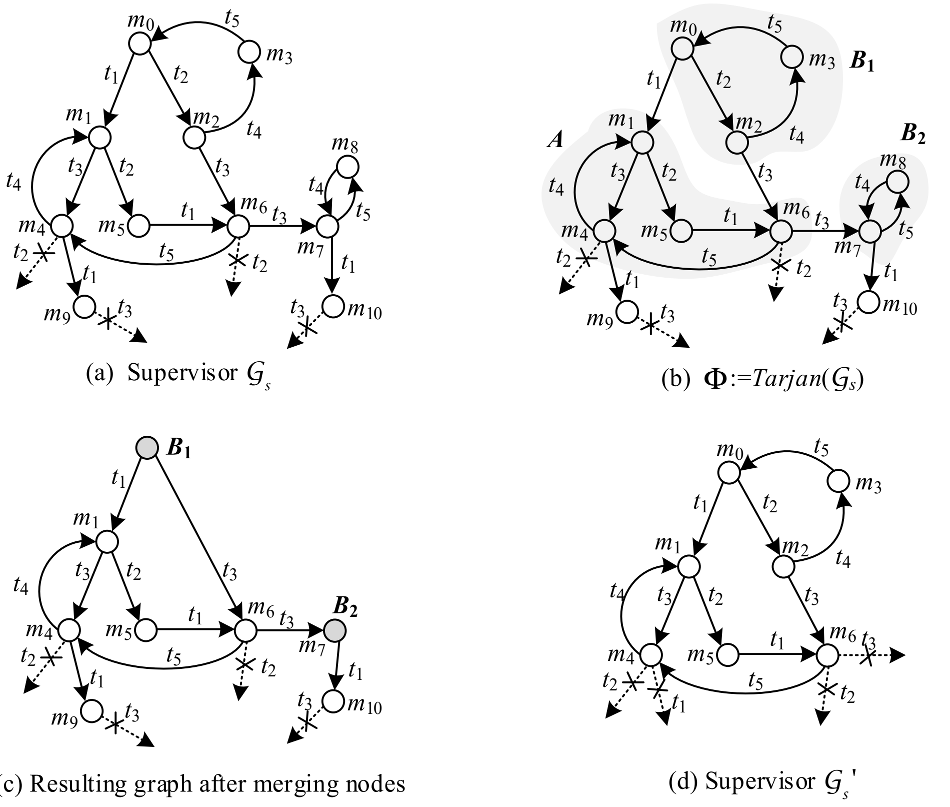

(a) Supervisor $\mathcal{G}_s$
(b) Φ:=*Tarjan*($\mathcal{G}_s$)
(c) Resulting graph after merging nodes
(d) Supervisor $\mathcal{G}_s'$

Fig. 5.4 Example to illustrate the computation of $\mathcal{G}_s'$=*MaxLiveEnforce*($\mathcal{G}_s$, T), where $T=\{t_1\text{-}t_5\}$

Proposition 5.1: Consider a PN system (N, m_0) with N=(P, T, F, W), a supervisor $\mathcal{G}_s$ and $\mathcal{G}_s'$=*MaxLiveEnforce*($\mathcal{G}_s$, T).

1) $\mathcal{G}_s'$ is a liveness-enforcing supervisor of (N, m_0) if $\mathcal{G}_s'\neq\emptyset$;

2) for any liveness-enforcing supervisor $\mathcal{G}_s''$ of (N, m_0) s.t. $\mathcal{G}_s''\preceq\mathcal{G}_s$, it holds that $\mathcal{G}_s''\preceq\mathcal{G}_s'$.

Proof: We preliminarily define that an SCC is a *sink SCC* if none of the nodes in the SCC can reach a node outside the SCC.

1) We observe that $\mathcal{G}_s'\neq\emptyset$ satisfies the following two conditions: (a) it contains no sink node; (b) if a sink SCC exists, it contains all transitions in T. Hence, an SCC containing all transitions in T can always be reached from any node in $\mathcal{G}_s'$. Trivially, $\forall t\in T$, t is live at m_0 in the system supervised by $\mathcal{G}_s'$. Hence, $\mathcal{G}_s'$ is a liveness-enforcing supervisor.

2) Suppose by contradiction that $\mathcal{G}_s''\npreceq\mathcal{G}_s'$. Since by assumption, it is $\mathcal{G}_s''\preceq\mathcal{G}_s$, then in $\mathcal{G}_s''$ there exists either a sink node or a sink SCC that does not contain all transitions in T. The system supervised by $\mathcal{G}_s''$ thereby contains deadlocks or local deadlocks, which contradicts the assumption that $\mathcal{G}_s''$ is a liveness-enforcing supervisor. Hence, it is $\mathcal{G}_s''\preceq\mathcal{G}_s'$. ■

Straightforward from Proposition 5.1, the following conclusion is drawn in the case that the RG of a PN system is the input of Function *MaxLiveEnforce*.

Corollary 5.1: Given a PN system (N, m_0) with $N=(P, T, F, W)$ and its RG $\mathcal{G}$,

1) no liveness-enforcing supervisor exists for (N, m_0) if and only if *MaxLiveEnforce*$(\mathcal{G}, T)=\emptyset$; and

2) $\mathcal{G}_s=$*MaxLiveEnforce*$(\mathcal{G}, T)$ is a maximally permissive liveness-enforcing supervisor for (N, m_0) if $\mathcal{G}_s\neq\emptyset$.

Proof: The RG $\mathcal{G}$ can be viewed as a maximally permissive supervisor for (N, m_0). Hence, the two results trivially hold by Proposition 5.1. ■

Example 5.5: Consider the PN system (N, m_0) in Fig. 5.1 with $T=\{t_1\text{-}t_5\}$ and its RG $\mathcal{G}$ in Fig. 5.2. Function *MaxLiveEnforce*$(\mathcal{G}, T)$ returns the maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ in Fig. 5.5. ♦

Note that, if there is no ambiguity, we may identify the nodes of a supervisor graph $\mathcal{G}_s$ using their corresponding markings. As an example, if we consider the supervisor

$\mathcal{G}_s$ in Fig. 5.5, we say that $Fb(m_1)=\{t_2\}$, $Fb(m_3)=\{t_4\}$, and $Fb(x)=\varnothing$ for any other node x in $\mathcal{G}_s$.

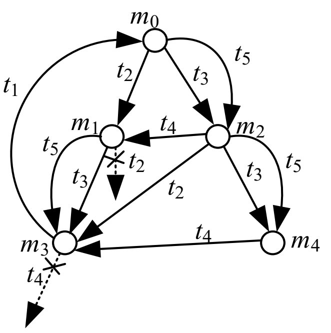


Fig. 5.5 Maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ of the PN system in Fig. 5.1

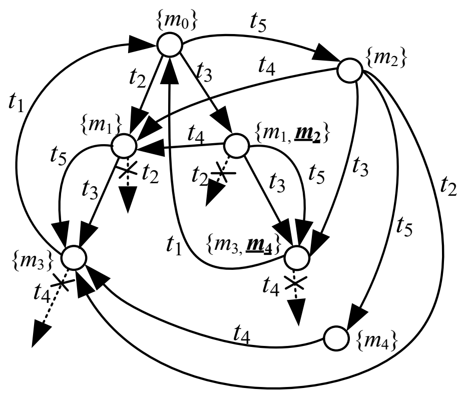


Fig. 5.6 Basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ relative to the supervisor $\mathcal{G}_s$ in Fig. 5.5 and the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$

Remark 5.1: Various approaches exist in the literature [53] to compute a maximally permissive liveness-enforcing supervisor for a PN system under no attack. Function *MaxLiveEnforce* is presented here for making the chapter self-contained. In addition, as indicated by Proposition 5.1, given an arbitrary supervisor $\mathcal{G}_s$ as the input, it computes a liveness-enforcing supervisor $\mathcal{G}_s'$ that is as permissive as possible but no more permissive than $\mathcal{G}_s$. This will be a fundamental step in the approach we propose to find a liveness-enforcing supervisor tolerant to an SD-attack. ♣

Remark 5.2: A liveness-enforcing supervisor under no attack typically cannot guarantee the liveness of the controlled system in the presence of an SD-attack. In addition, it could even be an "infeasible" supervisor in the presence of an SD-attack. In other words, it does not associate a control action with every observable sequence since unexpected sequences may be observed due to the SD-attack.

To clarify the above issue, let us assume that the PN system (N, m_0) in Fig. 5.1 is vulnerable to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$. Let us see what may happen if the system is supervised by $\mathcal{G}_s$ in Fig. 5.5. Clearly, $\mathcal{G}_s$ does not forbid the occurrence of any transition when the firing of t_3 is observed. However, due to the attack $\mathcal{A}=\{(t_2, t_3)\}$, it could happen that the observation of t_3 corresponds to the occurrence of t_2. Now, since $\mathcal{G}_s$ forbids no transition after the observation of t_3, then t_2 is enabled to fire. In other words, the sequence t_2t_2 may fire in the supervised system, which however is a sequence that prevents liveness. Therefore, to avoid the occurrence of the sequence t_2t_2 in the system vulnerable to the attack $\mathcal{A}=\{(t_2, t_3)\}$, it is necessary to forbid the occurrence of t_2 when the firing of t_3 is observed. ♣

In essence, to design an SD-attack-tolerant liveness-enforcing supervisor, when a sequence is observed, we have to consider all possible sequences consistent with the observation and forbid all the "bad" ones, namely all those who lead the system to be not live. In the following subsections, we show how to solve this problem.

5.4.2 Basic Supervisor Under SD-attacks

In this subsection, we introduce the so-called "*basic supervisor*" $\mathcal{G}_{s|\mathcal{A}}$ relative to a given supervisor $\mathcal{G}_s$ under no attack and an SD-attack $\mathcal{A}$. We show that the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is a "feasible" supervisor in the presence of the SD-attack $\mathcal{A}$ and guarantees that the sequences forbidden by $\mathcal{G}_s$ can never occur in the controlled system.

Before formally presenting it, we introduce more notation related to supervisor graphs. Given a supervisor graph $\mathcal{G}_s=(X, E, x_0, l_e, Fb)$ and a node $x\in X$, we use $T_{out}(x)$ to denote the set of transitions labeling output arcs of x. The notation is extended to a set of nodes $X'\subseteq X$ such that $T_{out}(X')=\bigcup_{x\in X'} T_{out}(x)$. In addition, given a set of nodes $X'\subseteq X$ and a sequence $\sigma\in T^*$, we denote $Ac(X', \sigma)$ the set of nodes accessible from X' via sequential arcs labeled σ, i.e.,

$$Ac(X', \sigma)=\{ac(x, \sigma) \mid x\in X' \wedge ac(x, \sigma)!\},$$

where "!" means "*is defined*".

Example 5.6: Consider the supervisor in Fig. 5.5 and node m_1. It is $T_{out}(m_1)=\{t_3, t_5\}$. Now, let $X'=\{m_1, m_2\}$. It is $T_{out}(X')=T_{out}(m_1)\cup T_{out}(m_2)=\{t_2\text{-}t_5\}$. Furthermore, given a sequence $\sigma_1=t_3$, it holds $Ac(X', \sigma_1)=\{ac(m_1, \sigma_1), ac(m_2, \sigma_1)\}=\{m_3, m_4\}$. Finally, given a sequence $\sigma_2=t_3t_1$, it is $Ac(X', \sigma_2)=\{ac(m_1, \sigma_2)\}=\{m_0\}$. ♦

Let us now discuss how the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is constructed. Note that each node in $\mathcal{G}_{s|\mathcal{A}}$ is labeled by a subset of nodes of $\mathcal{G}_s$. Formally, we introduce a *node labelling function*

$$l_v: Y\rightarrow 2^X,$$

where Y is the set of nodes of $\mathcal{G}_{s|\mathcal{A}}$ and X is the set of nodes of $\mathcal{G}_s$. Besides, to distinguish functions in $\mathcal{G}_s$ from those in $\mathcal{G}_{s|\mathcal{A}}$, we use the superscript $\mathcal{G}_s$ to mark functions in $\mathcal{G}_s$, while we use no superscript to indicate the corresponding functions in $\mathcal{G}_{s|\mathcal{A}}$.

The construction of a basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is done using Function *ConstructBasicSG*, where Function *CreateNode* is called. In addition to the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, Function *ConstructBasicSG* outputs a so-called *pruning set* Γ that will be useful to develop a liveness-enforcing supervisor tolerant to SD-attacks.

```
Function ($\mathcal{G}_{s|\mathcal{A}}$, Γ)=ConstructBasicSG($\mathcal{G}_s$, $\mathcal{A}$)
```

```
Input: A supervisor $\mathcal{G}_s=(X, E^{\mathcal{G}s}, x_0, l_e^{\mathcal{G}s}, Fb^{\mathcal{G}s})$ under no attack and an SD-attack $\mathcal{A}$.
Output: The basic supervisor $\mathcal{G}_{s|\mathcal{A}}=(Y, E, y_0, l_e, Fb)$ and the pruning set Γ.
1)  $\Gamma:=\varnothing$; /*Γ is a global variable updated in Function CreateNode*/
2)  initialize a graph $\mathcal{G}$ with initial node $y_0$ such that $l_v(y_0):=\{x_0\}$ and $Fb(y_0):=$
    $Fb^{\mathcal{G}s}(x_0)$, and tag $y_0$ “new”;
3)  while there exists a “new” node $y$ in $\mathcal{G}$ do
4)      $T_{real}(y):=T_{out}^{\mathcal{G}s}(l_v(y))\backslash Fb(y)$;
        /*$T_{real}(y)$ denotes the set of all possible really firable transitions at node $y$. */
5)      $T_{out}(y):=\bigcup_{t'\in T_{real}(y)} A(t')$;
        /*$T_{out}(y)$ denotes the set of all possible observed transition firings at node $y$ due to the
        existence of SD-attack $\mathcal{A}$. */
6)      for each $t\in T_{out}(y)$ do
7)          if $t\notin T_u$ then
8)              $X' := Ac^{\mathcal{G}s}(l_v(y), t)$;
9)          else
10)             $X' := \bigcup_{t'\in A^{-1}(t)\cap T_{real}(y)} Ac^{\mathcal{G}s}(l_v(y), t')$ ;
11)         end if
12)         $\mathcal{G}$ :=CreateNode($\mathcal{G}$, $y$, $t$, $X'$);
13)     end for
14)     untag “new” from node $y$;
15) end while
16) let $\mathcal{G}_{s|\mathcal{A}}:=\mathcal{G}$;
17) Output: $\mathcal{G}_{s|\mathcal{A}}$ and Γ.
```

```
Function $\mathcal{G}$ =CreateNode($\mathcal{G}$, $y$, $t$, $X'$)
```

```
Input: A graph $\mathcal{G}=(Y, E, y_0, l_e, Fb)$, a node $y$, a transition $t$ and a node set $X'$;
Output: An updated graph $\mathcal{G}=(Y, E, y_0, l_e, Fb)$.
1)  if there exists a node $y'$ in $\mathcal{G}$ with $l_v(y')=X'$ then
2)      add an arc from $y$ to $y'$ labeled $t$;
```

```
3)  else
4)      create a node $y'$ with $l_v(y')$:=$X'$ and add an arc from $y$ to $y'$ labeled $t$;
5)      $Fb(y')$:=$\bigcup_{x\in l_v(y')} Fb^{\mathcal{G}_s}(x)$;
6)      tag node $y'$ "new";
    *****Steps 7-13 update the pruning set $\Gamma$*****
7)      if $|l_v(y')|\geq 2$ then
8)          for each $x\in l_v(y')$ do
9)              if $T_{out}^{\mathcal{G}s}(x)\cap Fb(y')\neq\emptyset$ then
10)                 $\Gamma$:=$\Gamma\cup\{(x, y')\}$;
11)             end if
12)         end for
13)     end if
    ***************************************
14) end if
15) denote the update graph as $\mathcal{G}$;
16) Output: $\mathcal{G}$.
```

We explain the construction of the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ by calling Function *ConstructBasicSG*($\mathcal{G}_s$, $\mathcal{A}$). First, a graph $\mathcal{G}$ is initialized with initial node y_0 labeled by set $\{x_0\}$ and associated with a forbidden set equal to the forbidden set of x_0 in $\mathcal{G}_s$. This node is tagged "new". Next, we select a "new" node, denoted as y, to generate its output nodes by Steps 3-15:

1) We determine $T_{out}(y)$, i.e., the set of transitions labeling the output arcs of node y. Note that $\mathcal{G}_{s|\mathcal{A}}$ describes the possible observed behavior of the system in the presence of the attack $\mathcal{A}$. Thus, $T_{out}(y)$ enumerates all possible transition firings that could be observed at node y. Clearly, the firing of some transitions may be disguised as the firing of other transitions by the intruder. Hence, it is $T_{out}(y)\supseteq T_{real}(y)$, where $T_{real}(y)$ denotes

the set of all possible really firable transitions at node y and it is computed at Step 4. Furthermore, by Step 5 it is $T_{out}(y)=\bigcup_{t'\in T_{real}(y)} A(t')$.

2) For each $t\in T_{out}(y)$, we first compute the labeling set of the output node of node y reached via transition t in Steps 7-11. It indeed consists of the nodes in $\mathcal{G}_s$ reachable from the nodes constituting the labeling set of y via a transition whose observation may be t. If t is not an uncertain transition, we know for sure that t has fired if it has been observed. Thus, the labeling set denoted by X' is computed at Step 8. In the case that t is an uncertain transition, other transitions could have fired apart from t when t is observed. As a result, we need to consider all the possible transitions that produce the observation of t and the labeling set X' is thereby computed at Step 10. Next, *CreateNode*$(\mathcal{G}, y, t, X')$ is called to update the graph $\mathcal{G}$. In particular, we create a "new" node y' labeled with the set X', if it does not exist already. We add an arc from y to y' labeled t and compute its forbidden set at Step 5, which is the union of forbidden sets associated with the nodes in $\mathcal{G}_s$ that constitute the labeling set of node y'.

After generating all the output nodes of y, we untag it. Then, we repeatedly look for "new" nodes to generate their output nodes by iteratively executing the above operations. Finally, in the case that no "new" node exists, the graph $\mathcal{G}$ is exactly the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$.

Now, let us focus on the pruning set Γ. Clearly, it is initialized at the empty set and every time a new node is constructed, the pruning set Γ is considered to be updated at Steps 7-13 of function *CreateNode*. Specifically, we only consider new nodes whose labeling sets contain two or more elements. Let y' be such a new node. For each element $x\in l_V(y')$, we check if it holds $T_{out}^{\mathcal{G}s}(x)\cap Fb(y')\neq\emptyset$. If so, the pruning set Γ is updated by including the pair (x, y').

The following example illustrates the computation of the basic supervisor and the pruning set, and their physical meaning is clarified then.

Example 5.7: Consider again the PN system (N, m_0) in Fig. 5.1 and the supervisor $\mathcal{G}_s$ in Fig. 5.5 under no attack. Let us see how to compute the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ in the presence of the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$ and the pruning set Γ by calling *ConstructBasicSG*($\mathcal{G}_s$, $\mathcal{A}$).

First, we initialize the pruning set Γ at the empty set and initialize the graph at the initial node y_0 tagged "new" with label $l_v(y_0)=\{m_0\}$ and forbidden set $Fb(y_0)=Fb^{\mathcal{G}s}(m_0)=\varnothing$. Next, we generate the output nodes of y_0 as follows.

1) We compute $T_{out}(y_0)$. Clearly, $T_{real}(y_0)=T_{out}^{\mathcal{G}s}(l_v(y_0))\backslash Fb(y_0)=\{t_2, t_3, t_5\}$. Moreover, $T_{out}(y_0)=\bigcup_{t'\in T_{real}(y_0)} A(t')=\{t_2, t_3, t_5\}$.

2) For each $t\in T_{out}(y_0)$, we compute the labeling set of the output node of y_0 reached via the arc labeled t and update the graph by generating such a node. Note that $T_u=\{t_3\}$ since $\mathcal{A}=\{(t_2, t_3)\}$. Now, consider $t_2\in T_{out}(y_0)$. Clearly, set $X'=Ac^{\mathcal{G}s}(\{m_0\}, t_2)=\{m_1\}$. Then, by calling *CreateNode*($\mathcal{G}$, y_0, t_2, X'), the "new" node y_1 with $l_v(y_1)=\{m_1\}$ and $Fb(y_1)=Fb^{\mathcal{G}s}(m_1)=\{t_2\}$ is created as the output of y_0 reached via the arc labeled t_2. Consider $t_3\in T_{out}(y_0)$. We can see that $A^{-1}(t_3)\cap T_{real}(y_0)=\{t_2, t_3\}$. Accordingly, it is $X' = \bigcup_{t'\in\{t_2,t_3\}} Ac^{\mathcal{G}s}(\{m_0\},t')=\{m_1, m_2\}$. Then, by calling *CreateNode*($\mathcal{G}$, y_0, t_3, X'), the "new" node y_2 with $l_v(y_2)=\{m_1, m_2\}$ and $Fb(y_2)=Fb^{\mathcal{G}s}(m_1)\cup Fb^{\mathcal{G}s}(m_2)=\{t_2\}$ is created as the output of y_0 reached via the arc labeled t_3. Note that Γ is now updated as $\Gamma=\{(m_2, y_2)\}$ since $T_{out}^{\mathcal{G}s}(m_2)\cap Fb(y_2)=\{t_2\}$. Similarly, the "new" node y_3 with $l_v(y_3)=\{m_2\}$ and $Fb(y_3)=Fb^{\mathcal{G}s}(m_2)=\varnothing$ is created as the output of y_0 reached via the arc labeled t_5.

Then, we untag y_0 and select another "new" node to generate its output nodes. By repeating the above operations, when no "new" node can be found, we finish the construction of the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, which is shown in Fig. 5.6. Besides, we obtain the final pruning set $\Gamma=\{(m_2, \{m_1, m_2\}), (m_4, \{m_3, m_4\})\}$. Note that, for sake of clarity, the nodes of $\mathcal{G}_{s|\mathcal{A}}$ are identified by their corresponding sets of markings (which can also be a singleton as a special case). ♦

We notice that $\mathcal{G}_s$ not only is a supervisor under no attack but also shows the real evolution of the controlled system. Roughly speaking, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is like an observer of $\mathcal{G}_s$ in the presence of the SD-attack $\mathcal{A}$ but is typically more restrictive than $\mathcal{G}_s$.

In more detail, $\mathcal{G}_{s|\mathcal{A}}$ describes all possible observations of the controlled system in the presence of the SD-attack $\mathcal{A}$ and uses the label of each node, which is a subset of nodes in $\mathcal{G}_s$, to store the real evolution information of the controlled system corresponding to the observed evolution information. Indeed, the label of a node y in $\mathcal{G}_{s|\mathcal{A}}$ reached via a certain sequence δ, is the set of nodes in $\mathcal{G}_s$ that are really reachable when the observation is δ, taking into account all possible corruptions by the attack $\mathcal{A}$. In addition, the forbidden set associated with the generic node y in $\mathcal{G}_{s|\mathcal{A}}$ is the union of the forbidden sets associated with the nodes of $\mathcal{G}_s$ whose union generates the labeling set of node y, namely, it is $Fb(y)=\bigcup_{x\in l_v(y)} Fb^{\mathcal{G}_s}(x)$. This guarantees that none of the sequences forbidden by $\mathcal{G}_s$ can occur in the system supervised by $\mathcal{G}_{s|\mathcal{A}}$ even in the presence of the SD-attack $\mathcal{A}$. This also typically results in having $\mathcal{G}_{s|\mathcal{A}}$ more restrictive than $\mathcal{G}_s$.

Considering that $\mathcal{G}_{s|\mathcal{A}}$ is typically more restrictive than $\mathcal{G}_s$, the pruning set Γ is computed to save the information on which behavior of the plant may be additionally restricted by $\mathcal{G}_{s|\mathcal{A}}$ compared with $\mathcal{G}_s$. Formally, it is

$$\Gamma=\{(x, y)\in X\times Y|\ x\in l_v(y) \wedge (T_{out}^{\mathcal{G}s}(x)\cap Fb(y)\neq\varnothing)\},$$

where X and Y are the sets of nodes of $\mathcal{G}_s$ and $\mathcal{G}_{s|\mathcal{A}}$, respectively.

In words, the pruning set is a set of node pairs, where the first entry is a node in $\mathcal{G}_s$ and the second entry is a node in $\mathcal{G}_{s|\mathcal{A}}$. Consider a generic pair (x, y) in Γ. Node x is an element in the labeling set of node y, indicating that x is a possible really reached node corresponding to any observation leading to node y, and there exists a transition that is firable at node x in the supervisor $\mathcal{G}_s$ but forbidden to fire at node y in the supervisor $\mathcal{G}_{s|\mathcal{A}}$.

Note that, for each node y in $\mathcal{G}_{s|\mathcal{A}}$ with $|l_v(y)|=1$, assuming that x is the single node in $l_v(y)$, it holds $T_{out}^{\mathcal{G}s}(x)\cap Fb(y)=\emptyset$. This is because $T_{out}^{\mathcal{G}s}(x)\cap Fb^{\mathcal{G}s}(x)=\emptyset$ and $Fb(y)=Fb^{\mathcal{G}s}(x)$. As a result, it is enough to consider each node y in $\mathcal{G}_{s|\mathcal{A}}$ with $|l_v(y')|\geq 2$ to update the pruning set Γ in Function *CreateNode*.

Based on the above analysis, we formally present the following results.

Proposition 5.2: Consider a PN system (N, m_0), a supervisor $\mathcal{G}_s$ under no attack, and an SD-attack $\mathcal{A}$. Let $(\mathcal{G}_{s|\mathcal{A}}, \Gamma)$=*ConstructBasicSG*$(\mathcal{G}_s, \mathcal{A})$. The following three items hold:

1) $\mathcal{G}_{s|\mathcal{A}}$ is a supervisor for (N, m_0) associating a control action with each observation in the presence of the SD-attack $\mathcal{A}$;

2) any sequence $\sigma\in L(N, m_0)$ forbidden by $\mathcal{G}_s$ can never occur in the system supervised by $\mathcal{G}_{s|\mathcal{A}}$;

3) there exists a sequence $\sigma\in L(N, m_0)$ such that σ is permitted by $\mathcal{G}_s$ but forbidden by $\mathcal{G}_{s|\mathcal{A}}$ if and only if $\Gamma\neq\emptyset$.

Proof: 1) Since $\mathcal{G}_s$ is a supervisor under no attack, it associates a forbidden set with each observation when there is no attack. Supervisor $\mathcal{G}_{s|\mathcal{A}}$ is constructed based on $\mathcal{G}_s$. At the initial node y_0 of $\mathcal{G}_{s|\mathcal{A}}$, it is $T_{real}(y_0)=T_{out}^{\mathcal{G}s}(x_0)$. Furthermore, $T_{out}(y_0)$ computed at Step 5 enumerates all the transitions that may be observed as the first one during the evolution of the system (N, m_0) vulnerable to the SD-attack $\mathcal{A}$. Assume t_1 is a transition in $T_{out}(y_0)$ and let $y_1=ac(t_1)$. Clearly, when t_1 is observed, $l_v(y_1)$ is the set of nodes in $\mathcal{G}_s$ that are reached from x_0 due to the firing of a transition t_1' s.t. $t_1'\in A^{-1}(t_1)$ and $t_1'\notin Fb(y_0)$. Hence, according to Step 4, $T_{real}(y_1)$ is the set of all possible really firable transitions after the observation of t_1, while $T_{out}(y_1)$ computed at Step 5 enumerates all the possible observable transitions after the observation of t_1. Now, assume that t_2 is a transition in $T_{out}(y_1)$ and let $y_2=ac(y_1, t_2)$. Similarly, when t_2 is observed, $l_v(y_2)$ is the set of nodes in $\mathcal{G}_s$ that are reached from nodes in $l_v(y_1)$ due to the firing of a transition t_2' s.t. $t_2'\in A^{-1}(t_2)$ and $t_2'\notin Fb(y_1)$. It also implies that when t_1t_2 is observed, $l_v(y_2)$ is the set of nodes in $\mathcal{G}_s$ that are reached from x_0 due to the firing of a sequence $t_1't_2'$ s.t. $t_1't_2'\in A^{-1}(t_1t_2)$ and $t_1'\notin Fb(y_0)$ and $t_2'\notin Fb(y_1)$. Summarizing, given a node y in $\mathcal{G}_{s|\mathcal{A}}$ and an observed sequence δ leading to y in $\mathcal{G}_{s|\mathcal{A}}$, the labeling set $l_v(y)$ is the set of nodes in $\mathcal{G}_s$ that are reached from x_0 firing a sequence σ s.t. $\sigma\in A^{-1}(\delta)$ and σ is not restricted by $\mathcal{G}_{s|\mathcal{A}}$. Finally, it is trivial to see that $\mathcal{G}_{s|\mathcal{A}}$ associates with each observed sequence a forbidden set when there exists the SD-attack $\mathcal{A}$. In other words, it is a supervisor for (N, m_0) vulnerable to the SD-attack $\mathcal{A}$.

2) Let $\sigma\in L(N, m_0)$ be a sequence forbidden by $\mathcal{G}_s$. This means that there exist σ_1, $\sigma_2\in T^*$ and $t\in T$ s.t. $\sigma=\sigma_1t\sigma_2$ and there exists a node x in $\mathcal{G}_s$ s.t. $ac^{\mathcal{G}s}(\sigma_1)=x$ and $t\in Fb^{\mathcal{G}s}(x)$. By the rules of constructing $\mathcal{G}_{s|\mathcal{A}}$, two cases may occur: (a) σ_1 is forbidden by $\mathcal{G}_{s|\mathcal{A}}$; and (b) σ_1 is not forbidden by $\mathcal{G}_{s|\mathcal{A}}$. In case (a), σ is clearly forbidden by $\mathcal{G}_{s|\mathcal{A}}$. Consider case (b). For any node y in $\mathcal{G}_{s|\mathcal{A}}$ with $x\in l_v(y)$, it holds that $t\in Fb(y)$ since $Fb(y)= \bigcup_{x\in l_v(y)} Fb^{\mathcal{G}_s}(x)$. Hence, σ is also forbidden by $\mathcal{G}_{s|\mathcal{A}}$.

3) (=>) The pruning set $\Gamma\neq\varnothing$ implies that there exists a pair $(x, y)\in X\times Y$ with $x\in l_v(y)$, s.t. $T_{out}^{\mathcal{G}s}(x)\cap Fb(y)\neq\varnothing$. Let δ be an observed sequence s.t. $ac(\delta)=y$ and σ_1 be a sequence

s.t. $ac^{\mathcal{G}s}(\sigma_1)=x$, $\sigma_1 \in A^{-1}(\delta)$, and σ_1 is not restricted by $\mathcal{G}_{s|\mathcal{A}}$. Moreover, let $t \in T_{out}^{\mathcal{G}s}(x) \cap Fb(y)$. Sequence $\sigma=\sigma_1 t \in L(N, m_0)$ is permitted by $\mathcal{G}_s$ since $t \in T_{out}^{\mathcal{G}s}(x)$. When δ is observed, σ may be forbidden by $\mathcal{G}_{s|\mathcal{A}}$ since $t \in Fb(y)$.

(<=) Since σ is forbidden by $\mathcal{G}_{s|\mathcal{A}}$, there exist $\sigma_1, \sigma_2 \in T^*$ and $t \in T$ s.t. $\sigma=\sigma_1 t \sigma_2$ and $t \in Fb(y)$, where $y=ac(\delta)$ and $\delta \in A(\sigma_1)$. Since σ is permitted by $\mathcal{G}_s$, it is $t \in T_{out}^{\mathcal{G}s}(x)$, where $x=ac(\sigma_1)$. Hence, $T_{out}^{\mathcal{G}s}(x) \cap Fb(y) \neq \emptyset$. Trivially, $x \in l_v(y)$. As a result, $\Gamma \neq \emptyset$. ■

Based on Proposition 5.2, we can easily derive a sufficient but not necessary condition under which a basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$.

Proposition 5.3: Consider a PN system (N, m_0), a liveness-enforcing supervisor $\mathcal{G}_s$ under no attack, and an SD-attack $\mathcal{A}$. The supervisor $\mathcal{G}_{s|\mathcal{A}}$ is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ for (N, m_0) if $\Gamma=\emptyset$, where $(\mathcal{G}_{s|\mathcal{A}}, \Gamma)=ConstructBasicSG(\mathcal{G}_s, \mathcal{A})$.

Proof: By Proposition 5.2, in the case that $\Gamma=\emptyset$, the behavior of the system controlled by the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is the same as that of the system controlled by the supervisor $\mathcal{G}_s$. Since $\mathcal{G}_s$ is a liveness-enforcing supervisor under no attack, the supervisor $\mathcal{G}_{s|\mathcal{A}}$ is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$. ■

Remark 5.3: Consider the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ relative to a liveness-enforcing supervisor $\mathcal{G}_s$ under no attack. In the case that $\Gamma \neq \emptyset$, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ might not guarantee the liveness of the supervised system since it restricts more behavior than $\mathcal{G}_s$. Fortunately, the real evolution of the system supervised by $\mathcal{G}_{s|\mathcal{A}}$ can be checked based on $\mathcal{G}_s$ and Γ. If the supervised system is not live, one way to guarantee its liveness is designing a supervisor more restrictive than $\mathcal{G}_{s|\mathcal{A}}$. Based on this idea, a method is developed in the next subsection to compute a liveness-enforcing supervisor tolerant to SD-attacks. ♣

Remark 5.4: As mentioned before, given a node x in a supervisor $\mathcal{G}_s$ under no attack, the forbidden set of x and the set of transitions labeling output arcs of x are disjoint sets, i.e., $T_{out}^{\mathcal{G}s}(x)\cap Fb^{\mathcal{G}s}(x)=\emptyset$. Specifically, it is $T_{out}^{\mathcal{G}s}(x)=En(m)\backslash Fb^{\mathcal{G}s}(x)$, where m is the corresponding marking of x. In contrast, given a node y in a basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, it could happen that $T_{out}(y)\cap Fb(y)\neq\emptyset$. Indeed, it is true that $T_{real}(y)$ and $Fb(y)$ are disjoint sets since $T_{real}(y)$ is the set of all possible really firable transitions at node y. However, $T_{out}(y)=\bigcup_{t'\in T_{real}(y)} A(t')$. Hence, $T_{out}(y)$ and $Fb(y)$ are not necessarily disjoint sets. This shows the necessity of associating a forbidden set with each node to represent a control policy in the case that there exists an SD-attack. ♣

5.4.3 Liveness-enforcing Supervisor Tolerant to SD-attacks

In this subsection, we propose an approach that computes a liveness-enforcing supervisor tolerant to SD-attacks. Before that, we introduce two functions as follows.

First, we introduce a function called *SeparateNode*. Specifically, it is $\mathcal{G}_s'=SeparateNode(\mathcal{G}_s, X_m)$, where $\mathcal{G}_s$ and $\mathcal{G}_s'$ are supervisor graphs and X_m is a set of multi-sequence nodes in $\mathcal{G}_s$. The function works like this: For each node $x\in X_m$ that corresponds to n elementary sequences, the supervisor graph $\mathcal{G}_s$ is updated such that node x is separated into n nodes x^1, x^2, …, x^n with each node x^i, $i\in\{1, 2, …, n\}$, corresponding to a single and different elementary sequence associated with x. Note that the multi-sequence nodes in the paths from the initial node to node x have to be separated as well to guarantee that each separated node of x is a single-sequence node.

Example 5.8: Consider the supervisor $\mathcal{G}_s$ in Fig. 5.5 and let $X_m=\{m_2, m_4\}$. By computing $\mathcal{G}_s=SeparateNode(\mathcal{G}_s, X_m)$, we obtain the updated $\mathcal{G}_s$ as shown in Fig. 5.7. ♦

We note that Function *SeparateNode* actually performs the *state space refinement* [15]. It is clear that the supervisors before and after calling Function *SeparateNode* are the same. They only differ for their graphical representation.

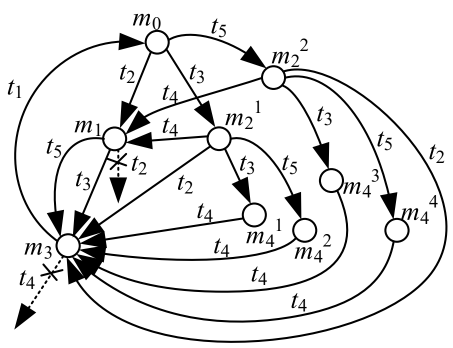


Fig. 5.7 Supervisor $\mathcal{G}_s$=*SeparateNode*($\mathcal{G}_s$, X_m), where the input $\mathcal{G}_s$ is the supervisor in Fig. 5.5 and X_m={m_2, m_4}

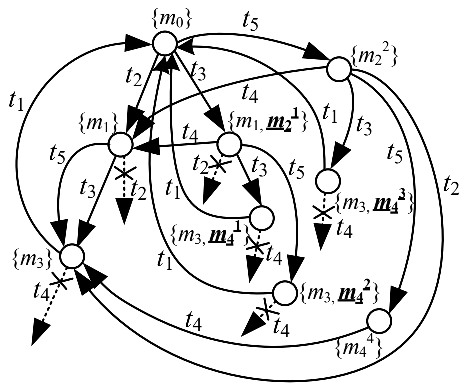


Fig. 5.8 Basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ relative to the supervisor $\mathcal{G}_s$ in Fig. 5.7 and the SD-attack $\mathcal{A}$={(t_2, t_3)}

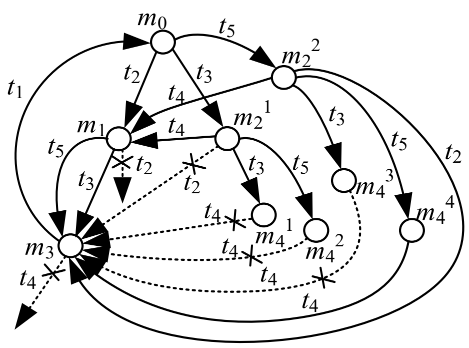


Fig. 5.9 Supervisor $\mathcal{G}_s'$=*GsPruning*($\mathcal{G}_s$, $\mathcal{G}_{s|\mathcal{A}}$, Γ), where $\mathcal{G}_s$ is the supervisor in Fig. 5.5, $\mathcal{G}_{s|\mathcal{A}}$ the basic supervisor in Fig. 5.6, and Γ the corresponding pruning set

Function $\mathcal{G}_s'=GsPruning(\mathcal{G}_s, \mathcal{G}_{s|\mathcal{A}}, \Gamma)$

```
Input: A supervisor $\mathcal{G}_s=(X, E^{\mathcal{G}s}, x_0, l_e^{\mathcal{G}s}, Fb^{\mathcal{G}s})$, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}=(Y, E, y_0, l_e, Fb)$, and the pruning set $\Gamma\neq\varnothing$.
Output: The updated supervisor $\mathcal{G}_s'=(X', E^{\mathcal{G}s}{}', x_0', l_e^{\mathcal{G}s}{}', Fb^{\mathcal{G}s}{}')$.
1)  if $\exists x\in\Gamma_x$, s.t. $x$ is a multi-sequence node in $\mathcal{G}_s$ then
2)        $X_m:=\{x\in\Gamma_x| x$ is a multi-sequence node in $\mathcal{G}_s\}$;
3)        $\mathcal{G}_s:=SeparateNode(\mathcal{G}_s, X_m)$;
4)        $(\mathcal{G}_{s|\mathcal{A}}, \Gamma):=ConstructBasicSG(\mathcal{G}_s, \mathcal{A})$;
5)  end if
6)  for each $(x, y)\in\Gamma$ do
7)      $T_{del}:=T_{out}^{\mathcal{G}s}(x)\cap Fb(y)$;
8)      $Fb^{\mathcal{G}s}(x):=Fb^{\mathcal{G}s}(x)\cup T_{del}$;
9)      delete every output arc of $x$ whose transition label $t\in T_{del}$;
10) end for
11) delete nodes that are not accessible from the initial node $x_0$ and their related arcs
    from $\mathcal{G}_s$ and denote the resulting supervisor graph as $\mathcal{G}_s'$;
12) Output: $\mathcal{G}_s'$.
```

Next, we introduce a function called *GsPruning*. To this aim the following definition is provided.

Definition 5.3: Consider a supervisor $\mathcal{G}_s$, an SD-attack $\mathcal{A}$, and the pruning set Γ output by *ConstructBasicSG*($\mathcal{G}_s$, $\mathcal{A}$). The set of *affected nodes* in $\mathcal{G}_s$ is defined as

$$\Gamma_x=\{x\ |(x, y)\in\Gamma\}.$$

Now, looking at the function *GsPruning*, given a supervisor $\mathcal{G}_s$, it indeed computes a more restrictive supervisor $\mathcal{G}_s'$ than $\mathcal{G}_s$ by removing sequences in $\mathcal{G}_s$ whose firing is possibly forbidden by $\mathcal{G}_{s|\mathcal{A}}$ based on the pruning set Γ. Provided that each affected node $x\in\Gamma_x$ is a single-sequence node in $\mathcal{G}_s$, it can be simply done as in Steps 6-12, which can be explained as follows: For each pair (x, y) in the pruning set Γ, we delete from $\mathcal{G}_s$ the output arcs of node x that are labeled by transitions forbidden to fire at node y in $\mathcal{G}_{s|\mathcal{A}}$, i.e., transitions in $Fb(y)$, and $Fb^{\mathcal{G}s}(x)$ is accordingly updated by including the deleted transitions. Then, inaccessible parts from the initial node x_0 are deleted from $\mathcal{G}_s$, resulting in the supervisor $\mathcal{G}_s'$.

Note that in the case that any of the affected nodes $x \in \Gamma_x$ is a multi-sequence node in $\mathcal{G}_s$, Steps 1-5 are required in advance to update $\mathcal{G}_s$ by separating each multi-sequence affected node $x \in \Gamma_x$ into several single-sequence nodes and then update accordingly the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ and the pruning set Γ.

Example 5.9: Consider the supervisor $\mathcal{G}_s$ in Fig. 5.5 and the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$. As discussed before, calling Function *ConstructBasicSG*($\mathcal{G}_s$, $\mathcal{A}$) we obtain the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ in Fig. 5.6 and the pruning set $\Gamma=\{(m_2, \{m_1, m_2\}), (m_4, \{m_3, m_4\})\}$. Now, let us see how Function $\mathcal{G}_s'$=*GsPruning*($\mathcal{G}_s$, $\mathcal{G}_{s|\mathcal{A}}$, Γ) works. Clearly, the affected nodes m_2 and m_4 in $\mathcal{G}_s$ are both multi-sequence nodes. Thus, we have to separate both of them into several single-sequence nodes, which results in the updated supervisor $\mathcal{G}_s$ in Fig. 5.7. Accordingly, we get the updated basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ in Fig. 5.8 and the pruning set now is $\Gamma=\{(m_2^1, \{m_1, m_2^1\}), (m_4^1, \{m_3, m_4^1\}), (m_4^2, \{m_3, m_4^2\}), (m_4^3, \{m_3, m_4^3\})\}$. Now, we remove sequences in $\mathcal{G}_s$ whose firing is possibly forbidden by $\mathcal{G}_{s|\mathcal{A}}$. As an example, consider the affected node m_2^1. Its output arc with label t_2 should be deleted from $\mathcal{G}_s$ and $Fb^{\mathcal{G}s}(m_2^1)$ is updated to $\{t_2\}$ since $T_{out}^{\mathcal{G}s}(m_2^1) \cap Fb(\{m_1, m_2^1\})=\{t_2\}$. Similarly, the output arcs of nodes m_4^1, m_4^2, m_4^3 with label t_4 are deleted from $\mathcal{G}_s$ and $Fb^{\mathcal{G}s}(m_4^1)=Fb^{\mathcal{G}s}(m_4^2)=Fb^{\mathcal{G}s}(m_4^3)=\{t_4\}$. The resulting supervisor is denoted as $\mathcal{G}_s'$ and depicted in Fig. 5.9. ♦

We note that separating each multi-sequence affected node into several single-sequence nodes is a necessary step to guarantee that only sequences whose firing is possibly forbidden by the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ are removed from the supervisor $\mathcal{G}_s$. To clarify this, suppose that we directly execute Steps 6-12 of Function *GsPruning* in the above example. In other words, we directly handle the supervisor $\mathcal{G}_s$ in Fig. 5.5 based on the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ in Fig. 5.6 and the pruning set $\Gamma=\{(m_2, \{m_1, m_2\}), (m_4, \{m_3, m_4\})\}$. Consider the affected node m_2 of $\mathcal{G}_s$ in Fig. 5.5. We delete its output arc with label t_2 from $\mathcal{G}_s$ and update the corresponding forbidden set as $Fb^{\mathcal{G}s}(m_2)=\{t_2\}$ since $Fb(\{m_1, m_2\})=\{t_2\}$ in $\mathcal{G}_{s|\mathcal{A}}$ in Fig. 5.6. We can see that in this case some sequences whose firing is permitted by $\mathcal{G}_{s|\mathcal{A}}$ are also removed, e.g. the sequence t_5t_2.

Now, we propose Algorithm 5.1 to compute a liveness-enforcing supervisor tolerant to SD-attacks.

Algorithm 5.1: Computation of a liveness-enforcing supervisor tolerant to SD-attacks

Input: A PN system (N, m_0) and an SD-attack $\mathcal{A}$;

Output: A supervisor $\mathcal{G}_s^{\#}$, "*no liveness-enforcing supervisor exists*", or "*failure*".

```
1)  Construct the RG $\mathcal{G}$ of the net system $(N, m_0)$;
2)  $\mathcal{G}_s$:=MaxLiveEnforce($\mathcal{G}$, $T$);
3)  if $\mathcal{G}_s$ is empty then
4)      exit and output "no liveness-enforcing supervisor exists";
5)  end if
6)  ($\mathcal{G}_{s|\mathcal{A}}$, $\Gamma$):=ConstructBasicSG($\mathcal{G}_s$, $\mathcal{A}$);
7)  while $\Gamma\neq\emptyset$ do
8)      $\mathcal{G}_s'$:=GsPruning($\mathcal{G}_s$, $\mathcal{G}_{s|\mathcal{A}}$, $\Gamma$);
9)      $\mathcal{G}_s''$:=MaxLiveEnforce($\mathcal{G}_s'$, $T$);
10)     if $\mathcal{G}_s''=\emptyset$ then
11)         exit and output "failure";
12)     end if
13)     $\mathcal{G}_s$:=$\mathcal{G}_s''$;
```

```
14)    ($\mathcal{G}_{s|\mathcal{A}}$, Γ):=ConstructBasicSG($\mathcal{G}_s$, $\mathcal{A}$);
15) end while
16) $\mathcal{G}_s^{\#}$:=$\mathcal{G}_{s|\mathcal{A}}$;
17) Output:. $\mathcal{G}_s^{\#}$.
```

Let us explain Algorithm 5.1. First, we construct the maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ under no attack. Then, we construct the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ and compute the pruning set Γ. If it is Γ=∅, $\mathcal{G}_{s|\mathcal{A}}$ is the resulting supervisor provided as an output by the algorithm. It is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ by Proposition 5.3. On the contrary, if Γ≠∅, $\mathcal{G}_{s|\mathcal{A}}$ might not guarantee the liveness of the supervised system. In this case, the following steps are executed: we repeatedly compute a more restrictive liveness-enforcing supervisor $\mathcal{G}_s$ under no attack and compute the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, as well as the pruning set Γ. In the case that the pruning set Γ=∅, the iterative procedure stops and the corresponding basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is provided as an output. It is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$.

Now, let us see in more detail the individual steps in the case that Γ≠∅. By Proposition 5.2, Γ≠∅ indicates that $\mathcal{G}_{s|\mathcal{A}}$ leads to a more restrictive behavior of the plant compared to $\mathcal{G}_s$. The supervisor $\mathcal{G}_s$ reflects the real evolution of the controlled system. Thus, we want to check the real evolution of the system supervised by $\mathcal{G}_{s|\mathcal{A}}$ based on $\mathcal{G}_s$. Here, we simply compute a more restrictive supervisor $\mathcal{G}_s'$ than $\mathcal{G}_s$ by removing sequences in $\mathcal{G}_s$ whose firing is possibly forbidden by $\mathcal{G}_{s|\mathcal{A}}$, which is done by computing $\mathcal{G}_s'$=*GsPruning*($\mathcal{G}_s$, $\mathcal{G}_{s|\mathcal{A}}$, Γ). Now, $\mathcal{G}_s'$ is typically not a liveness-enforcing supervisor under no attack. Thus, $\mathcal{G}_s''$=*MaxLiveEnforce*($\mathcal{G}_s'$, *T*) is computed, which is a liveness-enforcing supervisor more restrictive than $\mathcal{G}_s$ under no attack. Then, $\mathcal{G}_s''$ is renamed as

$\mathcal{G}_s$, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ as well as the pruning set Γ are computed again, and the above operations are repeated until $\Gamma=\varnothing$.

We note that, in addition to $\Gamma=\varnothing$, there is another termination condition for the iterative procedure (Steps 7-15), i.e., when we want to get a more restrictive liveness-enforcing supervisor under no attack, the solution turns out to be empty. In this case, Algorithm 5.1 terminates and outputs "*failure*" (see Steps 10-12). Trivially, these two termination conditions ensure that the iterative procedure of Algorithm 5.1 cannot proceed infinitely since the PN system is assumed to be bounded. Besides, we note that the output "*failure*" does not mean that there is no liveness-enforcing supervisor tolerant to the given SD-attack for the given net system. It only implies that Algorithm 5.1 fails to get such a solution although it might exist.

Theorem 5.1: Given a PN system (N, m_0) and an SD-attack $\mathcal{A}$ as the inputs of Algorithm 5.1,

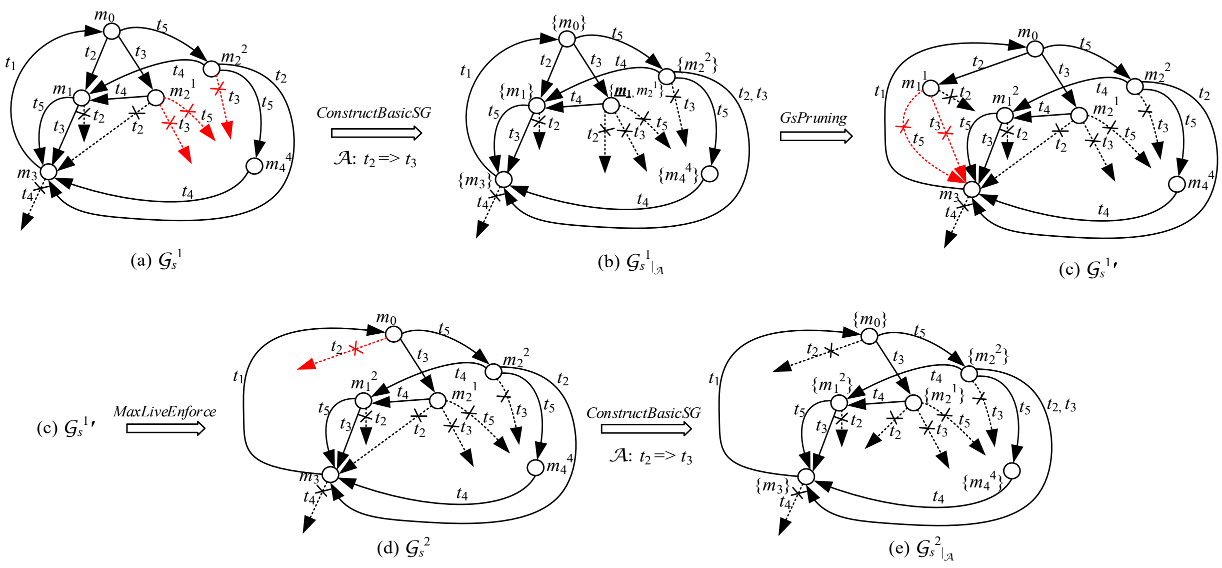
1) no liveness-enforcing supervisor exists for (N, m_0) if Algorithm 5.1 outputs "*no liveness-enforcing supervisor exists*";

2) $\mathcal{G}_s^{\#}$ output by Algorithm 5.1 is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ for (N, m_0).

Proof: 1) Straightforward from Corollary 5.1. 2) Clearly, $\mathcal{G}_s^{\#}$ is actually a basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ and the corresponding pruning set $\Gamma=\varnothing$. Moreover, the supervisor $\mathcal{G}_s$ that generates the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is a liveness-enforcing supervisor for (N, m_0) under no attack. Hence, according to Proposition 5.3, $\mathcal{G}_s^{\#}$ is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ for (N, m_0). ■

Example 5.10: Consider the PN system (N, m_0) in Fig. 5.1 vulnerable to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$. Let us compute a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ for (N, m_0) by Algorithm 5.1. Firstly, the maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ under no attack is computed, as shown in Fig. 5.5. Next, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is constructed as depicted in Fig. 5.6. The pruning set is $\Gamma=\{(m_2, \{m_1, m_2\}), (m_4, \{m_3, m_4\})\}$. Since $\Gamma\neq\emptyset$, we remove sequences in $\mathcal{G}_s$ whose firing is possibly forbidden by $\mathcal{G}_{s|\mathcal{A}}$ by calling *GsPruning*($\mathcal{G}_s$, $\mathcal{G}_{s|\mathcal{A}}$, Γ), resulting in the graph $\mathcal{G}_s'$ in Fig. 5.9. Then, a liveness-enforcing supervisor more restrictive than $\mathcal{G}_s$, can be obtained by calling *MaxLiveEnforce*($\mathcal{G}_s'$, T), which is shown in Fig. 5.10 (a) denoted as $\mathcal{G}_s^1$. Again, we compute the basic supervisor $\mathcal{G}_s^1|_{\mathcal{A}}$ shown in Fig. 5.10 (b) and the pruning set $\Gamma^1=\{(m_1, \{m_1, m_2^1\})\}$. Similarly, we get the graph $\mathcal{G}_s^{1\prime}$ in Fig. 5.10 (c) by calling *GsPruning*($\mathcal{G}_s^1$, $\mathcal{G}_s^1|_{\mathcal{A}}$, Γ^1) and then a liveness-enforcing supervisor more restrictive than $\mathcal{G}_s^1$, denoted as $\mathcal{G}_s^2$ is obtained as shown in Fig. 5.10 (d). Again, the basic supervisor $\mathcal{G}_s^2|_{\mathcal{A}}$ in Fig. 5.10 (e) is computed and the pruning set $\Gamma^2=\emptyset$. Since $\Gamma^2=\emptyset$, the basic supervisor $\mathcal{G}_s^2|_{\mathcal{A}}$ is the output of Algorithm 5.1, which is a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$ for the PN system (N, m_0) in Fig. 5.1.

♦



(a) $\mathcal{G}_s^1$
ConstructBasicSG
$\mathcal{A}$: $t_2 \Rightarrow t_3$
(b) $\mathcal{G}_s^1|_{\mathcal{A}}$
GsPruning
(c) $\mathcal{G}_s^{1\prime}$
(c) $\mathcal{G}_s^{1\prime}$
MaxLiveEnforce
(d) $\mathcal{G}_s^2$
ConstructBasicSG
$\mathcal{A}$: $t_2 \Rightarrow t_3$
(e) $\mathcal{G}_s^2|_{\mathcal{A}}$

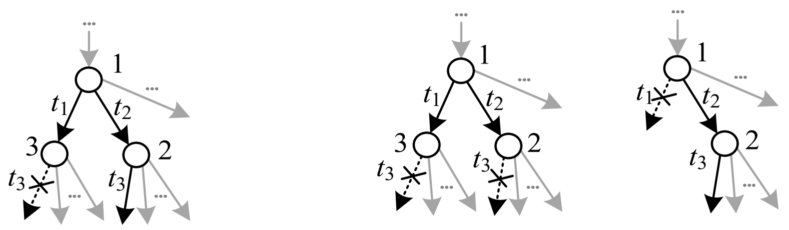
Fig. 5.10 Computation of a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$ for the PN system (N, m_0) in Fig. 5.1

5.4.4 Further Comments

In this subsection, we make some comments on the proposed supervisor synthesis method and provide the reason why we use PNs as a formalism in this work.

1) *Permissiveness*

We note that, given a PN system vulnerable to an SD-attack, there could exist more than one maximally permissive liveness-enforcing supervisor tolerant to the SD-attack. They are incomparable solutions in terms of the language of the controlled system. Let us see an illustrative example. Suppose that Fig. 5.11(a) shows part of a supervisor under no attack. In the case that the system is vulnerable to the SD-attack $\mathcal{A}=\{(t_1, t_2)\}$, it might lead to two maximal solutions, as shown in Fig. 5.11(b). One is permitting the firing of both t_1 and t_2 at node 1. In this case, when t_2 is observed at node 1, the firing of t_3 has to be forbidden since otherwise the bad sequence t_1t_3 may fire if the observation t_2 is produced by the firing of t_1. The second supervisor (on the right) is forbidding the firing of t_1 at node 1. In this case, when t_2 is observed at node 1, we know for sure it is the firing of t_2. Hence, there is no need to forbid the firing of t_3 at node 2. We can see that both of the solutions permit the firing of a sequence that is forbidden by the other solution.



(a) Supervisor under no attack (b) Two incomparable maximal solutions under the SM-attack $\mathcal{A}=\{(t_1, t_2)\}$

Fig. 5.11 Illustration of multiple maximal solutions

Concerning Algorithm 5.1, its solution is not guaranteed to be maximally permissive. In other words, there may exist a liveness-enforcing supervisor tolerant to the given SD-attack for the given net system that is more permissive than the supervisor obtained by Algorithm 5.1. Besides, Algorithm 5.1 might fail to get a solution even when it exists (i.e., “*failure*” is the output). In what follows, we give intuitive explanations for such shortcomings.

We start with a maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ under no attack. In the case that there is an SD-attack $\mathcal{A}$, some different transitions may produce the same observation. We know that a supervisor associates a control action with an observation. Thus, transition sequences producing the same observation correspond to the same control action. Due to the control mechanism, to guarantee that sequences forbidden by $\mathcal{G}_s$ are still forbidden under the attack, a supervisor in the presence of the attack has to forbid all sequences whose observations are the same as those forbidden by $\mathcal{G}_s$. The basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ is such a supervisor and it does not forbid any other sequences. As discussed before, when the pruning set is not empty, the system supervised by $\mathcal{G}_{s|\mathcal{A}}$ is not guaranteed to be live. Thus, the proposed method performs the pruning operation on $\mathcal{G}_s$ and then computes a new liveness-enforcing supervisor under no attack by removing more sequences. It is worth noting that a sequence (e.g., α) removed for enforcing liveness might be a sequence whose existence before has led to some sequences being forbidden due to the same observation. Since sequence α is removed now, those sequences forbidden before due to the existence of sequence α are unnecessarily forbidden. Consequently, after iterative computations, the final solution might not be maximally permissive and it could happen that no solution is found although it does exist.

Consider again Example 5.10. The solution computed by Algorithm 5.1 is not maximally permissive. Look at the computation procedure in Fig. 5.10. When we enforce liveness on $\mathcal{G}_s^{1\prime}$, sequence t_2 is removed (see $\mathcal{G}_s^2$). However, some sequences have been forbidden due to the existence of sequence t_2 before. For example, sequence t_3t_2 has been forbidden. Specifically, because the firing of transition t_2 is forbidden after sequence t_2 in $\mathcal{G}_s$ (Fig. 5.5), considering the existence of the attack $\mathcal{A}=\{(t_2, t_3)\}$, the firing of transition t_2 is also forbidden after sequence t_3 in $\mathcal{G}_s'$ (Fig. 5.9). Now, since sequence t_2 is removed, sequence t_3t_2 is unnecessarily forbidden.

One preliminary remedial idea for getting a maximal solution is that, during iterative computation, once it happens that a sequence whose existence has led to the forbidding of other sequences is removed, we "roll back" to remove the sequence in advance. Then, some sequences can be made up, resulting in a maximal solution. Consider Example 5.10. Since sequence t_2 is eventually removed, we can pre-handle $\mathcal{G}_s$ in Fig. 5.5 by forbidding t_2 in advance and then compute a liveness-enforcing supervisor under no attack as permissive as possible, which is named as $\mathcal{G}_s^*$ shown in Fig. 5.12(a). Starting with supervisor $\mathcal{G}_s^*$ instead of $\mathcal{G}_s$, we perform Algorithm 5.1. After iterative computations, it finally outputs a more permissive solution than the original one. The new solution is shown in Fig. 5.12(b), which is actually maximally permissive.

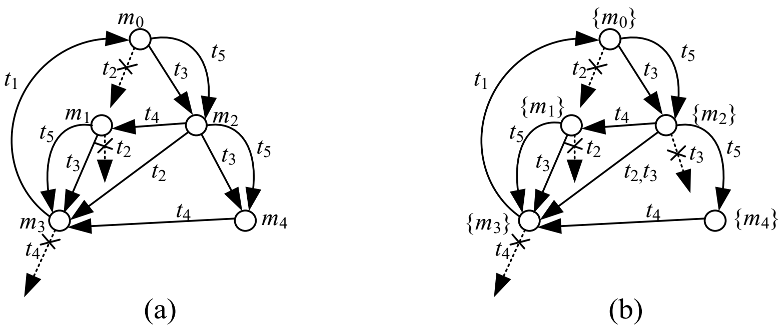


Fig. 5.12 (a) Liveness-enforcing supervisor $\mathcal{G}_s^*$ under no attack; and (b) Maximally permissive liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}=\{(t_2, t_3)\}$

How to formalize the idea will be investigated in our future work so that an improved approach is developed that is capable of getting one (among possibly several) maximally permissive liveness-enforcing supervisor tolerant to a given SD-attack.

2) *Computational Complexity*

We analyze the complexity of Algorithm 5.1. It is clear that the first step, consisting in the construction of the RG of the given PN system, has exponential complexity with the net size (i.e., the number of places, transitions, and tokens initially in the net). Next, we focus on functions *MaxLiveEnforce*, *ConstructBasicSG*, and *GsPruning* in Algorithm 5.1.

-*MaxLiveEnforce* computes a liveness-enforcing supervisor under no attack when given a supervisor under no attack. It is of polynomial complexity with the size of the given supervisor since function *Tarjan* used in it is known to be polynomial in the size of the given supervisor [91].

-*ConstructBasicSG* computes a basic supervisor when given a liveness-enforcing supervisor under no attack. Suppose that the given supervisor has a nodes. In the worst case, the computed basic supervisor contains no more than 2^a nodes since every node in the basic supervisor consists of nodes in the given supervisor under no attack.

-*GsPruning* prunes sequences in a supervisor under no attack. It involves functions *SeparateNode* and *ConstructBasicSG* when affected nodes in the considered supervisor are multi-sequence nodes. *SeparateNode* separates each multi-sequence affected node into several single-sequence nodes. In the worst case, each node has to be separated into single-sequence nodes but the number of nodes in the resulting supervisor by *SeparateNode* cannot be more than $b=1+|T|+|T|^2+\ldots+|T|^n$, where n is the length of the longest elementary sequence of a node in the considered supervisor. Then, the computed basic supervisor contains no more than 2^b nodes. The following pruning procedure is straightforward.

Algorithm 5.1 is basically an iterative procedure of calling the above three functions after constructing the RG. In the worst case, the number of iterations cannot be more than the size (i.e., the number of nodes and arcs) of a graph derived by separating each multi-sequence node in the RG into single-sequence nodes. Consequently, based on the above analysis, the overall complexity of Algorithm 5.1 is exponential with the size of the RG constructed in the first step and is also obviously exponential with the size of the handled PN. Thus, it could happen that Algorithm 5.1 can hardly obtain a solution in a reasonable time if the handled PN has a very large size.

3) *About PN formalism*

Problem 5.1 may be formulated in the formalism of *finite-state automata* (FSA). The property of *liveness* for PNs can be transformed into an equivalent specification on the behavior of FSA. However, to our best knowledge, regardless of the modelling tools, the problem studied in this chapter has never been investigated in the literature. The advantages of using the formalism of PNs are as follows. First, PNs could be promising on investigating the problem in more general cases in the future work. In the case of partial observation, we may modify the approach in order to use the basis reachability graph instead of the reachability graph, thus avoiding exhaustive enumeration [11]. Moreover, we may extend the proposed approach to unbounded systems, which can be modelled by a PN but not by an FSA. When dealing with an unbounded PN system, we may construct the coverability graph or some special finite reachability graphs/trees [98, 104]. Thus, the proposed approach can be adapted to get a solution by handling such a graph. Second, we could have a PN model of the system that is used for the solutions of other problems where the PN features are greatly important, e.g., fault diagnosis or many other problems that can be solved using structural analysis. Finally, the PN formalism allows us to consider if the liveness property can be transformed into a state specification in terms of GMECs for some particular class of PNs. If so, a tolerant *monitor-based* supervisor is possibly designed.

5.5 Study Cases

In this section, we present two study cases where the proposed approach is applied.

5.5.1 Study Case 1

An automatic control system is designed for playing a game on chemical reactions, where chemical equations (1)-(5) are involved.

$$Na_2CO_3 + CO_2 + H_2O = 2NaHCO_3 \qquad (1)$$

$$NaHCO_3 + NaOH = Na_2CO_3 + H_2O \qquad (2)$$

$$NaHCO_3 + HCl = NaCl + CO_2\uparrow + H_2O \qquad (3)$$

$$CO_2 + 2NaOH = Na_2CO_3 + H_2O \tag{4}$$

$$NaHCO_3 + HNO_3 = NaNO_3 + CO_2\uparrow + H_2O \tag{5}$$

Suppose that NaOH, HCl, HNO_3 and H_2O are sufficiently provided and the chemical transformation among Na_2CO_3, CO_2, and $NaHCO_3$ is the focus of the game. Consequently, the chemical reactions can be modelled by the PN in Fig. 5.1, where p_1-p_3 represent $NaHCO_3$, Na_2CO_3 and CO_2, respectively, and t_1-t_5 represent the chemical reactions (1)-(5), respectively. To be intuitive, the PN is redrawn in Fig. 5.13, where the meanings of places and the needed substances triggering chemical transformation are annotated. Place p_1 contains two tokens in the initial marking, which models the initial condition that two copies of $NaHCO_3$ are provided. The control specification on the system requires that any of the five chemical equations (1)-(5) can always be performed after finite times of other chemical reactions. This exactly corresponds to the *liveness* specification on the PN system in Fig. 5.13.

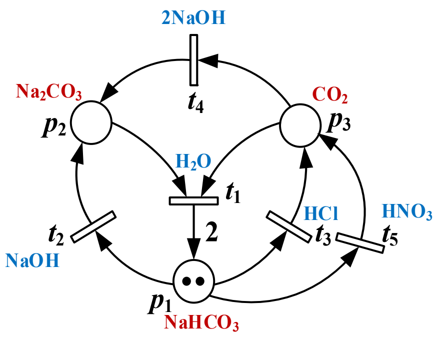


Fig. 5.13 PN system (N, m_0) modelled for chemical reactions

The supervisor in the closed-loop control system observes the occurrence of chemical reactions by receiving signals from sensors and gives control actions based on observations. The supervisor communicates with sensors/actuators via communication networks, which makes the system possibly suffer from the intrusion of malicious agents. Suppose that we have the prior knowledge that the sensor signals produced by the chemical equation (2) are prone to be disguised as the sensor signals produced by the chemical equation (3) in related sensor communication channels by an intruder. We can see that it is exactly the SD-attack $\mathcal{A}$={(t_2, t_3)}. Recalling Example 5.10, we may design a liveness-enforcing supervisor tolerant to the attack $\mathcal{A}$ using the proposed approach (Algorithm 5.1), which is shown in Fig. 5.10(e).

5.5.2 Study Case 2

Consider a flexible manufacturing system (FMS). It contains two types of robotic arms r_1 and r_2 for processing parts and each type has two copies. Raw parts may enter the FMS via loading buffers I1 and I2 and finished products may leave the FMS via unloading buffer O1 and O2. Two types of products P1 and P2 are produced by the FMS. Their production routes are:

P1: I1→r_1→ 2r_2→O1;

P2: I2→r_2→ 2r_1→O2.

In words, producing type P1 requires first one robotic arm r_1 and then two robotic arms r_2, whereas producing type P2 requires first one robotic arm r_2 and then two robotic arms r_1. The loading buffers I1 and I2 both can contain at most three raw parts. Moreover, it is assumed that when a finished product leaves the FMS, a new raw part is added to the corresponding loading buffer. Consequently, in the case that the loading buffers I1 and I2 are fully loaded, the FMS can be modelled by the PN system shown in Fig. 5.14.

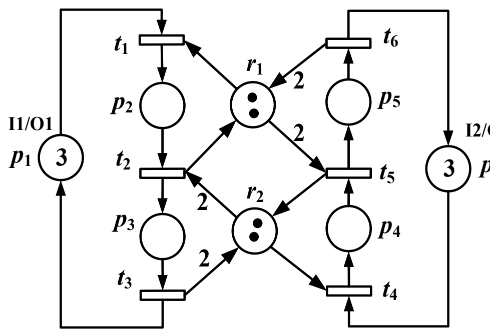


Fig. 5.14 PN system (N, m_0) modelled for an FMS

Typically, it is required that any production route in an FMS cannot be blocked. This actually corresponds to the liveness specification on the PN model in Fig. 5.14. Note that the system in Fig. 5.14 now is not live. For example, neither of two production routes can continue in the reachable state where two robotic arms r_1 are occupied for processing P1-type raw parts and two robotic arms r_2 are occupied for processing P2-type raw parts.

When computing a liveness-enforcing supervisor, we need to consider the existence of network attacks in the scenario that sensors and actuators are used respectively to detect and actuate the occurrence of transitions and the supervisor communicates with sensors/actuators via communication networks. Suppose that we have the prior knowledge that the sensor signals produced by the occurrence of transition t_1 are prone to be disguised as those of transition t_2 and the sensor signals produced by the occurrence of transition t_5 are prone to be disguised as those of transition t_4. In other words, we need to consider the SD-attack $\mathcal{A}=\{(t_1, t_2), (t_5, t_4)\}$. Hence, we may compute a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$ by Algorithm 5.1. Firstly, the maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ under no attack is computed, as shown in Fig. 5.15. Note that the concrete markings are omitted here. Next, considering the attack $\mathcal{A}=\{(t_1, t_2), (t_5, t_4)\}$, we obtain the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, as depicted in Fig. 5.16. It is worth noting that the pruning set Γ, which is computed together with $\mathcal{G}_{s|\mathcal{A}}$, is empty. Hence, the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ in Fig. 5.16 is exactly a solution, i.e., a liveness-enforcing supervisor tolerant to the SD-attack $\mathcal{A}$. Moreover, it is trivial to see that the solution is maximally permissive.

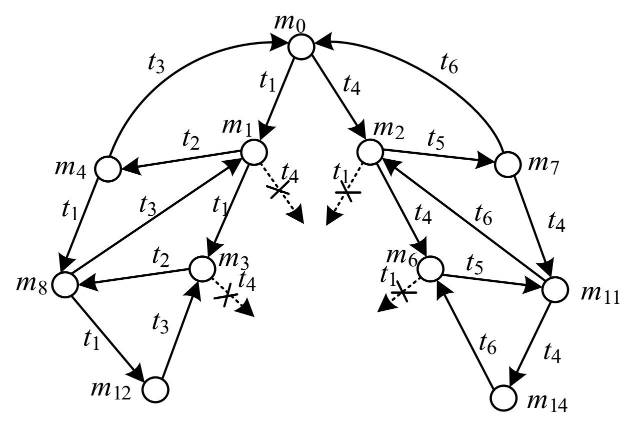


Fig. 5.15 Maximally permissive liveness-enforcing supervisor $\mathcal{G}_s$ of the PN system in Fig. 5.14 under no attack

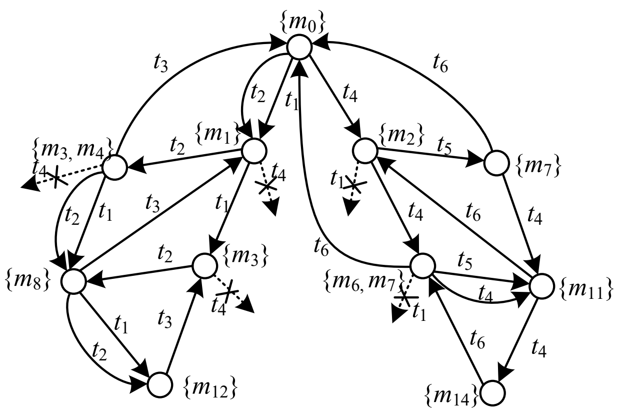


Fig. 5.16 Basic supervisor $\mathcal{G}_{s|\mathcal{A}}$ relative to the supervisor $\mathcal{G}_s$ in Fig. 5.15 and the SD-attack $\mathcal{A}=\{(t_1, t_2), (t_5, t_4)\}$

5.6 Conclusions

This chapter investigates the problem of synthesizing a liveness-enforcing supervisor tolerant to sensor-reading disguising attacks (SD-attacks) in a closed-loop control system, with the plant modelled as a bounded PN system. In particular, given a supervisor $\mathcal{G}_s$ under no attack and an SD-attack $\mathcal{A}$, we propose an approach that constructs the basic supervisor $\mathcal{G}_{s|\mathcal{A}}$, which guarantees that the behavior forbidden by $\mathcal{G}_s$ can never occur in the controlled system vulnerable to the SD-attack $\mathcal{A}$, but possibly restricts the behavior of the plant more than $\mathcal{G}_s$. Based on basic supervisors, we develop a method that computes a liveness-enforcing supervisor tolerant to SD-attacks.

The work of this chapter is now accepted by *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

CHAPTER VI

Verification of Fault-predictability

6.1 Introduction

Starting from this chapter, we enter the second topic of this thesis, i.e., analysis of partially-observed DES. In this chapter, we verify the *fault-predictability* of such DES and in the next chapter, we propose a new property called *event-based opacity* and study its verification in partially-observed DES.

A fault-predictable DES is a system where any fault can be correctly predicted before its occurrence. In this chapter, we study the verification of fault-predictability in bounded and unbounded DESs modeled by *labeled* PNs. An approach based on the construction of a *Predictor Net* and a *Predictor Graph* is proposed. In particular, a necessary and sufficient condition for fault-predictability is derived by characterizing the structure of the Predictor Graph. Furthermore, two rules are proposed to reduce the size of a given PN, which allow one to analyze the fault-predictability of the original net by verifying the fault-predictability of the reduced net.

This chapter is organized as follows. Section 6.2 introduces the background on labeled PNs. Section 6.3 formulates the notion of fault-predictability in the framework of labeled PNs. The verification of fault-predictability is investigated in Section 6.4. Section 6.5 concludes this chapter.

We notice that the work presented in this chapter has been published in *IEEE Transactions on Automatic Control*; see [125].

6.2 Background on Labeled PN

A *labeled* PN system is a triple (N, m_0, l), where (N, m_0) is a PN system and l is a function l: $T\rightarrow\Sigma\cup\{\varepsilon\}$ that associates with each transition, either a symbol from a given alphabet Σ, or the empty string ε. In this framework, the associated symbol of a transition decides if or not it is observable. Specifically, it is $T=T_o\cup T_{uo}$, where T_o is the set of transitions with their labels being a symbol in Σ, defining *observable transitions*, and T_{uo} is the set of transitions with their labels being the empty string, defining *unobservable transitions*. Furthermore, the function l can be extended as l: $T^*\rightarrow\Sigma^*$ such that (a) $l(\varepsilon)=\varepsilon$; and (b) for any $\alpha\in T^*$ and $t\in T$, $l(\alpha t)=l(\alpha)l(t)$. We usually use w to denote the label sequence that corresponds to a transition sequence $\alpha\in T^*$, i.e., $w=l(\alpha)$ and call w the *observed word* of α. Also, we define the inverse function l^{-1}: $\Sigma^*\rightarrow 2^{T^*}$ such that for any $w\in\Sigma^*$, $l^{-1}(w)=\{\alpha\in L(N, m_0)|\ l(\alpha)=w\}$. We use $l(L(N, m_0))$ to denote the set of words generated by transition sequences of N that are enabled at m_0, i.e., $l(L(N, m_0))=\{l(\alpha)|\ \alpha\in L(N, m_0)\}$, called the *language* of the labeled PN (N, m_0, l).

We note that in this chapter, when no ambiguity occurs, a PN refers to a PN system and we simply write L instead of $L(N, m_0)$. Besides, we modify the notion of *prefixes* of a transition sequence such that the set of all prefixes of a sequence excludes the sequence itself. In more detail, given a sequence $\alpha\in T^*$, we denote the set of all prefixes of α as $pr(\alpha)$ such that $pr(\alpha)=\{\alpha_1\in T^*|\ \exists\alpha_2\in T^*\backslash\{\varepsilon\}$ such that $\alpha=\alpha_1\alpha_2\}$. Clearly, $\alpha\notin pr(\alpha)$. Moreover, we may view a transition sequence as a set of transitions. We use $t\in\alpha$ ($t\notin\alpha$) to indicate that the transition t appears (does not appear) in the sequence α. Furthermore, given a set of transitions $T'\subseteq T$, we use $\alpha\cap T'\neq\emptyset$ to indicate that $\exists t\in T'$ such that $t\in\alpha$, and $\alpha\cap T'=\emptyset$ indicate that $\forall t\in T'$, $t\notin\alpha$.

6.3 Fault-predictability of Labeled PN

We consider a labeled PN with fault transitions, under the following two assumptions:

*A*1: The PN does not enter a deadlock before the occurrence of a fault.

*A*2: The PN suffers from a single class of faults.

Assumption A1 is a technical assumption and its importance will appear in the following. It is the counterpart of the assumption typically done when performing diagnosability analysis: the system does not enter a deadlock after the firing of any fault transition. Assumption A2 means that we are not interested in distinguishing among fault transitions. Therefore, transitions could be partitioned into two subsets, i.e., $T=T_N\cup T_F$, where T_N is the set of normal (non-fault) transitions, and T_F is the set of fault transitions.

Typically in the framework of fault diagnosis, fault transitions are modeled as unobservable transitions and the goal is to detect their occurrence. In this work, we do not make such an assumption since our goal is to predict the occurrence of a fault before its occurrence. Therefore, the proposed approach applies regardless of the fact that fault transitions are observable or not.

Before presenting the definition of fault-predictability, we introduce three kinds of transition sequences. Given a sequence $\sigma\in L$, we call σ

- a *normal sequence* if $\sigma\in T_N^*$;
- a *fault-reached sequence* if σ ends with a fault transition $f\in T_F$;
- a *fault-end sequence* if $\sigma=\sigma' f$, where $\sigma'\in T_N^*$ and $f\in T_F$.

Therefore, a fault-end sequence is a fault-reached sequence where all of its transitions before the last one are normal transitions. We use $\Psi(T_F)$ to denote the set of fault-reached sequences and $\Theta(T_F)$ the set of fault-end sequences.

Let us now introduce the definition of fault-predictability, which is the PN counterpart of the fault-predictability definition given by Genc *et al.* [33] for regular languages.

Definition 6.1: A labeled PN (N, m_0, l) is said to be *fault-predictable* w.r.t. a set of fault transitions T_F if

$$(\forall\alpha\in\Psi(T_F))(\exists\beta\in pr(\alpha):\beta\cap T_F=\varnothing)[\mathbf{A}],$$

where $\mathbf{A}$: $(\forall\theta\in L:(l(\theta)=l(\beta))\wedge(\theta\cap T_F=\varnothing))$
$(\exists K\in\mathbb{N})(\forall\theta\theta'\in L)\ [(|\theta'|>K)\Rightarrow(\theta'\cap T_F\neq\varnothing)]$,

otherwise it is said to be *fault-unpredictable* w.r.t. T_F.

In words, a labeled PN is fault-predictable if for any fault-reached sequence, there exists at least one of its normal prefixes whose observation can be used to set a fault alarm. In more detail, once the word corresponding to this prefix is observed, a fault alarm can be launched implying that a fault will definitely occur within a finite number of event occurrences, which is indeed the true situation for the PN. On the other hand, a labeled PN is fault-unpredictable if there exists a fault-reached sequence, e.g. α, such that for any of its normal prefixes, e.g. β, there exists a normal sequence θ producing the same observation of β that may either be continued infinitely never including a fault, or enter a deadlock. However, for sake of simplicity in the formulation of the following results, the latter case is excluded by Assumption A1. In other words, none of the normal prefixes of α produces an observation that can be used to set a fault alarm.

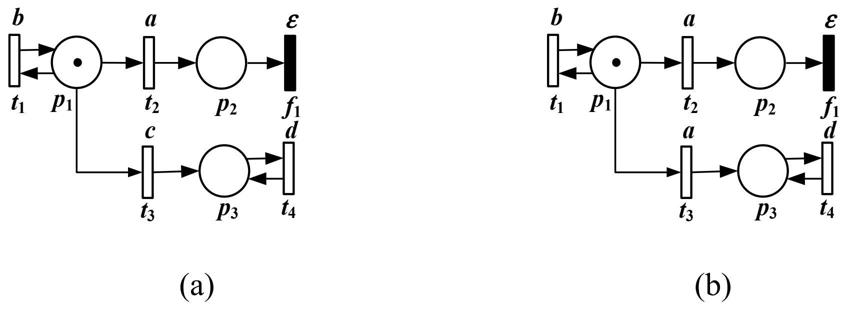


Fig. 6.1 Two labeled PNs: (a) a fault-predictable labeled PN and
(b) a fault-unpredictable labeled PN

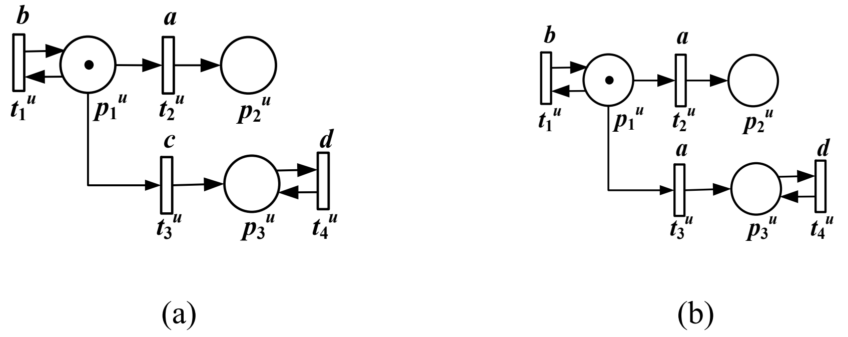


Fig. 6.2 Unfaulty nets of the labeled PNs in Fig. 6.1

Example 6.1: Consider the labeled PN in Fig. 6.1(a), where $T_F=\{f_1\}$ and $T_N=\{t_1\text{-}t_4\}$. Transition f_1 is unobservable, while the other transitions are observable. Clearly, for any $\alpha\in\Psi(T_F)$, α is in the form $(t_1)^i t_2 f_1$, $i\in\mathbb{N}$ and we can accordingly find a normal prefix of α, namely, $\beta=(t_1)^i t_2$ such that by looking at the observed word $l(\beta)=b^i a$, we know for sure that f_1 will occur within the following one step. Hence, the labeled PN is fault-predictable.

Consider now the labeled PN in Fig. 6.1(b), which is the same as that in Fig. 6.1(a) with the only difference that t_3 is labeled "a". Let us consider the fault-reached sequence $t_1t_2f_1$. Its set of prefixes is $pr(t_1t_2f_1)=\{\varepsilon, t_1, t_1t_2\}$. Let us focus on t_1t_2. There exists another sequence $t_1t_3\in L$, which produces the same observation, i.e., $l(t_1t_3)=l(t_1t_2)=ba$ and transition t_4 may fire an arbitrarily large number of times after t_1t_3. In other words, when we observe ba, it is possible that no fault will occur in the following steps, i.e., ba cannot be used to set a fault alarm, claiming that a fault will surely happen after the occurrence of a finite number of events. Similarly, none of the prefixes of $t_1t_2f_1$ corresponds to an observation that can be used to set a fault alarm. Hence, the labeled PN in Fig. 6.1(b) is fault-unpredictable. ♦

A greedy approach to determine whether a labeled PN is fault-predictable, clearly consists in considering all the normal prefixes of all the fault-reached sequences, and check if **A** is satisfied. In the following, we provide an alternative approach that only requires the examination of the longest prefixes of all the fault-end sequences. It is based on the following proposition, which claims that we can determine that a labeled PN is fault-unpredictable if we can find a fault-end sequence σf and a normal sequence θ with $l(\theta)=l(\sigma)$ such that an arbitrarily long normal sequence can occur following θ. Otherwise, we conclude that the labeled PN is fault-predictable.

Proposition 6.1: A labeled PN (N, m_0, l) is fault-unpredictable w.r.t. a set of fault transitions T_F *iff*

$$(\exists \sigma f\in\Theta(T_F):\sigma\in T_N^{*}\wedge f\in T_F)$$
$$(\exists\theta\in L:(l(\theta)=l(\sigma))\wedge(\theta\cap T_F=\varnothing))$$
$$(\forall K\in\mathbf{N})(\exists\theta\theta'\in L)[(|\theta'|>K)\wedge(\theta'\cap T_F=\varnothing)].$$

Proof: (=>) It is clear that $\sigma f \in \Psi(T_F)$ since $\Theta(T_F) \subseteq \Psi(T_F)$. Let $\beta \in pr(\sigma)$. Since $l(\theta)=l(\sigma)$, we can see that there exists $\theta_1 \in pr(\theta)$ such that $l(\theta_1)=l(\beta)$. Let $\theta=\theta_1\theta_2$. Clearly, $\theta_2 \cap T_F=\varnothing$. Since $(\forall K \in \mathbf{N})(\exists \theta\theta' \in L)[(|\theta'| > K) \wedge (\theta' \cap T_F = \varnothing)]$, it holds that $(\forall K \in \mathbf{N})$ $(\exists \theta_1\theta_2\theta' \in L)[(|\theta_2\theta'| > K) \wedge (\theta_2\theta' \cap T_F = \varnothing)]$. In other words, $(\exists \sigma f \in \Psi(T_F))(\forall \beta \in pr(\sigma f))[\overline{\mathbf{A}}]$, where $\overline{\mathbf{A}}$ denotes the opposite of $\mathbf{A}$ in Definition 6.1. By Definition 6.1, the labeled PN (N, m_0, l) is fault-unpredictable w.r.t. T_F.

(<=) Since (N, m_0, l) is fault-unpredictable w.r.t. T_F, it holds that $(\exists \alpha \in \Psi(T_F))(\forall \beta \in pr(\alpha) : \beta \cap T_F = \varnothing)[\overline{\mathbf{A}}]$ by Definition 6.1. Clearly, there exists $\sigma f \in pr(\alpha)$ or $\sigma f=\alpha$ such that $\sigma f \in \Theta(T_F)$. It is trivial to see that $(\forall \beta \in pr(\sigma f))[\overline{\mathbf{A}}]$. Hence, for the case $\beta=\sigma$, it holds $\overline{\mathbf{A}}$. Specifically, due to Assumption A1, it holds:

$$(\exists \theta \in L : (l(\theta) = l(\sigma)) \wedge (\theta \cap T_F = \varnothing))$$
$$(\forall K \in \mathbf{N})(\exists \theta\theta' \in L)[(|\theta'| > K) \wedge (\theta' \cap T_F = \varnothing)].$$ ■

6.4 Verification of Fault-predictability

In this section, we first propose a method to verify fault-predictability of labeled PNs based on a special net and a special graph, called respectively, *Predictor Net* and *Predictor Graph*. Then we show how some rules can be applied to reduce the computational complexity of the analysis.

6.4.1 Predictor Net

Let us introduce a preliminary definition. Given a labeled PN (N, m_0, l), its *unfaulty net*, denoted by (N^u, m_0^u, l^u), is a net derived from (N, m_0) by deleting all fault transitions as well as their related arcs, and the labeling function l^u coincides with l restricted to the remaining transitions, i.e., transitions in T_N.

Example 6.2: The nets in Fig. 6.2(a) and (b) are the unfaulty nets of the labeled PNs in Fig. 6.1(a) and (b), respectively. To distinguish transitions and places in unfaulty nets from those in the original nets, we use the superscript "u" to mark transitions and places in unfaulty nets. ♦

We now define the Predictor Net of a labeled PN.

Definition 6.2: Let (N, m_0, l) be a labeled PN and (N^u, m_0^u, l^u) its unfaulty net, where $N=(P, T_N \cup T_F, F, W)$ and $N^u=(P^u, T^u, F^u, W^u)$. The *Predictor Net* $(\mathcal{N}, \mathcal{m}_0)$, where $\mathcal{N}=(\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{W})$, is defined as follows:

1. $\mathcal{P}=P \cup P^u$;

2. $\mathcal{m}_0=[m_0^T, (m_0^u)^T]^T$;

3.

a) For any $t \in T_N$ and $t^u \in T^u$ with $l(t)=l^u(t^u) \neq \varepsilon$, let $(t, t^u) \in \mathcal{T}$ and

--for any $(p, t) \in F$, let $(p, (t, t^u)) \in \mathcal{F}$ and $\mathcal{W}(p, (t, t^u))=W(p, t)$;

--for any $(t, p) \in F$, let $((t, t^u), p) \in \mathcal{F}$ and $\mathcal{W}((t, t^u), p)=W(t, p)$;

--for any $(p^u, t^u) \in F^u$, let $(p^u, (t, t^u)) \in \mathcal{F}$ and $\mathcal{W}(p^u, (t, t^u))=W^u(p^u, t^u)$;

--for any $(t^u, p^u) \in F^u$, let $((t, t^u), p^u) \in \mathcal{F}$ and $\mathcal{W}((t, t^u), p^u)= W^u(t^u, p^u)$;

b) For any $t \in T_N$ with $l(t)=\varepsilon$, let $(t, \varepsilon) \in \mathcal{T}$ and

--for any $(p, t) \in F$, let $(p, (t, \varepsilon)) \in \mathcal{F}$ and $\mathcal{W}(p, (t, \varepsilon))= W(p, t)$;

--for any $(t, p) \in F$, let $((t, \varepsilon), p) \in \mathcal{F}$ and $\mathcal{W}((t, \varepsilon), p)=W(t, p)$;

c) For any $t^u \in T^u$ with $l^u(t^u)=\varepsilon$, let $(\varepsilon, t^u) \in \mathcal{T}$ and

--for any $(p^u, t^u) \in F^u$, let $(p^u, (\varepsilon, t^u)) \in \mathcal{F}$ and $\mathcal{W}(p^u, (\varepsilon, t^u))=W^u(p^u, t^u)$;

--for any $(t^u, p^u) \in F^u$, let $((\varepsilon, t^u), p^u) \in \mathcal{F}$ and $\mathcal{W}((\varepsilon, t^u), p^u)= W^u(t^u, p^u)$;

d) For any $f \in T_F$, let $(f, \varepsilon) \in \mathcal{T}$ and

--for any $(p, f) \in F$, let $(p, (f, \varepsilon)) \in \mathcal{F}$ and $\mathcal{W}(p, (f, \varepsilon))=W(p, f)$;

--for any $(f, p) \in F$, let $((f, \varepsilon), p) \in \mathcal{F}$ and $\mathcal{W}((f, \varepsilon), p)=W(f, p)$;

e) For any $t^u \in T^u$, let $t^u \in \mathcal{T}$ and for any arc $(x, y) \in F^u$, let $(x, y) \in \mathcal{F}$ with $\mathcal{W}(x, y) = W^u(x, y)$.

In simple words, the Predictor Net is derived in two main steps.

1) The original labeled PN and its unfaulty net are combined by synchronizing transitions in the two nets according to their labels. In more detail, every two observable normal transitions with the same label are synchronized (item a in Definition 6.2), while each unobservable transition and each fault transition is synchronized with an empty transition (items b, c and d in Definition 6.2).

2) All transitions in the unfaulty net are additionally connected to the above combination net via exactly the same arcs in the unfaulty net (item e in Definition 6.2). As a result, the Predictor Net can be viewed as the composition of two nets. The first net is the parallel composition of the original labeled PN and its unfaulty net, where the synchronization is done on the labels. The second net is the unfaulty net. These two nets share places from the unfaulty net.

Based on the above Definition 6.2, we can partition transitions in a Predictor Net $\mathcal{N}$ as $\mathcal{T} = \mathcal{T}^2 \cup \mathcal{T}^1$, where $\mathcal{T}^2$ denotes the set of transitions that are in the form of pairs of transitions and $\mathcal{T}^1$ denotes the set of transitions that are exactly a single transition from the unfaulty net. Finally, we use $\mathcal{T}^f \subseteq \mathcal{T}^2$ to denote the set of pairs of transitions containing a fault transition as a component.

Example 6.3: The net in Fig. 6.3 is the Predictor Net of the labeled PN in Fig. 6.1(a) and the net in Fig. 6.4 is the Predictor Net of the labeled PN in Fig. 6.1(b). In both cases, transitions in $\mathcal{T}^1$ are denoted with biased bars. In the Predictor Net in Fig. 6.3, it is $\mathcal{T}^1 = \{t_1^u, t_2^u, t_3^u, t_4^u\}$, $\mathcal{T}^2 = \{(t_1, t_1^u), (t_2, t_2^u), (t_3, t_3^u), (t_4, t_4^u), (f_1, \varepsilon^u)\}$ and $\mathcal{T}^f = \{(f_1, \varepsilon^u)\}$. In the Predictor Net in Fig. 6.4, it is: $\mathcal{T}^1 = \{t_1^u, t_2^u, t_3^u, t_4^u\}$, $\mathcal{T}^2 = \{(t_1, t_1^u), (t_2, t_2^u), (t_3, t_3^u), (t_4, t_4^u), (f_1, \varepsilon^u), (t_2, t_3^u), (t_3, t_2^u)\}$ and $\mathcal{T}^f = \{(f_1, \varepsilon^u)\}$. ♦

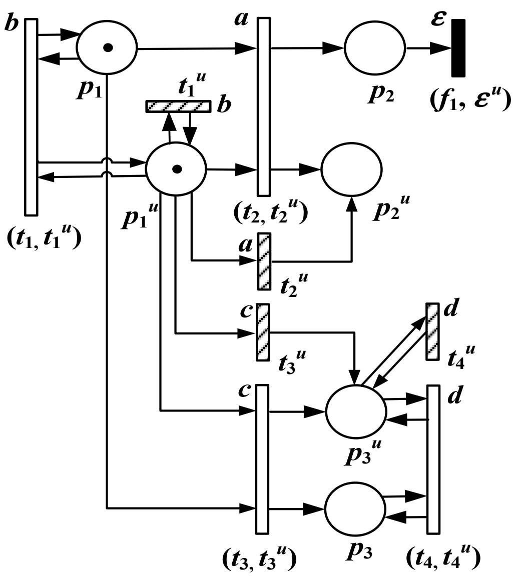


Fig. 6.3 Predictor Net of the labeled PN in Fig. 6.1(a)

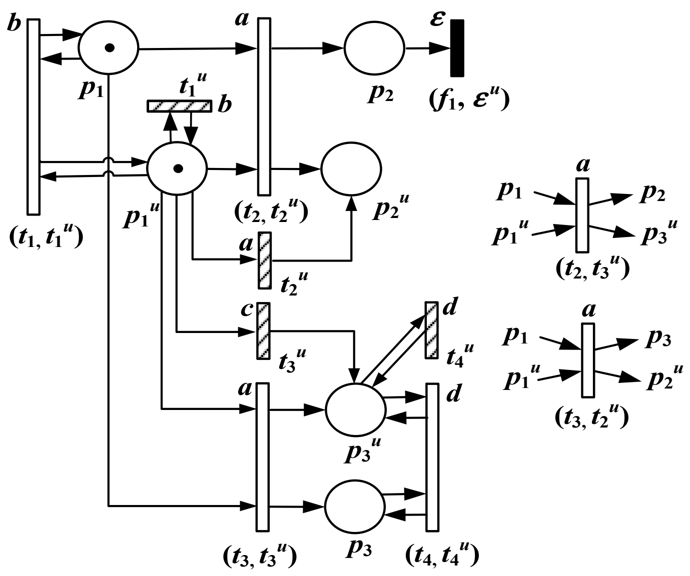


Fig. 6.4 Predictor Net of the labeled PN in Fig. 6.1(b)

Remark 6.1: The Predictor Net has strong similarities with the Verifier Net proposed by Cabasino *et al.* [10] used to verify the fault-diagnosability of a PN. However, the Verifier Net is basically the concurrent composition of the original net and the unfaulty net, where synchronization is done on the labels associated with transitions. In contrast, the Predictor Net also contains a replication of the unfaulty net. ♣

The following proposition reveals the relationship among sequences in the Predictor Net, the original net and the unfaulty net.

Proposition 6.2: Consider a labeled PN (N, m_0, l), its unfaulty net (N^u, m_0^u, l^u) and its Predictor Net $(\mathcal{N}, m_0)$,

1) for any sequence $\sigma=(t_{i1},t_{j1}{}^u)(t_{i2},t_{j2}{}^u)...(t_{ik},t_{jk}{}^u)\in L(\mathcal{N}, m_0)\cap(\mathcal{T}^2)^*$, it holds that $\sigma_1=t_{i1}t_{i2}\dots t_{ik}\in L(N, m_0)$, $\sigma_2=t_{j1}{}^u t_{j2}{}^u\dots t_{jk}{}^u\in L(N^u, m_0{}^u)$ and $l(\sigma_1)=l^u(\sigma_2)$; and

2) for any two sequences $\sigma_1=t_{i1}t_{i2}\dots t_{ik}\in L(N, m_0)\cap T_N{}^*$ and $\sigma_2=t_{j1}{}^u t_{j2}{}^u\dots t_{jk}{}^u\in L(N^u, m_0{}^u)$ with $l(t_{iq})=l^u(t_{jq}{}^u)$, $\forall q\in\{1, 2, \dots, k\}$, it holds that $\sigma=(t_{i1}, t_{j1}{}^u)(t_{i2}, t_{j2}{}^u)\dots(t_{ik}, t_{jk}{}^u)\in L(\mathcal{N}, m_0)\cap(\mathcal{T}^2)^*$.

Proof: Trivially follows from Definition 6.2. ■

6.4.2 Predictor Graph

Before introducing the formal definition of Predictor Graph, we provide some preliminary notation. Given a path π in a graph, we use $X(\pi)$ to denote the set of nodes in π. We say that a node x_1 is *accessible* from another node x_2 if there exists a path from x_2 to x_1. By default, every node is accessible from itself. Furthermore, we say that a cycle c is *accessible* from a node x if $\exists x'\in X(c)$ such that x' is accessible from x.

The *Predictor Graph* (PG) is similar to the reachability graph or the coverability graph [69] of the Predictor Net but with some modifications clearly highlighted in the following. Furthermore, some nodes could be tagged "dangerous". Algorithm 6.1 provides the main steps for its construction. Note that, analogously to the coverability graph, in the case of unbounded nets, a node in PG may correspond to a special marking where some entries coincide with the symbol *inf*, implying that tokens can grow infinitely in the corresponding places. We assume that $\forall k\in\mathbb{N}$, $inf>k$ and it holds that $inf\pm k=inf$.

Algorithm 6.1: Construction of the Predictor Graph (PG)

Input: A labeled PN (N, m_0, l);

Output: The PG of (N, m_0, l).

```
1.  Generate the Predictor Net ($\mathcal{N}$, $\mathcal{m}_0$) according to Definition 6.2;
2.  create the root node $x_0$ with the initial marking $\mathcal{m}_0$ and tag it “new”;
3.  while there exists a node $x$ tagged “new” do
4.      let $\mathcal{m}$ be the marking of $x$;
5.      if $\exists t \in \mathcal{T}^f$ such that $t$ is enabled at $\mathcal{m}$ then
6.          tag node $x$ “dangerous”;
7.          execute SonNodes($x$) defined in Algorithm 6.2;
8.      end if
9.      for each $t \in \mathcal{T}^2 \setminus \mathcal{T}^f$ enabled at $\mathcal{m}$ do
10.         compute the reachable marking $\mathcal{m}'$ from $\mathcal{m}$ by firing $t$;
11.         if there exists a node on the path from $x_0$ to $x$ such that its marking is smaller than $\mathcal{m}'$ then
12.             select the first-met node on the reverse path from $x$ to $x_0$ such that its marking $\mathcal{m}''<\mathcal{m}'$ and let $\mathcal{m}'(p):=inf$ for each $p \in \mathcal{P}$ such that $\mathcal{m}'(p)>\mathcal{m}''(p)$;
13.         end if
14.         if there already exists a node $x'$ with the marking $\mathcal{m}'$ then
15.             add an arc from $x$ to $x'$ labeled $t$;
16.         else
17.             create a node $x'$ with marking $\mathcal{m}'$, add an arc from $x$ to $x'$ labeled $t$, and tag $x'$ “new”;
18.         end if
19.     end for
20.     untag “new” from node $x$;
21. end while
22. End.
```

Algorithm 6.2: *Function SonNodes* (x)

```
1. Let $x_0 := x$ and tag it "novel" and "$a$";
2. while there exists a node $x$ tagged "novel" do
3.     let $m$ be the marking of $x$;
4.     for each $t \in \mathcal{T}^1$ enabled at $m$ do
5.         compute the reachable marking $m'$ from $m$ by firing $t$;
6.         if there exists a node on the path from $x_0$ to $x$ such that its marking is smaller than $m'$ then
7.             select the first-met node on the reverse path from $x$ to $x_0$ such that its marking $m'' < m'$ and let $m'(p) := inf$ for each $p \in \mathcal{P}$ such that $m'(p) > m''(p)$;
8.         end if
9.         if there already exists a node $x'$ tagged "$a$" with marking $m'$ then
10.            add an arc from $x$ to $x'$ labeled $t$;
11.        else
12.            create a node $x'$ with marking $m'$, add an arc from $x$ to $x'$ labeled $t$, and tag $x'$ "$a$" and "novel";
13.        end if
14.    end for
15.    untag "novel" from node $x$;
16. end while
17. untag "$a$" from all nodes.
```

Let us now provide some comments on Algorithm 6.1. If we ignore Steps 5-8 and replace $\mathcal{T}^2 \backslash \mathcal{T}^f$ by $\mathcal{T}$ in Step 9, then the resulting graph coincides with the reachability graph or the coverability graph of the Predictor Net. Now, by Step 9, every time we generate son nodes of a "new" node, we consider transitions in $\mathcal{T}^2 \backslash \mathcal{T}^f$ only. Moreover, considering Steps 5-8, for each "dangerous" node, Function *SonNodes* is additionally called, where only transitions in $\mathcal{T}^1$ is permitted to fire. Essentially, starting from a "dangerous" node, Function *SonNodes* generates the reachability graph or the coverability graph of the unfaulty net.

Note that the tag "*a*" used in Function *SonNodes* guarantees that all nodes generated by Function *SonNodes* are not connected to nodes generated outside the current execution of the function, except the starting node. As a consequence, given a cycle *c* in the PG, it holds that either $\sigma(c)\in(\mathcal{T}^1)^*$ or $\sigma(c)\in(\mathcal{T}^2)^*$, where $\sigma(c)$ is the corresponding transition sequence associated with *c*.

Example 6.4: The graph in Fig. 6.5 is the PG of the labeled PN in Fig. 6.1(a) and the graph in Fig. 6.6 is the PG of the labeled PN in Fig. 6.1(b). Dangerous nodes are highlighted in grey. Besides, the first entries of each marking with no superscript correspond to places of the original PN, while the other entries with superscript "*u*" correspond to places of the unfaulty net. ♦

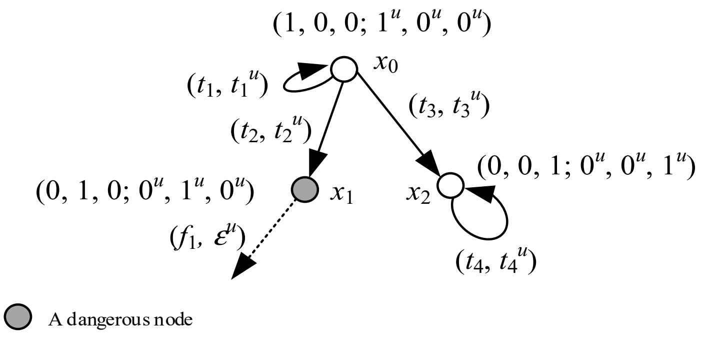


Fig. 6.5 PG of the labeled PN in Fig. 6.1(a)

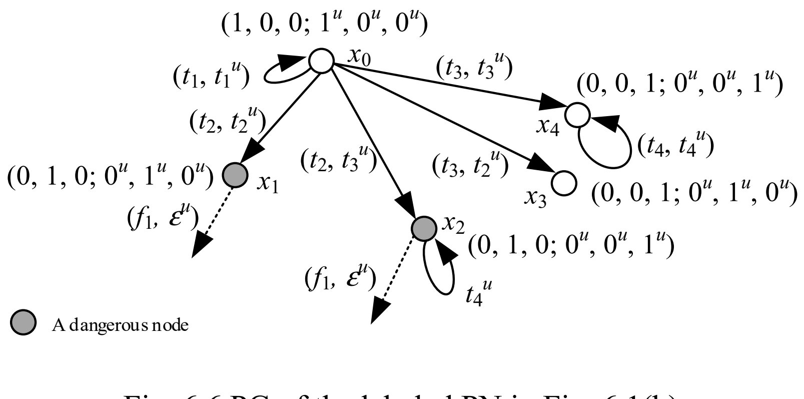


Fig. 6.6 PG of the labeled PN in Fig. 6.1(b)

6.4.3 Necessary and Sufficient Condition for Fault-predictability

In this subsection, we propose a necessary and sufficient condition for the fault-predictability of a labeled PN based on its PG. Before that, we introduce the notion of *repeatable cycle*.

Definition 6.3: Given a labeled PN (N, m_0, l) and its PG, a cycle c in the PG is called *repeatable* if its related transition sequence $\sigma(c)$ is such that $[\mathcal{N}]\cdot\overrightarrow{\sigma(c)}\geq 0$, where $[\mathcal{N}]$ is the incidence matrix of the Predictor Net.

Next, we provide a preliminary result involving repeatable cycles. We note that a marking is called an *ordinary marking* if none of its entries is *inf*, otherwise it is called an *inf-marking*. Indeed, an *inf*-marking can be viewed as a marking set consisting of infinitely many ordinary markings. For instance, $m=(1, 0, inf)^T$ is an *inf*-marking, which can be viewed as $m=\{(1, 0, 0)^T, (1, 0, 1)^T, (1, 0, 2)^T, \ldots\}$. For consistency, an ordinary marking can be viewed as a marking set containing only one ordinary marking. In a PG, we use $m(x)$ to denote the marking associated with a node x.

Property 6.1: Given a cycle c in PG, $\forall x\in X(c)$, there exists an ordinary marking m_1 in $m(x)$ such that $m_1[\sigma(c)\rangle m_2[\sigma(c)\rangle m_3\ldots$, i.e., the sequence $\sigma(c)$ can fire infinitely from m_1, *iff* c is a repeatable cycle.

Proof: (=>) By Definition 6.3, if c is a repeatable cycle then $[\mathcal{N}]\cdot\overrightarrow{\sigma(c)}\geq 0$. This implies that $m_i\leq m_j$ in the case that m_j is reached from m_i by firing $\sigma(c)$ in the Predictor Net $\mathcal{N}$. Trivially, there exists an ordinary marking m_1 of x such that $m_1[\sigma(c)\rangle m_2[\sigma(c)\rangle m_3\ldots$.

(<=) We can see that $m_i\leq m_{i+1}$, $\forall i\in\{1, 2, \ldots\}$. Clearly, we have $[\mathcal{N}]\cdot\overrightarrow{\sigma(c)}\geq 0$. Hence, c is a repeatable cycle. ■

Roughly speaking, only when a cycle in PG is a repeatable cycle, the sequence associated with the cycle can fire infinitely often.

Remark 6.2: The concept of repeatable cycles is essentially the same as that of cycles associated with a firable repetitive sequence in [10]. ♣

Theorem 6.1: A labeled PN (N, m_0, l) is fault-unpredictable *iff* there exists a dangerous node x_d in the corresponding PG such that there exists a repeatable cycle accessible from it.

Proof: (=>) Let c be a repeatable cycle in PG that is accessible from a dangerous node x_d. By Algorithm 6.1, two cases may occur: 1) c is generated by *SonNodes*(x_d); 2) c is not generated by *SonNodes*(x_d).

In Case 1, $\sigma(c)\in(\mathcal{T}^1)^*$, i.e., c is a repeatable cycle of the reachability/coverability graph of the unfaulty net starting from the marking $\mathcal{m}(x_d)$. Based on Property 6.1, it is trivial to see that there exists an ordinary marking $\mathcal{m}_1\in\mathcal{m}(x_d)$ such that an arbitrarily long sequence can occur from m_1^u in the unfaulty net, where m_1^u is the restriction of $\mathcal{m}_1$ to places of the unfaulty net.

In Case 2, $\sigma(c)\in(\mathcal{T}^2)^*$. By Proposition 6.2 and Property 6.1, we can also conclude that there exists an ordinary marking $\mathcal{m}_1\in\mathcal{m}(x_d)$ such that an arbitrarily long sequence can occur from m_1^u in the unfaulty net.

If $\mathcal{m}(x_d)$ itself is an ordinary marking, then $\mathcal{m}_1$ definitely enables a transition $t\in\mathcal{T}^f$ since x_d is a dangerous node. If $\mathcal{m}(x_d)$ is an *inf*-marking, then $\exists\mathcal{m}_1'\in\mathcal{m}(x_d)$ such that $\mathcal{m}_1'\geq\mathcal{m}_1$ and $\mathcal{m}_1'$ enables a transition $t\in\mathcal{T}^f$ since x_d is a dangerous node. Consequently, no matter in which case, there exists a sequence $\sigma^\#\in L(\mathcal{N},\mathcal{m}_0)\cap(\mathcal{T}^2)^*$ whose firing at $\mathcal{m}_0$ leads to an ordinary marking $\mathcal{m}'\in\mathcal{m}(x_d)$ such that $\mathcal{m}'$ enables a transition $t\in\mathcal{T}^f$ and an arbitrarily long sequence can occur from m'^u in the unfaulty net, where m'^u is the restriction of $\mathcal{m}'$ to places of the unfaulty net. Clearly, t is a transition pair and its first component is a fault transition. Here, we use f to denote this fault transition. Let σ and θ be two sequences corresponding to the first and second components of $\sigma^\#$. By

Proposition 6.2, $\sigma \in L(N, m_0)$ and $\theta \in L(N^u, m_0^u)$ with $l(\sigma) = l^u(\theta)$. Besides, we can see that $\sigma f \in L(N, m_0)$ and an arbitrarily long sequence can occur following θ in the unfaulty net. Trivially, it holds that $\theta \in L(N, m_0)$, $l(\sigma) = l(\theta)$ and $(\forall K \in \mathbf{N})$ $(\exists \theta\theta' \in L(N, m_0))$ $[(|\theta'| > K) \wedge (\theta' \cap T_F = \emptyset)]$, i.e., an arbitrarily long normal sequence can occur following θ in (N, m_0). Therefore, the labeled PN is fault-unpredictable by Proposition 6.1.

(<=) By contradiction, suppose that for each dangerous node in PG, there exists no repeatable cycle accessible from it. By Algorithm 6.1 and Property 6.1, it holds: $(\forall \sigma f \in L(N, m_0)\colon \sigma \in T_N^* \wedge f \in T_F)(\forall \theta \in L(N^u, m_0^u): l(\sigma) = l^u(\theta))$, a deadlock definitely arises in finite steps following θ in (N^u, m_0^u). Since (N^u, m_0^u) is derived from (N, m_0) by deleting fault transitions only and (N, m_0) does not enter a deadlock before the occurrence of a fault by Assumption A1, we can see that a fault transition definitely fires in finite steps following θ in (N, m_0). In other words,

$$(\forall \sigma f \in L(N, m_0): \sigma \in T_N^* \wedge f \in T_F)$$

$$(\forall \theta \in L(N, m_0): (l(\theta) = l(\sigma)) \wedge (\theta \cap T_F = \emptyset))$$

$$(\exists K \in \mathbf{N})(\forall \theta\theta' \in L(N, m_0))[(|\theta'| > K) \Rightarrow (\theta' \cap T_F \neq \emptyset)].$$

It obviously contradicts Proposition 6.1. Consequently, there exists a dangerous node x_d in PG such that there exists a repeatable cycle accessible from x_d. ■

It is easy to see that, for labeled PNs that are bounded, any cycle in the corresponding PG is repeatable. As a consequence, the following result trivially follows from Theorem 6.1.

Corollary 6.1: A labeled bounded PN (N, m_0, l) is fault-unpredictable *iff* there exists a dangerous node x_d in the corresponding PG such that there exists a cycle accessible from it.

Example 6.5: Consider the PG in Fig. 6.5 relative to the bounded labeled PN in Fig. 6.1(a), where x_1 is the only dangerous node. We can see that no cycle is accessible from x_1. Hence, by Corollary 6.1, the labeled PN in Fig. 6.1(a) is fault-predictable.

As for the PG in Fig. 6.6 relative to the bounded labeled PN in Fig. 6.1(b), there exists a dangerous node x_2 that is contained in a cycle and thus the labeled PN in Fig. 6.1(b) is fault-unpredictable. ♦

Example 6.6: Consider the labeled unbounded PN in Fig. 6.7, where $T_F=\{f_1,f_2\}$ and $T_N=\{t_1\text{-}t_5\}$. Transitions f_1 and f_2 are unobservable, while the other transitions are observable. Note that f_1 and f_2 belong to the same class of faults. We use the proposed approach to determine if the labeled PN is fault-predictable.

First, we compute the unfaulty net reported in Fig. 6.8.

Next, the Predictor Net is constructed as shown in Fig. 6.9.

Finally, the PG of the labeled PN is computed as shown in Fig. 6.10. Looking at the PG, we can see that dangerous nodes x_2, x_3, x_4, x_5 are followed by cycles. According to Theorem 6.1, we need to further determine if such cycles are repeatable. It can be verified that $[\mathcal{N}]\cdot\overrightarrow{t_3^u} = [\mathcal{N}]\cdot\overrightarrow{t_5^u} = (0, 0, 0, 0; 0^u, 0^u, 0^u, -1^u)$ and $[\mathcal{N}]\cdot\overrightarrow{(t_3,t_3^u)} = [\mathcal{N}]\cdot\overrightarrow{(t_5,t_5^u)} = (0, 0, 0, -1; 0^u, 0^u, 0^u, -1^u)$, where $[\mathcal{N}]$ is the incidence matrix of the Predictor Net. Hence, by Definition 6.3, we can see that there is no repeatable cycle accessible from a dangerous node. Consequently, we may conclude that the labeled unbounded PN in Fig. 6.7 is fault-predictable. ♦

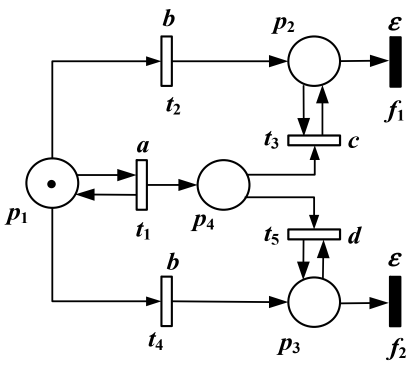


Fig. 6.7 Labeled unbounded PN

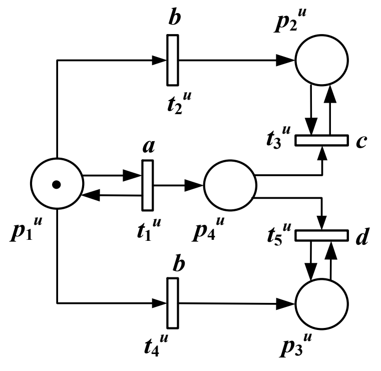


Fig. 6.8 Unfaulty net of the net in Fig. 6.7

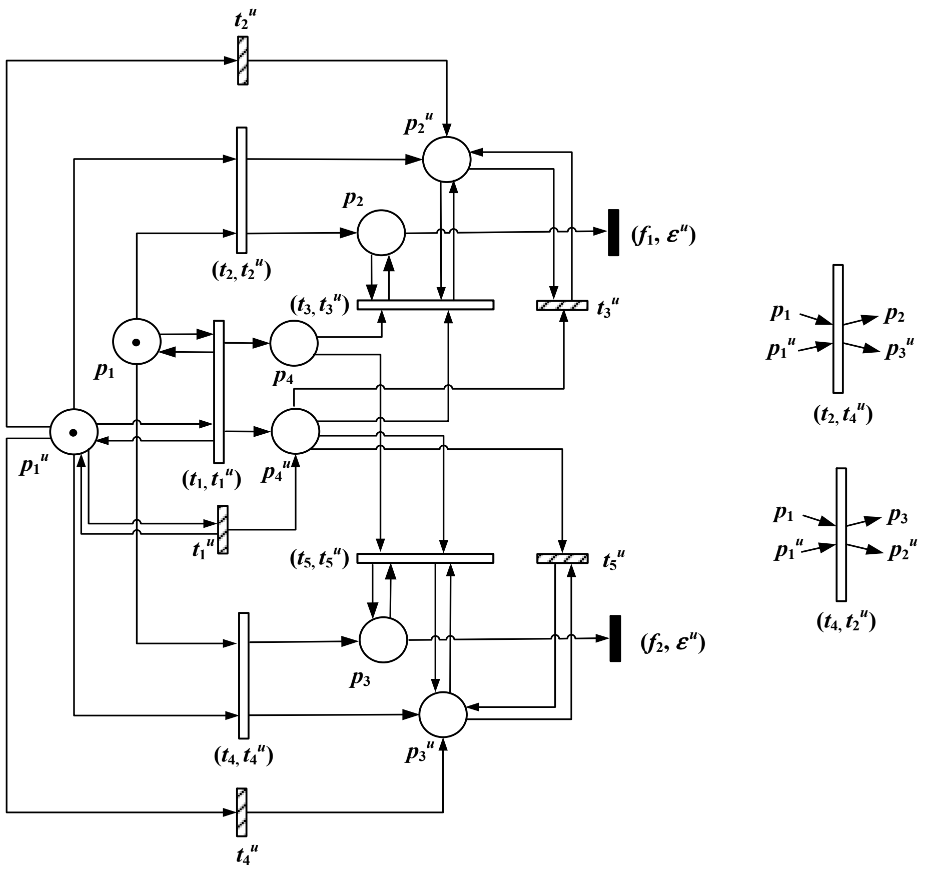


Fig. 6.9 Predictor Net of the labeled PN in Fig. 6.7

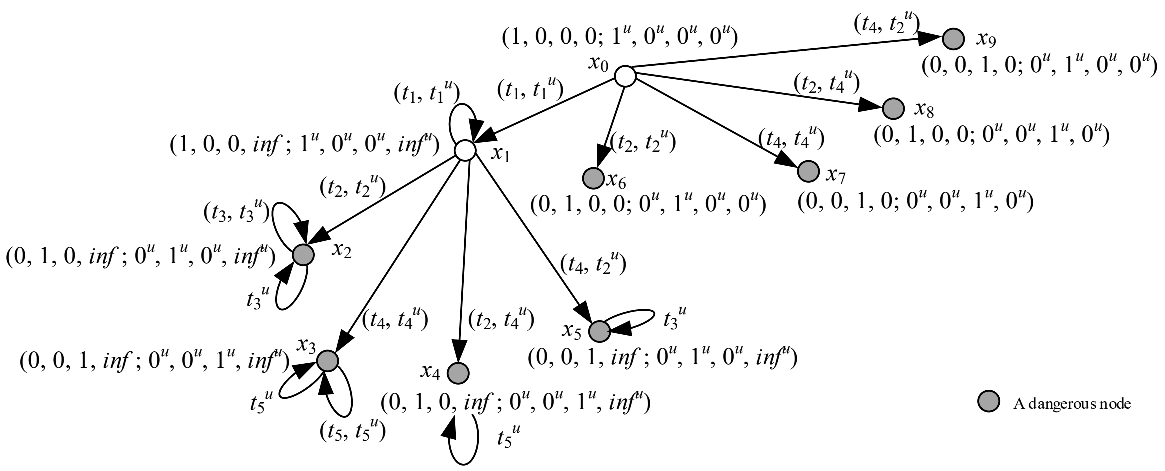

x_0 $(1, 0, 0, 0; 1^u, 0^u, 0^u, 0^u)$
x_1 $(1, 0, 0, inf; 1^u, 0^u, 0^u, inf^u)$
x_2 $(0, 1, 0, inf; 0^u, 1^u, 0^u, inf^u)$
x_3 $(0, 0, 1, inf; 0^u, 0^u, 1^u, inf^u)$
x_4 $(0, 1, 0, inf; 0^u, 0^u, 1^u, inf^u)$
x_5 $(0, 0, 1, inf; 0^u, 1^u, 0^u, inf^u)$
x_6 $(0, 1, 0, 0; 0^u, 1^u, 0^u, 0^u)$
x_7 $(0, 0, 1, 0; 0^u, 0^u, 1^u, 0^u)$
x_8 $(0, 1, 0, 0; 0^u, 0^u, 1^u, 0^u)$
x_9 $(0, 0, 1, 0; 0^u, 1^u, 0^u, 0^u)$
(t_1, t_1^u)
(t_2, t_2^u)
(t_3, t_3^u)
t_3^u
(t_4, t_4^u)
(t_2, t_4^u)
(t_4, t_2^u)
t_5^u
(t_5, t_5^u)
A dangerous node

Fig. 6.10 PG of the labeled PN in Fig. 6.7

Remark 6.3: According to Corollary 6.1, it is easy to verify the fault-predictability of PNs that are bounded. In the case of unbounded PNs, it could be more complicated especially for large scale nets since establishing whether cycles are repeatable requires further investigation. Fortunately, Cabasino *et al.* [10] discuss in detail how to search repeatable cycles efficiently in a graph. In particular, they show that repeatable cycles can be found out by solving linear programming problems. ♣

Remark 6.4: The computational complexity of the proposed approach is exponential with respect to the net size (i.e., the number of places, transitions, and tokens initially present in the net). This follows from the fact that the PG is similar to the reachability/coverability graph of the Predictor Net. ♣

6.4.4 Reduction Rules

When the labeled PN has large size or it has many transitions with the same label, the Predictor Net can be very complicated since any two normal transitions from the original PN and its unfaulty net with the same label are required to be synchronized. Actually, not all transitions are relevant when studying fault-predictability. In this subsection, we propose two rules to remove some unnecessary transitions and places from the original labeled PN before constructing the Predictor Net.

Rule 1: Given a labeled PN, it is reduced by two steps:

1) Delete all normal transitions that become dead in the case that all fault transitions are forbidden to fire, as well as their related arcs; and then

2) Delete all places with no output transitions as well as their related arcs.

Theorem 6.2: Given a labeled PN (N, m_0, l) and its reduced net (N', m_0', l') obtained by Rule 1, (N, m_0, l) is fault-predictable *iff* (N', m_0', l') is fault-predictable.

Proof: We can see that the transitions deleted by Rule 1 can only fire after the firing of fault transitions in the original net. Trivially, since the original net satisfies Assumptions A1 and A2, the reduced net still satisfies Assumptions A1 and A2. According to Proposition 6.1, there is no need to consider transitions that fire only after the firing of fault transitions. As a result, we can conclude that (N, m_0, l) is fault-predictable *iff* (N', m_0', l') is fault-predictable. ■

Rule 1 is of exponential complexity with respect to the net size since it involves the enumeration of dead transitions. Consequently, it is not applicable to PNs with very large size. In the following, we present a reduction rule that can be applied to ordinary PNs, which is a refinement of Rule 1 and enjoys polynomial complexity with respect to the net size.

Rule 2: Given a labeled ordinary PN (N, m_0, l), the reduced net (N', m_0', l') is derived by the following procedure:

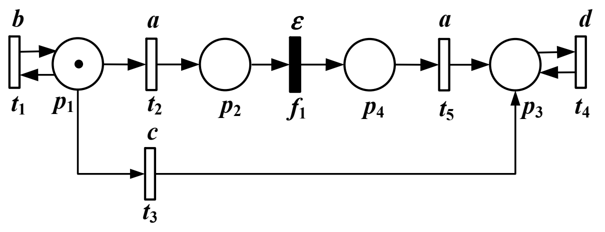
```
$P_D:=\{p\in P \mid ((^\bullet p)\cap T\subseteq T_F) \wedge (m_0(p)=0)\}$;
while $P_D\neq\emptyset$ do
    $P:=P\backslash P_D$, $T:=T\backslash P_D{}^\bullet$, and delete the related arcs;
    $P_D:=\{p\in P \mid ((^\bullet p)\cap T\subseteq T_F) \wedge (m_0(p)=0)\}$;
end while
denote the resulting net as ($N'$, $m_0'$, $l'$).
```

In words, Rule 2 repeatedly deletes places whose inputs are all fault transitions and token count is 0 at the initial marking m_0, plus their output transitions and their related arcs.

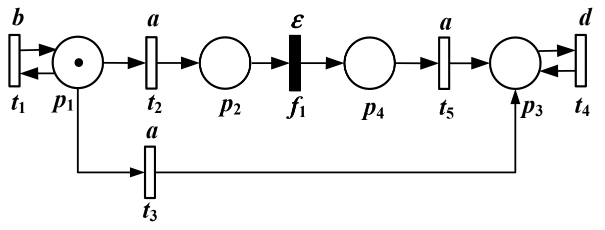
Theorem 6.3: Given a labeled ordinary PN (N, m_0, l) and its reduced net (N', m_0', l') obtained by Rule 2, (N, m_0, l) is fault-predictable *iff* (N', m_0', l') is fault-predictable.

Proof: For any place p with all its input transitions dead at m_0 and $m_0(p)=0$, its output transitions are clearly dead. Since N is ordinary, it is trivial to see that all deleted transitions by Rule 2 are those that should be removed by Rule 1, i.e., normal transitions that become dead transitions in the case that all fault transitions are forbidden to fire. Besides, all deleted places by Rule 2 are exactly those that should be removed by Rule 1. Hence, the conclusion holds according to Theorem 6.2. ■

Remark 6.5: The reduced net obtained by Rule 1 or 2 still satisfies Assumptions A1 and A2 in the case that the original net satisfies Assumptions A1 and A2. Hence, the proposed approach can be correctly applied to the reduced nets. In addition, after we apply the reduction rules to PNs, the Predictor Net of the reduced net is typically simpler than that of the original net while the PG does not change. However, the PG is constructed starting from the Predictor Net, so we have an advantage while performing intermediate computations. ♣



(a) A fault-predictable labeled PN



(b) A fault-unpredictable labeled PN

Fig. 6.11 Two labeled PNs

Example 6.7: Let us consider the two ordinary labeled PNs in Fig. 6.11. According to Rule 2, in both cases, transition t_5 and place p_4, as well as their related arcs can be removed, resulting in the reduced nets exactly the same as those in Fig. 6.1(a) and (b), respectively. By Theorem 6.3, we can verify the fault-predictability of the two PNs in Fig. 6.11(a) and (b) by verifying the fault-predictability of the nets in Fig. 6.1(a) and (b), respectively. Note that the Predictor Net of the labeled PN in Fig. 6.11(a) contains 8 places and 13 transitions, while the Predictor Net of its reduced net contains 6 places and 9 transitions only, as shown in Fig. 6.3. As for the labeled PN in Fig. 6.11(b), its Predictor Net has 8 places and 17 transitions, while the Predictor Net of its reduced net has 6 places and 11 transitions only, as shown in Fig. 6.4. ♦

Finally, we notice that after using reduction rules, the verification of the fault-predictability by the proposed approach still requires the construction of the PG. It implies that the computational complexity of the proposed approach after using reduction rules is still exponential with respect to the net size.

6.5 Conclusions

This chapter presents a novel approach to verify fault-predictability in labeled PNs that may also be unbounded. Specifically, by characterizing the structure of a special graph, called *Predictor Graph*, a necessary and sufficient condition for fault-predictability is derived. Moreover, two rules to reduce the size of the net are proposed and it is proven that the fault-predictability of the original net can be verified by checking the fault-predictability of the reduced net.

The work of this chapter has been published in *IEEE Transactions on Automatic Control*; see [125].

CHAPTER VII

Event-based Opacity and Its Verification

7.1 Introduction

Opacity is an important information flow property related to privacy and security of a system. The notions of opacity in the literature are generally classified into language-based opacity and state-based opacity. In this chapter, we use deterministic finite-state automata (DFA) as the reference formalism, proposing four notions of event-based opacity, namely, *K-observation event-opacity*, *infinite-observation event-opacity*, *event-opacity* and *combinational event-opacity*. In simple words, these properties characterize situations in which an intruder, based on a partial observation of the system evolution, may never establish the occurrence of a secret event within its critical horizon. The critical horizon of a secret event is characterized by a positive integer, e.g., K, which defines the horizon from the instant when the secret event occurs until the instant when the K-th observable event occurs after the secret. A motivation example is presented in the chapter, relative to a business company taking decisions, some of which should remain secret to external observers. In addition, the relationship between K-observation/infinite-observation event-opacity and (K-step) diagnosability is analyzed. Moreover, appropriate verifiers are proposed to verify the proposed four properties.

This chapter is organized as follows. Section 7.2 presents some notations used in this chapter but not introduced in Chapter II. Section 7.3 provides a practical scenario that motivates the notions of event-based opacity. Four notions of event-based opacity are proposed in Section 7.4, as well as the analysis on the relationship between the proposed K-observation/infinite-observation event-opacity and (K-step) diagnosability. Methods for the verification of the proposed notions are provided in Section 7.5. Section 7.6 concludes this chapter.

We notice that the work presented in this chapter is now under review by *IEEE Transactions on Automatic Control*.

7.2 Preliminaries

In this chapter we assume that DES are modeled as *deterministic finite-state automata* (DFA). The readers are referred to Section 2.3.1 for the basic notions related to DFA.

Given a DFA $G=(X, \Sigma, \delta, x_0)$, the event set Σ is partitioned into two disjoint sets, Σ_o the set of *observable* events and Σ_u the set of *unobservable* events, i.e., $\Sigma=\Sigma_o\cup\Sigma_u$ and $\Sigma_o\cap\Sigma_u=\varnothing$. The *observation function* in the framework of DFA is typically denoted by P, namely, it is a projection P: $\Sigma^*\rightarrow\Sigma_o^*$ such that

1) $P(\epsilon)=\epsilon$; and

2) $\forall u\in\Sigma^*, v\in\Sigma,\ P(uv)=\begin{cases} P(u)v & \text{if } v\in\Sigma_o, \\ P(u) & \text{otherwise.}\end{cases}$

The observation function P is also extended to the domain 2^{Σ^*} such that $\forall L\in 2^{\Sigma^*}$, $P(L)=\{P(\sigma)|\sigma\in L\}$. Moreover, we define P_G^{-1}: $\Sigma_o^*\rightarrow L(G)$ such that $P_G^{-1}(w)=P^{-1}(w)\cap L(G)$, i.e., $P_G^{-1}(w)=\{\sigma\in L(G)|\ P(\sigma)=w\}$. We call $\sigma\in P_G^{-1}(w)$ a *consistent string* of w in G.

7.3 Motivational Scenario

In this section, we introduce a practical scenario that motivates the notions of event-based opacity proposed in this work.

Consider a company whose board of directors issues various decisions. Some decisions are public while others are private, i.e., only known inside the company. Suppose that the company issues a public decision every day at a fixed time and may issue a private decision at any time. Some of the private decisions are critical to the interests and the development of the company and thus its competitor is really interested in detecting such decisions that we call *secret decisions*. If a secret decision is successfully detected by the competitor, the competitor may propose some countermeasures. We assume that there is a *critical horizon* (given in days in this scenario) that allows the competitor to cope with a secret decision since it is issued. It means that, if a secret decision is not detected within its critical horizon, it is of no use for the competitor to propose any countermeasure. Thus, to be precise, the competitor aims to detect secret decisions that are within their critical horizons, while the company certainly hopes to keep every secret decision undetectable within its critical horizon.

Suppose that a DFA models how the decisions will be issued by the company. Such a model can be established based on the historical data. In the DFA model, an event corresponds to a decision. Naturally, public decisions are modelled as observable events and private decisions are unobservable events. Secret decisions correspond to some of the unobservable events that we call *secret events*. We use Σ_s to denote the set of secret events. Note that the DFA model does not contain any cycle with unobservable events only since a public decision is issued every day. For example, a possible decision-making model of a company is shown in Fig. 7.1 formalized as a DFA $G=(X, \Sigma, \delta, x_0)$. It is $\Sigma_o=\{a, b, c, d\}$, $\Sigma_u=\{u_1\text{-}u_4, s\}$ and $\Sigma_s=\{s\}$. It indeed shows two possible decision-making routes, namely, $u_1absu_3cd^*$ and $u_2au_4bcsd^*$. In the two routes, the daily issued public decisions are the same, that is, $a, b, c, d, d, \ldots$, but the issuing of private decisions is different.

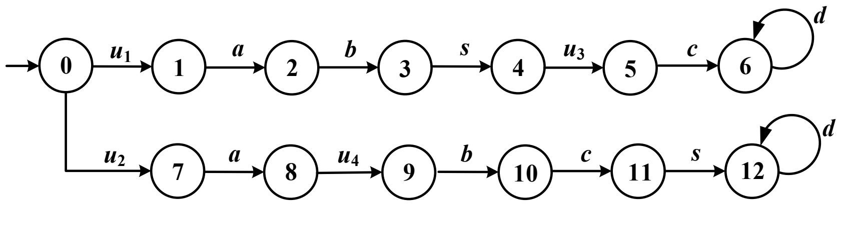


Fig. 7.1 1-observation event-opaque DFA G

We assume that the competitor has the complete knowledge of the decision-making model. Based on the model, according to the observed events, it detects secret events that are within their critical horizons. The critical horizon of each secret event is assumed to be known to both the company and the competitor. In addition, the competitor is assumed to be *conservative* in the sense that it takes countermeasures only when it knows for sure that it is now within the critical horizon of a secret event. Consider the case that the critical horizon of each secret decision is within three days. From the perspective of the company, it should be guaranteed that every time a secret decision is issued, in the following three days, the competitor cannot know for sure that a secret decision was issued in the past three days; or equivalently, every day the competitor is informed about a new public decision, it cannot know for sure that a secret decision was issued in the past three days. Such a property is called *3-observation event-opacity*, which will be formally introduced in this work. In the case that the model is 3-observation event-opaque, no effective countermeasure can be taken by the competitor to cope with any secret decision. In other words, the security of the decision-making model is guaranteed.

7.4 Four Notions of Event-based Opacity

In this section, we propose four notions of event-based opacity, which guarantee the security of a system in different scenarios from the simplest one to the most complicate one. In addition, we analyze the relationship between the proposed *K-observation/infinite-observation event-opacity* and (*K*-step) diagnosability in the literature due to their similarity.

7.4.1 *K*-observation and Infinite-observation Event-opacity

In this subsection, we propose the notions of *K-observation* and *infinite-observation event-opacity*, which characterize the security requirement of a system in a scenario where the following two assumptions are made:

A1) the critical horizon of each secret event is the same;

A2) the adversary is not interested in distinguishing among different secret events.

We introduce some notations before presenting these two notions. Note that, it is assumed that $K \in \mathbb{Z}^+$ in the chapter.

Given a DFA $G=(X, \Sigma, \delta, x_0)$ and an event set $\Sigma' \subseteq \Sigma$, we denote $\Psi(\Sigma')=\{\alpha\in L(G)| \alpha =\sigma e$, where $\sigma\in \Sigma^*$ and $e\in \Sigma'\}$, i.e., the set of strings in $L(G)$ ending with an event $e\in \Sigma'$.

Recall that a string $v\in \Sigma^*$ is a *suffix* of $\sigma\in \Sigma^*$ if $\exists u\in \Sigma^*$ such that $uv=\sigma$. Now, we define the *K-observation suffix* of $\sigma\in \Sigma^*$, denoted by $sufob_K(\sigma)$, as the suffix of σ starting with the last K-th observable event if $|P(\sigma)|\geq K$ and coincides with the entire σ if $|P(\sigma)|<K$.

As an example, consider a string $\sigma=u_1u_2au_1bu_1cu_2$, where u_1, $u_2\in \Sigma_u$ and a, b, $c\in \Sigma_o$. It is $sufob_2(\sigma)=bu_1cu_2$ and $sufob_4(\sigma)=\sigma=u_1u_2au_1bu_1cu_2$.

Given a string $\sigma\in \Sigma^*$ and an event set $\Sigma' \subseteq \Sigma$, with a slight abuse of notation, we write $\sigma\cap\Sigma'\neq\emptyset$ if σ contains an event in Σ', and $\sigma\cap\Sigma'=\emptyset$ otherwise.

Definition 7.1: A DFA $G=(X, \Sigma, \delta, x_0)$ is said to be *K-observation event-opaque* w.r.t. a set of secret events $\Sigma_S\subseteq\Sigma_u$ if

$$(\forall \alpha\in \Psi(\Sigma_S))(\forall \beta\in L(G)/\alpha\colon 0<|P(\beta)|\leq K)$$
$$[\exists \sigma\in P_G^{-1}(P(\alpha\beta)), sufob_{K+1}(\sigma)\cap\Sigma_S=\emptyset].$$

In words, K-observation event-opacity implies that for any string α ending with a secret event and any follow-up string β containing at least one but no more than K observable events, there exists a string σ that produces the same observation as $\alpha\beta$ but its $(K+1)$-observation suffix does not contain any secret event.

Recall the motivational scenario. The physical meaning of K-observation event-opacity in that scenario is that every time a secret decision is issued, in the following K days, according to the daily public decisions, the competitor cannot know for sure that a secret decision was issued in the past K days because there exists a possibility that no secret decision was issued in the past K days. Note that the $(K+1)$-observation suffix refers to the past K days.

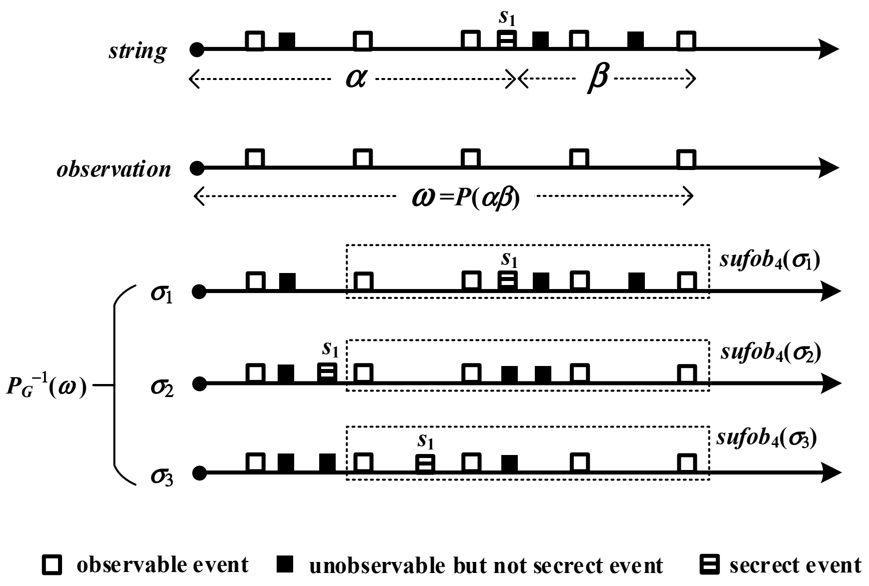


Fig. 7.2 Illustration of 3-observation event-opacity

To be intuitive, the illustration of 3-observation event-opacity is shown in Fig. 7.2, where we focus on one string $\alpha\in\Psi(\Sigma_S)$ and one string $\beta\in L(G)/\alpha$ with $|P(\beta)|=2$ only. We observe that there exists a string consistent with the observation $w=P(\alpha\beta)$, namely σ_2, whose 4-observation suffix does not contain any secret event. If for any $\alpha\in\Psi(\Sigma_S)$ and any $\beta\in L(G)/\alpha$ with $0<|P(\beta)|\leq 3$, it is the case like the one shown in Fig. 7.2, i.e., there exists a string consistent with the observation $w=P(\alpha\beta)$ whose 4-observation suffix does not contain any secret event, the DFA G is 3-observation event-opaque.

Example 7.1: Consider again the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.1, where $\Sigma_o=\{a, b, c, d\}$, $\Sigma_u=\{u_1\text{-}u_4, s\}$ and $\Sigma_S=\{s\}$. It is $\Psi(\Sigma_S)=\{\alpha_1, \alpha_2\}=\{u_1abs, u_2au_4bcs\}$.

Let $K=1$. Consider $\alpha_1=u_1abs$ and $\beta=u_3c\in L(G)/\alpha_1$. It is $P(\alpha_1\beta)=abc$ and $P_G^{-1}(abc)=\{u_1absu_3c, u_2au_4bc, u_2au_4bcs\}$. It is $sufob_2(u_1absu_3c)=bsu_3c$, $sufob_2(u_2au_4bc)=bc$ and $sufob_2(u_2au_4bcs)=bcs$. Thus, $sufob_2(u_2au_4bc)\cap\Sigma_S=\emptyset$.

Consider $\alpha_2=u_2au_4bcs$ and $\beta=d\in L(G)/\alpha_2$. It is $P(\alpha_2\beta)=abcd$ and $P_G^{-1}(abcd)=\{u_2au_4bcsd, u_1absu_3cd\}$. Moreover, it is $sufob_2(u_2au_4bcsd)=csd$ and $sufob_2(u_1absu_3cd)=cd$. Thus, $sufob_2(u_1absu_3cd)\cap\Sigma_S=\emptyset$.

Consequently, the DFA is 1-observation event-opaque by Definition 7.1.

Now, let K=2. The DFA is not 2-observation event-opaque. To show this, let us consider $\alpha_1=u_1abs$ and $\beta=u_3cd\in L(G)/\alpha_1$. It is $P(\alpha_1\beta)=abcd$ and $P_G^{-1}(abcd)=\{u_1absu_3cd, u_2au_4bcsd\}$. Thus, $sufob_3(u_1absu_3cd)\cap\Sigma_S\neq\varnothing$ and $sufob_3(u_2au_4bcsd)\cap\Sigma_S\neq\varnothing$. ♦

The following proposition presents an alternative way to determine whether a DFA is K-observation event-opaque.

Proposition 7.1: A DFA $G=(X, \Sigma, \delta, x_0)$ is *K-observation event-opaque* w.r.t. a set of secret events $\Sigma_S\subseteq\Sigma_u$ *iff*

$$(\forall w\in P(L(G)))[\exists\sigma\in P_G^{-1}(w), sufob_{K+1}(\sigma)\cap\Sigma_S=\varnothing].$$

Proof: (=>) It trivially holds by Definition 7.1.

(<=) By contradiction, suppose that

$$(\exists w\in P(L(G)))[\forall\sigma\in P_G^{-1}(w), sufob_{K+1}(\sigma)\cap\Sigma_S\neq\varnothing].$$

Hence, there exists a string $\alpha\beta\in P_G^{-1}(w)$ with $\alpha\in\Psi(\Sigma_S)$ and $0<|P(\beta)|\leq K$. Since $P(\alpha\beta)=w$, it holds that $\forall\sigma\in P_G^{-1}(P(\alpha\beta))$, $sufob_{K+1}(\sigma)\cap\Sigma_S\neq\varnothing$, which however contradicts the fact that G is K-observation event-opaque by Definition 7.1. Consequently, it is

$$(\forall w\in P(L(G)))[\exists\sigma\in P_G^{-1}(w), sufob_{K+1}(\sigma)\cap\Sigma_S=\varnothing]. \quad ■$$

By Proposition 7.1, K-observation event-opacity implies that for any observation from the DFA, there exists at least one consistent string whose (K+1)-observation suffix does not contain any secret event. Recall the motivational scenario, it means that every day the competitor gets a public decision issued by the company, it cannot know for sure that a secret decision was issued in the past K days since there exists a possibility that no secret decision was issued in the past K days.

Example 7.2: Consider again the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.1, where $\Sigma_o=\{a, b, c, d\}$, $\Sigma_u=\{u_1\text{-}u_4, s\}$ and $\Sigma_S=\{s\}$. Now, we use Proposition 7.1 to determine if it is K-observation event-opaque. It is $P(L(G))=\{\epsilon, a, ab, abc, abcd, abcdd, \ldots\}$.

Let K=1. Consider $ab\in P(L(G))$ as an example. We can find a string $u_2au_4b\in P_G^{-1}(ab)$ such that $sufob_2(u_2au_4b)\cap\Sigma_S=\varnothing$. By checking all the observations, we can see that

$$(\forall w\in P(L(G)))[\exists\sigma\in P_G^{-1}(w), sufob_2(\sigma)\cap\Sigma_S=\varnothing].$$

Thus, the DFA is 1-observation event-opaque.

Now, let K=2. We can find an observation $abcd \in P(L(G))$ whose $P_G^{-1}(abcd)=\{u_1absu_3cd, u_2au_4bcsd\}$. It holds $\forall \sigma \in P_G^{-1}(abcd)$, $sufob_3(\sigma) \cap \Sigma_S \neq \emptyset$. Hence, the DFA is not 2-observation event-opaque. ♦

Remark 7.1: Proposition 7.1 may be viewed as the definition of K-observation event-opacity from a different viewpoint with respect to Definition 7.1. Specifically, Definition 7.1 cares if a certain condition is satisfied for every secret-ended string, while Proposition 7.1 considers if a certain condition is satisfied for every observation. Definition 7.1 is in a form similar to that of (K-step) diagnosability [83], thus facilitating the comparison between the notions of K-observation event-opacity and K-step diagnosability. Proposition 7.1 provides a more intuitive explanation of K-observation event-opacity, namely, it captures a situation where an intruder, based on a partial observation of the system evolution, may never establish the occurrence of a secret event within its critical horizon, i.e., the horizon from the instant when the secret event occurs until the instant when the K-th observable event occurs. ♣

In what follows, we present the notion of *infinite-observation event-opacity*, which is the case that K is infinity.

Definition 7.2: A DFA $G=(X, \Sigma, \delta, x_0)$ is said to be *infinite-observation event-opaque* w.r.t. a set of secret events $\Sigma_S \subseteq \Sigma_u$ if

$$(\forall \alpha \in \Psi(\Sigma_S))(\forall \beta \in L(G)/\alpha\text{: } |P(\beta)|>0)$$
$$[\exists \sigma \in P_G^{-1}(P(\alpha\beta)), \sigma \cap \Sigma_S = \emptyset].$$

Proposition 7.2: A DFA $G=(X, \Sigma, \delta, x_0)$ is *infinite-observation event-opaque* w.r.t. a set of secret events $\Sigma_S \subseteq \Sigma_u$ *iff*

$$(\forall w \in P(L(G)))[\exists \sigma \in P_G^{-1}(w), \sigma \cap \Sigma_S = \emptyset].$$

Proof: Similar to the proof of Proposition 7.1. ■

The following two results show the relationship among infinite-observation, k-observation and l-observation event-opacity, where k and l are positive integers such that $k>l$, which is also depicted in Fig. 7.3 for intuition.

Result 7.1: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S \subseteq \Sigma_u$ and two positive integers k and l with $k>l$, it holds that

G is k-observation event-opaque w.r.t. Σ_S

=> G is l-observation event-opaque w.r.t. Σ_S.

Proof: Since G is k-observation event-opaque w.r.t. Σ_S, then $(\forall w \in P(L(G)))$ $[\exists \sigma \in P_G^{-1}(w), sufob_{k+1}(\sigma) \cap \Sigma_S = \varnothing]$ according to Proposition 7.1. Since $k>l$, it holds that $sufob_{l+1}(\sigma)$ is a suffix of $sufob_{k+1}(\sigma)$ or $sufob_{l+1}(\sigma)=sufob_{k+1}(\sigma)$. In any case, $sufob_{l+1}(\sigma) \cap \Sigma_S = \varnothing$. Thus, G is l-observation event-opaque w.r.t. Σ_S. ■

Result 7.2: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S \subseteq \Sigma_u$, it holds that

G is infinite-observation event-opaque w.r.t. Σ_S

=> G is k-observation event-opaque w.r.t. Σ_S, $\forall k \in \mathbb{Z}^+$.

Proof: Since G is infinite-observation event-opaque w.r.t. Σ_S, then $(\forall w \in P(L(G)))[\exists \sigma \in P_G^{-1}(w), \sigma \cap \Sigma_S = \varnothing]$ by Proposition 7.2. Trivially, $(\forall w \in P(L(G)))[\exists \sigma \in P_G^{-1}(w), sufob_{k+1}(\sigma) \cap \Sigma_S = \varnothing]$, $\forall k \in \mathbb{Z}^+$. Thus, G is k-observation event-opaque w.r.t. Σ_S, $\forall k \in \mathbb{Z}^+$. ■

Example 7.3: Consider again the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.1, where $\Sigma_o=\{a, b, c, d\}$, $\Sigma_u=\{u_1\text{-}u_4, s\}$ and $\Sigma_S=\{s\}$. As discussed in Example 7.1 or 7.2, G is not 2-observation event-opaque. Thus, it is not K-observation event-opaque for any $K\geq 2$ and is not infinite-observation event-opaque due to Results 7.1 and 7.2. ♦

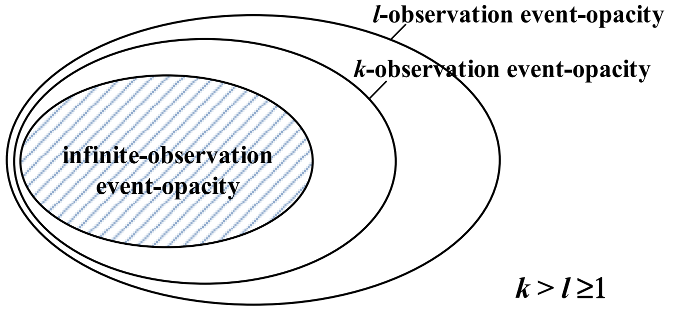

l-observation event-opacity
k-observation event-opacity
infinite-observation event-opacity
k > *l* ≥1

Fig. 7.3 Relationship among infinite-observation, k-observation and l-observation event-opacity with $k>l\geq 1$

7.4.2 Related Notions: (K-step) Diagnosability

We observe that K-observation and infinite-observation event-opacity are closely related to (K-step) diagnosability [83]. Thus, this subsection is devoted to the analysis on their relationship. To this end, we recall the definitions of diagnosability and K-step diagnosability as follows.

Definition 7.3 [83]: A DFA $G=(X, \Sigma, \delta, x_0)$ is *diagnosable* w.r.t. a set of fault events $\Sigma_F \subseteq \Sigma_u$ if

$$(\forall \alpha \in \Psi(\Sigma_F))(\exists K \in \mathbb{Z}^+)(\forall \beta \in L(G)/\alpha: |P(\beta)| \geq K)$$
$$[\forall \sigma \in P_G^{-1}(P(\alpha\beta)), \sigma \cap \Sigma_F \neq \varnothing].$$

Definition 7.4 [83]: A DFA $G=(X, \Sigma, \delta, x_0)$ is *K-step diagnosable* w.r.t. a set of fault events $\Sigma_F \subseteq \Sigma_u$ if

$$(\forall \alpha \in \Psi(\Sigma_F))(\forall \beta \in L(G)/\alpha: |P(\beta)| \geq K)$$
$$[\forall \sigma \in P_G^{-1}(P(\alpha\beta)), \sigma \cap \Sigma_F \neq \varnothing].$$

For the purpose of comparison, we illustrate the definitions of diagnosability, K-step diagnosability, K-observation event-opacity and infinite-observation event-opacity in Fig. 7.4, where we only consider one fault-ended string $\alpha \in \Psi(\Sigma_F)$ or one secret-ended string $\alpha \in \Psi(\Sigma_S)$ and assume that three traces follow α. Note that every dot on the traces implies the occurrence of an observable event. Thus, every dot corresponds to an observation $w \in P(L(G))$. Besides, the following notations are introduced in Fig. 7.4 for intuition:

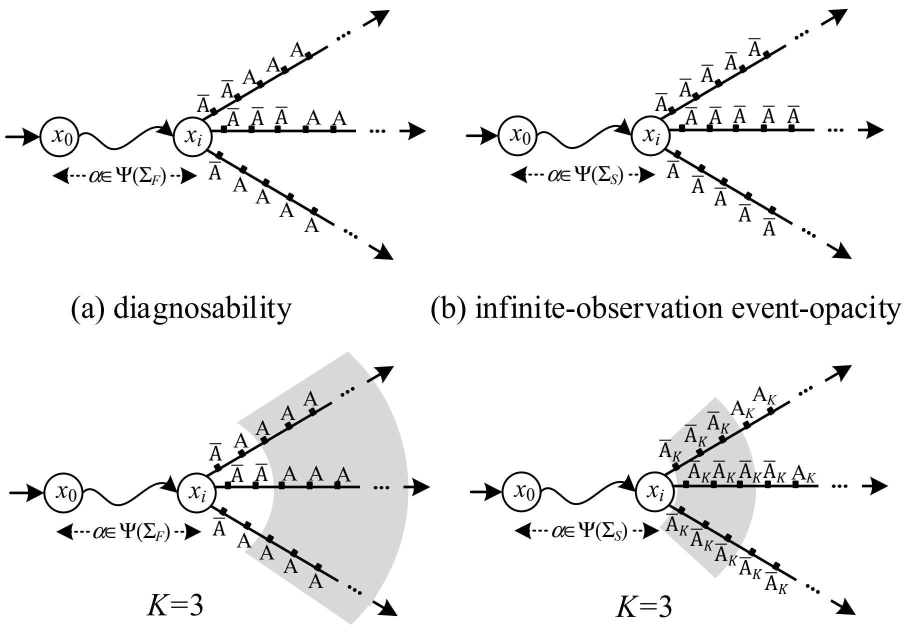
$$\mathbf{A}: [\forall \sigma \in P_G^{-1}(w), \sigma \cap \Sigma' \neq \varnothing];$$

$$\mathbf{A}_K: [\forall \sigma \in P_G^{-1}(w), sufob_{K+1}(\sigma) \cap \Sigma' \neq \varnothing],$$

where $\Sigma' \subseteq \Sigma_u$ is a set of fault events Σ_F or a set of secret events Σ_S depending on the context. Accordingly, it follows that

$$\bar{\mathbf{A}}: [\exists \sigma \in P_G^{-1}(w), \sigma \cap \Sigma' = \varnothing];$$

$$\bar{\mathbf{A}}_K: [\exists \sigma \in P_G^{-1}(w), sufob_{K+1}(\sigma) \cap \Sigma' = \varnothing].$$



(a) diagnosability (b) infinite-observation event-opacity

(c) K-step diagnosability (d) K-observation event-opacity

Fig. 7.4 Illustration of different notions

First, let us focus on the notions of diagnosabilty and infinite-observation event-opacity. We observe that diagnosability implies that, for any fault-ended string α and any of its following string $\beta\in L(G)/\alpha$, as long as β produces a sufficiently long observable word, $P(\alpha\beta)$ satisfies condition **A**, i.e., the observation $P(\alpha\beta)$ reveals the occurrence of a fault. In contrast, infinite-observation event-opacity implies that, for any secret-ended string α and any of its following string $\beta\in L(G)/\alpha$, the observation $P(\alpha\beta)$ can never satisfy condition **A**, i.e., $P(\alpha\beta)$ can never reveal the occurrence of a secret. Consequently, the following result holds.

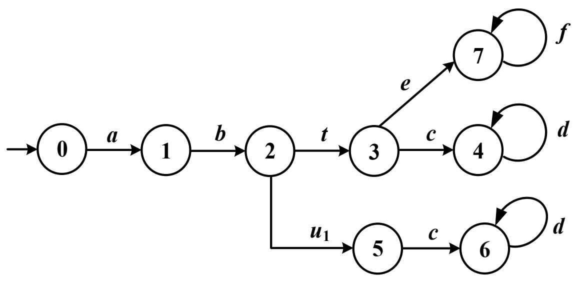
Proposition 7.3: Given a DFA G=(X, Σ, δ, x_0) and a set of events $\Sigma'\subseteq\Sigma_u$, it holds that

G is infinite-observation event-opaque w.r.t. Σ'

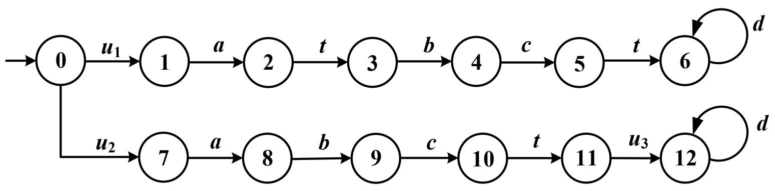
⇨ G is not diagnosable w.r.t. Σ'.

Proof: Straightforward from Definitions 7.2 and 7.3. ■

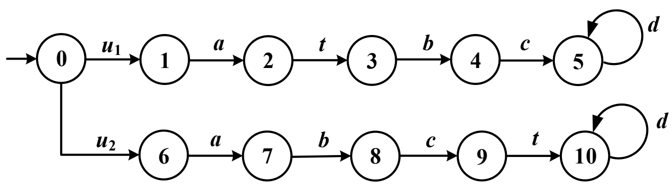
We observe that a DFA can be neither infinite-observation event-opaque nor diagnosable w.r.t. a set of events $\Sigma'\subseteq\Sigma_u$. For example, the DFA G=(X, Σ, δ, x_0) in Fig. 7.5 (a), where $\Sigma_o=\{a, b, c, d, e, f\}$ and $\Sigma_u=\{u_1, t\}$, is neither infinite-observation event-opaque nor diagnosable w.r.t. $\Sigma'=\{t\}$.



(a) Neither diagnosable nor infinite-observation event-opaque DFA w.r.t. Σ′={*t*}



(b) Neither 1-step diagnosable nor 1-observation event-opaque DFA w.r.t. Σ′={*t*}



(c) Both 1-observation event-opaque and diagnosable DFA w.r.t. Σ′={*t*}

Fig. 7.5 Examples of DFA with different properties

Now, looking at Fig. 7.4 (c) and (d), let us consider the relationship between *K*-step diagnosability and *K*-observation event-opacity. We observe that *K*-step diagnosability requires that every time a fault occurs, at the latest when the *K*-th observable event occurs after that, condition **A** is satisfied, i.e., it is revealed that a fault has occurred. In contrast, *K*-observation event-opacity requires that every time a secret occurs, within the occurrence of *K* observable events after that, condition $\mathbf{A}_K$ is never satisfied, i.e., it can never be established that a secret occurred in the middle of the past *K*+1 observable events. Note that *K*-observation event-opacity cares about the (*K*+1)-observation suffix of a string rather than the whole string, which is different from the other three notions. Indeed, the following result holds.

Proposition 7.4: Given a DFA $G=(X, \Sigma, \delta, x_0)$ and a set of events $\Sigma'\subseteq\Sigma_u$, it holds that

G is K-observation event-opaque w.r.t. Σ'

$\Rightarrow$ G is not K-step diagnosable w.r.t. Σ'.

Proof: Two cases related to $\Psi(\Sigma')$ should be distinguished: 1) $\exists\alpha\in\Psi(\Sigma')$, $|P(\alpha)|=0$; and 2) $\forall\alpha\in\Psi(\Sigma')$, $|P(\alpha)|\geq1$.

(*Case* 1) Let $\beta\in L(G)/\alpha$ such that $|P(\beta)|=K$. By Definition 7.1,

$$\exists\sigma\in P_G^{-1}(P(\alpha\beta)),\ sufob_{K+1}(\sigma)\cap\Sigma'=\varnothing.$$

It is easy to realize that $|P(\sigma)|=|P(\alpha\beta)|=K$. Thus, $\sigma=sufob_{K+1}(\sigma)$. As a result, $\sigma\cap\Sigma'=\varnothing$.

(*Case* 2) Let $\alpha\in\Psi(\Sigma')$ such that

$$\forall\alpha'\in\Psi(\Sigma'), P(\alpha')\notin \overline{P(\alpha)}\setminus\{P(\alpha)\}. \quad (1)$$

Let $\beta\in L(G)/\alpha$ such that $|P(\beta)|=K$. By Definition 7.1,

$$\exists\sigma\in P_G^{-1}(P(\alpha\beta)),\ sufob_{K+1}(\sigma)\cap\Sigma'=\varnothing. \quad (2)$$

Since $|P(\alpha)|\geq1$, it follows that $\exists\sigma_1, \sigma_2\in\Sigma^*$ and $e\in\Sigma_o$, such that $\sigma=\sigma_1e\sigma_2$ with $P(\sigma_2)=P(\beta)$ and $P(\sigma_1e)=P(\alpha)$. Thus, by (1), it holds that $\forall\alpha'\in\Psi(\Sigma')$, $P(\alpha')\notin\overline{P(\sigma_1)}$. This implies that $\sigma_1\cap\Sigma'=\varnothing$. In addition, it is $e\sigma_2=sufob_{K+1}(\sigma)$. By (2), it follows that $e\sigma_2\cap\Sigma'=\varnothing$. Hence, $\sigma\cap\Sigma'=\varnothing$.

Consequently, in any case it holds that

$$(\exists\alpha\in\Psi(\Sigma'))(\exists\beta\in L(G)/\alpha\colon |P(\beta)|=K)$$
$$[\exists\sigma\in P_G^{-1}(P(\alpha\beta)),\ \sigma\cap\Sigma'=\varnothing].$$

According to Definition 7.4, G is not K-step diagnosable w.r.t. Σ'. ■

We note that a DFA can be neither K-step diagnosable nor K-observation event-opaque w.r.t. a set of events $\Sigma'\subseteq\Sigma_u$. As an example, the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.5 (b), where $\Sigma_o=\{a, b, c, d\}$ and $\Sigma_u=\{u_1\text{-}u_3, t\}$, is neither 1-step diagnosable nor 1-observation event-opaque w.r.t. $\Sigma'=\{t\}$. In detail, if we focus on string $u_1at\in\Psi(\Sigma')$, the strings u_1atb and u_2ab share the same observation ab. Thus, G is not 1-step diagnosable w.r.t. Σ'. On the other hand, if we focus on string $u_1atbct\in\Psi(\Sigma')$, the strings $u_1atbctd$ and u_2abctu_3d have the same observation $abcd$ and their 2-observation suffixes, i.e., ctd and ctu_3d, both contain the event t. Thus, G is not 1-observation event-opaque w.r.t. Σ'.

As mentioned above, an infinite-observation event-opaque DFA is K-observation event-opaque. In addition, it is easy to see that a K-step diagnosable DFA is diagnosable. Besides, it could happen that a DFA is both K-observation event-opaque and diagonosable w.r.t. a set of events $\Sigma'\subseteq\Sigma_u$. As an example, the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.5 (c), where $\Sigma_o=\{a, b, c, d\}$ and $\Sigma_u=\{u_1, u_2, t\}$, is 1-observation event-opaque and diagonosable w.r.t. $\Sigma'=\{t\}$. It could also happen that a DFA is neither K-observation event-opaque nor diagonosable w.r.t. a set of events $\Sigma'\subseteq\Sigma_u$. Consider again the DFA in Fig. 7.5 (a). It is neither K-observation event-opaque nor diagonosable w.r.t. $\Sigma'=\{t\}$.

According to all the above analysis, the relationship among these notions can be summarized in the Venn diagram in Fig. 7.6 considering a given $K\in\mathbb{Z}^+$ and the same set $\Sigma'\subseteq\Sigma_u$.

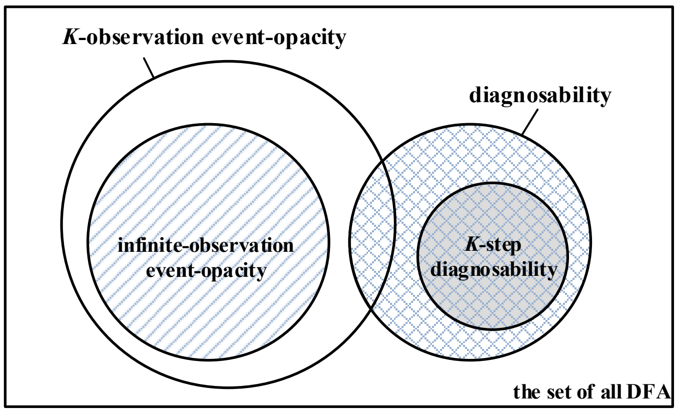

K-observation event-opacity
diagnosability
infinite-observation event-opacity
K-step diagnosability
the set of all DFA

Fig. 7.6 Relationship among infinite-observation event-opacity, K-observation event-opacity, diagnosability and K-step diagnosability

Remark 7.2: We notice that, in some papers, e.g. in [10], the definition of K-step diagnosability is slightly different from the one considered in this work. The difference is that K in [10] counts the number of events instead of the number of observable events that occur after a fault. In this work, we focus on the definition where K refers to the number of observable events that occur after a fault. By the way, it is clear that K in the definition of K-observation event-opacity refers to the number of observable events that occur after a secret. In order to avoid ambiguity, we name it K-observation event-opacity rather than K-step event-opacity. ♣

7.4.3 Event-opacity and Combinational Event-opacity

In this subsection, we propose generalized notions of event-based opacity considering more general scenarios by relaxing assumptions A1 and A2 made before.

First, we consider the scenario where assumption A1 is relaxed such that the critical horizons of different secret events are allowed to be different. Hence, we introduce a function $\mathcal{K}$: $\Sigma_S \rightarrow \mathbb{Z}^+ \cup \{+\infty\}$ that associates with each secret event a positive integer that may go to infinity, characterizing the critical horizon of the secret event. In this scenario, we propose the notion of *event-opacity* that generalizes the notion of *K*-observation/infinite-observation event-opacity. Note that we assume $sufob_{+\infty}(\sigma)=\sigma$ for $\sigma \in \Sigma^*$.

Definition 7.5: A DFA $G=(X, \Sigma, \delta, x_0)$ is said to be *event-opaque* w.r.t. $\Sigma_S \subseteq \Sigma_u$ and $\mathcal{K}$: $\Sigma_S \rightarrow \mathbb{Z}^+ \cup \{+\infty\}$ if

$$(\forall w \in P(L(G)))(\exists \sigma \in P_G^{-1}(w))$$
$$[\forall s \in \Sigma_S,\ \ sufob_{\mathcal{K}(s)+1}(\sigma) \cap \{s\} = \varnothing].$$

Event-opacity implies that for any observation from the DFA, there exists at least one consistent string such that any secret event s is not contained in the ($\mathcal{K}(s)$+1)-observation suffix of the consistent string. In terms of the motivational example, this means that every day the competitor gets the observation on issued public decisions, there exists a possibility that none of the secret decisions is within its critical horizon. Thus, the "conservative" competitor never takes an action, which guarantees the security of the decision-making model.

Furthermore, we consider a more general scenario by relaxing assumption A2 as well such that the adversary may distinguish among secret events. The physical meaning behind it is that the competitor needs to take different actions for different classes of secret decisions.

In this scenario, we assume that there is a partition $\mathcal{P}_m$ of the set of secret events Σ_S such that Σ_S is partitioned into $m\in\mathbb{Z}^+$ disjoint sets, that is,

$$\Sigma_S = \Sigma_S{}^1 \dot{\cup} \Sigma_S{}^2 \dot{\cup} \ldots \dot{\cup} \Sigma_S{}^m .$$

Accordingly, we denote $\mathcal{K}^1$, $\mathcal{K}^2$, …, $\mathcal{K}^m$ the restrictions of $\mathcal{K}$: $\Sigma_S\rightarrow\mathbb{Z}^+\cup\{+\infty\}$ to $\Sigma_S{}^1$, $\Sigma_S{}^2$, …, $\Sigma_S{}^m$, respectively. Then, we introduce the following notion.

Definition 7.6: Given a DFA $G=(X, \Sigma, \delta, x_0)$, a set of secret events $\Sigma_S\subseteq\Sigma_u$, $\mathcal{K}$: $\Sigma_S\rightarrow\mathbb{Z}^+\cup\{+\infty\}$, and a partition $\mathcal{P}_m$ of Σ_S, G is said to be *combinational event-opaque* w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_m)$ if it is event-opaque w.r.t. $\Sigma_S{}^i$ and $\mathcal{K}^i$, $\forall i\in\{1, 2, \ldots, m\}$, i.e.,

$$(\forall i\in\{1, 2, \ldots, m\})(\forall w\in P(L(G)))(\exists\sigma\in P_G{}^{-1}(w))$$
$$[\forall s\in\Sigma_S{}^i,\ sufob_{\mathcal{K}^i(s)+1}(\sigma)\cap\{s\}=\varnothing].$$

The combinational event-opacity w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_m)$ implies that for each class $\Sigma_S{}^i$, $i\in\{1, 2, \ldots, m\}$, every observation corresponds to a possibility that none of the secrets in the class $\Sigma_S{}^i$ is within its critical horizon. In the motivation example, this means that it could happen that when getting an observation, the competitor knows for sure that a secret decision is within its critical horizon but it cannot argue to which class it belongs. For example, Table 7.1 shows a case related to an observation $w\in P(L(G))$, where G is a combinational event-opaque DFA w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_2)$. There are three consistent strings corresponding to w. We observe that the competitor knows for sure that a secret decision among one of the classes $\Sigma_S{}^1$ and $\Sigma_S{}^2$ is now within its critical horizon but it cannot know to which class it belongs based on the observation w. In the case that different classes of secret decisions require different actions, the "conservative" competitor still cannot take an action. Thus, the combinational event-opacity implies the security of the decision-making model.

Table 7.1 A case related to $w \in P(L(G))$, where G is a combinational event-opaque DFA w.r.t. (Σ_S, $\mathcal{K}$, $\mathcal{P}_2$)

	$P_G^{-1}(w)$	Σ_S^1	Σ_S^2
$w \in P(L(G))$	σ_1	√	×
	σ_2	×	√
	σ_3	√	√

* "√" denotes $\exists s \in \Sigma_S^i$, $sufob_{\mathcal{K}^i(s)+1}(\sigma) \cap \{s\} \neq \varnothing$ *"×" denotes $\forall s \in \Sigma_S^i$, $sufob_{\mathcal{K}^i(s)+1}(\sigma) \cap \{s\} = \varnothing$

Result 7.3: Given a DFA $G=(X, \Sigma, \delta, x_0)$, a set of secret events $\Sigma_S \subseteq \Sigma_u$, $\mathcal{K}$: $\Sigma_S \rightarrow \mathbb{Z}^+ \cup \{+\infty\}$ and a partition $\mathcal{P}_m$ of Σ_S,

G is event-opaque w.r.t. Σ_S and $\mathcal{K}$

$\Rightarrow$ G is combinational event-opaque w.r.t. (Σ_S, $\mathcal{K}$, $\mathcal{P}_m$).

Proof: It trivially holds by Definitions 7.5 and 7.6. ■

Result 7.3 indicates that the event-opacity w.r.t. Σ_S and $\mathcal{K}$ typically characterizes a stronger security requirement than the combinational event-opacity w.r.t. (Σ_S, $\mathcal{K}$, $\mathcal{P}_m$).

7.5 Verification Methods

In this section, we propose methods for the verification of the proposed notions. In particular, we first introduce the verification method for event-opacity in an exhaustive manner, which is then reduced to be applied to the verification of K-observation and infinite-observation event-opacity and modified to be applied to the verification of combinational event-opacity.

7.5.1 Verification of Event-opacity

We propose a solution based on the construction of the *event-opacity verifier*, which is a DFA where every node consists of *compound states* of the considered system. We introduce the notion of compound states as well as related notations as follows and then formally define the verifier.

Given a DFA $G=(X, \Sigma, \delta, x_0)$, a *compound state* of G is $x_c \in \Sigma^* \times X \times La$, where $La=\{$"$\varnothing$", "S"$\}$. In other words, a compound state is a triple consisting of an event string, a state and a label that is either "$\varnothing$" or "S". The label "$\varnothing$" (resp., "S") indicates that its associated state is reachable from the initial state of G via a string without (resp., with) secret events.

Given a set of secret events $\Sigma_S'\subseteq\Sigma_S$, we define the *label evolution function* as $\eta_{\Sigma_S'}$: $La\times\Sigma^*\rightarrow La$ such that $\forall l\in La$, $\forall\sigma\in\Sigma^*$,

$$\eta_{\Sigma_S'}(l,\sigma)=\begin{cases}"\varnothing" & if \quad l="\varnothing"\wedge\sigma\cap\Sigma_S'=\varnothing\\ "S" & \quad otherwise\end{cases}.$$

Note that it holds $\eta_{\Sigma_S'}(l,\sigma_1\sigma_2)=\eta_{\Sigma_S'}(\eta_{\Sigma_S'}(l,\sigma_1),\sigma_2)$, $\forall l\in La$, $\forall\sigma_1$, $\sigma_2\in\Sigma^*$.

Given a set of compound states $X_c\in 2^{\Sigma^*\times X\times La}$ and a set of secret events $\Sigma_S'\subseteq\Sigma_S$, we define the set of compound states that are unobservably reachable from some compound state in X_c as

$UR_{\Sigma_S'}(X_c)=\{(\sigma', x', l')\in\Sigma^*\times X\times La\mid$

$\exists(\sigma, x, l)\in X_c$, $\exists u\in\Sigma_u^*$, s.t. $\delta(x, u)=x'\wedge\sigma u=\sigma'\wedge l'=\eta_{\Sigma_S'}(l,u)\}$.

We define the set of observable events that are enabled at the set X_c as

$$En_o(X_c)=\{v\in\Sigma_o\mid\exists(\sigma, x, l)\in X_c, \text{ s.t. } \delta(x, v)!\}.$$

Moreover, we define the set of compound states that are reachable from some compound state in X_c via an enabled observable event $v\in En_o(X_c)$ as

$$Next(X_c, v)=\{(\sigma', x', l')\in\Sigma^*\times X\times La\mid\exists(\sigma, x, l)\in X_c, \text{ s.t. } \delta(x, v)=x'\wedge\sigma v=\sigma'\wedge l'=l\}.$$

Example 7.4: Consider the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.7, where $\Sigma_o=\{a, b, c, d, e\}$, $\Sigma_u=\{u_1\text{-}u_3, s_1\text{-}s_5\}$, $\Sigma_S=\{s_1\text{-}s_5\}$, and a set of compound states $X_c=\{(a, 1, \varnothing), (u_3a, 13, \varnothing)\}$. It is $UR_{\Sigma_S'}(X_c)=\{(a, 1, \varnothing), (au_1, 4, \varnothing), (au_2, 8, \varnothing), (u_3a, 13, \varnothing), (u_3as_3, 14, S)\}$ given $\Sigma_S'=\{s_3, s_5\}$ and it is $UR_{\Sigma_S'}(X_c)=\{(a, 1, \varnothing), (au_1, 4, \varnothing), (au_2, 8, \varnothing), (u_3a, 13, \varnothing), (u_3as_3, 14, \varnothing)\}$ given $\Sigma_S'=\{s_5\}$. Moreover, $En_o(X_c)=\{c\}$ and $Next(X_c, c)=\{(ac, 2, \varnothing)\}$. ♦

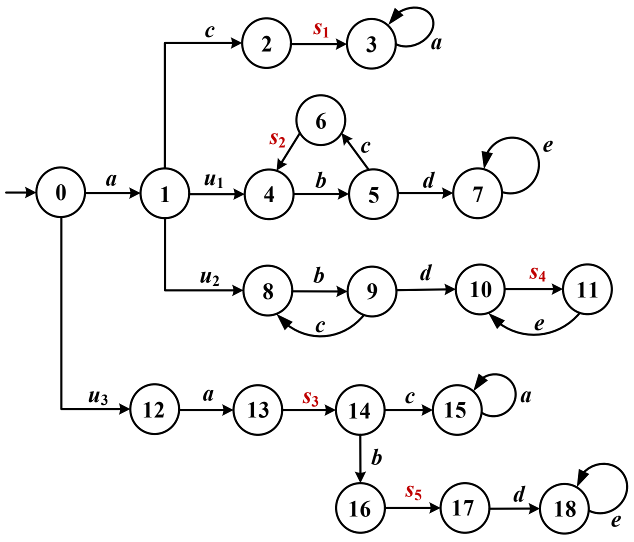


Fig. 7.7 DFA G with $\Sigma_S=\{s_1\text{-}s_5\}$

In the remainder of the chapter, without loss of generality, we assume that, under the function $\mathcal{K}$: $\Sigma_S \to \mathbb{Z}^+ \cup \{+\infty\}$, Σ_S can be divided into n+1 categories with different critical horizons, that is,

$$\Sigma_S = \Sigma_{S1} \dot{\cup} \Sigma_{S2} \dot{\cup} \ldots \dot{\cup} \Sigma_{S(n)} \dot{\cup} \Sigma_{S(+\infty)},$$

with $\forall s \in \Sigma_{Si}$, $\mathcal{K}(s)=K_i \in \mathbb{Z}^+$, $i \in \{1, 2, \ldots, n\}$, and $\forall s \in \Sigma_{S(+\infty)}$, $\mathcal{K}(s)=+\infty$. Moreover, we denote $K_{max}=\max\{K_1, K_2, \ldots, K_n\}$. Note that we specify $K_{max}=-1$ in the special case that $\Sigma_S=\Sigma_{S(+\infty)}$. In addition, we define $sufob_0(\sigma)=\epsilon$, $\forall \sigma \in \Sigma^*$. Now, we are ready to introduce the event-opacity verifier.

Definition 7.7: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S \subseteq \Sigma_u$ and $\mathcal{K}$: $\Sigma_S \to \mathbb{Z}^+ \cup \{+\infty\}$, the *event-opacity verifier* is defined as a DFA

$$Ver^{\mathcal{K}}(G)=(Z, \Sigma_o, \delta_Z, z_0),$$

where

- $Z \subseteq 2^{\Sigma^* \times X \times La}$;
- $z_0=\{(sufob_{K_{max}+1}(\sigma), x, l) \mid (\sigma, x, l) \in UR_{\Sigma_{S(+\infty)}}(\{(\epsilon, x_0, \varnothing)\})\}$;
- δ_Z: $Z \times \Sigma_o \to Z$ is the transition function such that

  $\forall z \in Z$, $\forall v \in \Sigma_o$, if $v \in En_o(z)$, $\delta_Z(z, v)$ is defined such that

$$\delta_Z(z, v)=\{(sufob_{Kmax+1}(\sigma), x, l) \mid (\sigma, x, l)\in UR_{\Sigma_{S(+\infty)}}(Next(z,v))\}.$$

$Ver^{\mathcal{K}}(G)$ is defined only considering the accessible part from its initial state.

In other words, the verifier $Ver^{\mathcal{K}}(G)$ can be constructed as follows. First, the initial node $z_0=\{(sufob_{Kmax+1}(\sigma), x, l) \mid (\sigma, x, l)\in UR_{\Sigma_{S(+\infty)}}(\{(\epsilon,x_0,\varnothing)\})\}$ is created. Normally, in the case that $K_{max}\neq -1$, it is $z_0=UR_{\Sigma_{S(+\infty)}}(\{(\epsilon,x_0,\varnothing)\})$. Next, for each enabled observable event $v\in En_o(z_0)$, we compute the next node $z=\{(sufob_{Kmax+1}(\sigma), x, l) \mid (\sigma, x, l)\in UR_{\Sigma_{S(+\infty)}}(Next(z_0,v))\}$. Specifically, we first compute $UR_{\Sigma_{S(+\infty)}}(Next(z_0,v))$ and then update each string in it by keeping its (K_{max}+1)-observation suffix only, which results in node z. If node z already exists, we add a transition from node z_0 to node z via event v; otherwise, we create node z together with a transition from node z_0 to node z via event v and then for each enabled observable event $v\in En_o(z)$, the above procedure is repeated to generate nodes. Finally, the event-opacity verifier is constructed.

Remark 7.3: The event-opacity verifier contains a finite number of states. This is because we only keep the (K_{max}+1)-observation suffix for any event string in the verifier and the considered system does not contain any cycle with unobservable events only. ♣

Example 7.5: Consider the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.7, where $\Sigma_o=\{a, b, c, d, e\}$, $\Sigma_u=\{u_1\text{-}u_3, s_1\text{-}s_5\}$ and $\Sigma_S=\{s_1\text{-}s_5\}$. Given a function $\mathcal{K}$: $\Sigma_S\rightarrow \mathbb{Z}^+\cup\{+\infty\}$ such that

$$\mathcal{K}(s_1)=1,\ \mathcal{K}(s_2)=\mathcal{K}(s_4)=2,\ \mathcal{K}(s_3)=\mathcal{K}(s_5)=+\infty,$$

we construct its event-opacity verifier w.r.t. Σ_S and $\mathcal{K}$.

It is K_{max}=2 and $\Sigma_{S(+\infty)}=\{s_3, s_5\}$. Thus, when constructing the event-opacity verifier, we always keep the 3-observation suffix of every computed string and update labels considering secret events s_3 and s_5. In more detail, we first create the initial node $z_0=\{(\epsilon, 0, \varnothing), (u_3, 12, \varnothing)\}$. Since $En_o(z_0)=\{a\}$, we compute $Next(z_0, a)=\{(a, 1, \varnothing), (u_3a, 13, \varnothing)\}$ and then node $z_1=UR_{\{s_3,s_5\}}(Next(z_0, a))=\{(a, 1, \varnothing), (au_1, 4, \varnothing), (au_2, 8, \varnothing), (u_3a,$

13, ∅), (u_3as_3, 14, S)} is created with a transition from z_0 to z_1 via a. Similarly, nodes are created one by one, resulting in the event-opacity verifier $Ver^{\mathcal{K}}(G)$ in Fig. 7.10. ♦

We note that the event-opacity verifier $Ver^{\mathcal{K}}(G)$, if the string and label components are ignored in each node, coincides with the *observer* [15] of G or is the state space refinement of the observer of G. In either case, it implies that the language of the verifier is the same as the set of all observations produced by G. Moreover, in each node of the verifier, each compound state is computed considering a possible consistent string corresponding to an observation leading to the node. Specifically, the compound state consists of a "string component" that is the (K_{max}+1)-observation suffix of the considered string, a "state component" that is the state of G reached by the string, and a "label component" that reflects the information on whether a secret event in $\Sigma_{S(+\infty)}$ occurred in the string. Such properties of the event-opacity verifier are formalized as follows.

Lemma 7.1: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S\subseteq\Sigma_u$, $\mathcal{K}$: $\Sigma_S\rightarrow \mathbb{Z}^+\cup\{+\infty\}$, and the event-opacity verifier $Ver^{\mathcal{K}}(G)=(Z, \Sigma_o, \delta_Z, z_0)$, it holds that

1) $\forall w\in L(Ver^{\mathcal{K}}(G))$,

$$\delta_Z(z_0, w)=\{(sufob_{K_{max}+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\emptyset,\sigma))\mid \sigma\in P_G^{-1}(w)\};$$

2) $P(L(G))=L(Ver^{\mathcal{K}}(G))$.

Proof: See the Appendix. ■

Given an observation, the event-opacity verifier allows us to determine whether a secret event in Σ_{S1}, Σ_{S2}, …, $\Sigma_{S(n)}$ is in its critical horizon by checking the "string components" of the corresponding node and determine whether a secret event in $\Sigma_{S(+\infty)}$ occurred by checking the "label components" of the node. Consequently, a method is developed for the verification of event-opacity, which is indicated by the following result. In words, we can verify the event-opacity by checking if there exists a node in the verifier where for every compound state, either it contains a label "S" or there exists

$i\in\{1, 2, \ldots, n\}$ such that the (K_i+1)-observation suffix of its "string component" contains a secret event in Σ_{Si}.

Theorem 7.1: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S\subseteq\Sigma_u$, $\mathcal{K}$: $\Sigma_S\rightarrow \mathbb{Z}^+\cup\{+\infty\}$, and the event-opacity verifier $Ver^{\mathcal{K}}(G)=(Z, \Sigma_o, \delta_Z, z_0)$,

G is not event-opaque w.r.t. Σ_S and $\mathcal{K}$

$$\Leftrightarrow(\exists z\in Z)(\forall(\sigma, x, l)\in z)$$
$$[(l=\text{“}S\text{”}) \vee (\exists i\in\{1, 2, \ldots, n\}: sufob_{Ki+1}(\sigma)\cap\Sigma_{Si}\neq\varnothing)]$$

Proof: (=>) By Definition 7.5, it is

$$(\exists w\in P(L(G)))(\forall\sigma\in P_G^{-1}(w))\,[\exists s\in\Sigma_S,\ sufob_{\mathcal{K}(s)+1}(\sigma)\cap\{s\}\neq\varnothing]. \qquad (3)$$

Since $P(L(G))=L(Ver^{\mathcal{K}}(G))$ by Lemma 7.1, we can find a node $z\in Z$, s.t. $\delta_Z(z_0, w)=z$. Moreover, it is

$$z=\{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma))|\ \sigma\in P_G^{-1}(w)\}.$$

Thus, due to (3), $\forall(\sigma', x, l)\in z$, $[(l=\text{“}S\text{”}) \vee (\exists i\in\{1, 2, \ldots, n\}: sufob_{Ki+1}(\sigma')\cap\Sigma_{Si}\neq\varnothing)]$.

(<=) There exists a string $w\in L(Ver^{\mathcal{K}}(G))$ such that $\delta_Z(z_0, w)=z$. By Lemma 7.1, it is

$$z=\{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma))|\ \sigma\in P_G^{-1}(w)\}.$$

Since $\forall(\sigma', x, l)\in z$, $[(l=\text{“}S\text{”}) \vee (\exists i\in\{1, 2, \ldots, n\}: sufob_{Ki+1}(\sigma')\cap\Sigma_{Si}\neq\varnothing)]$, it follows that $\forall\sigma\in P_G^{-1}(w)$, $[\exists s\in\Sigma_S,\ sufob_{\mathcal{K}(s)+1}(\sigma)\cap\{s\}\neq\varnothing]$. Since $P(L(G))=L(Ver^{\mathcal{K}}(G))$, it holds that

$$(\exists w\in P(L(G)))(\forall\sigma\in P_G^{-1}(w))[\exists s\in\Sigma_S,\ sufob_{\mathcal{K}(s)+1}(\sigma)\cap\{s\}\neq\varnothing].$$

Thus, G is not event-opaque w.r.t. Σ_S and $\mathcal{K}$ by Definition 7.5. ■

Example 7.5 (Continued): We verify whether the DFA G in Fig. 7.7 is event-opaque w.r.t. Σ_S and $\mathcal{K}$ based on its event-opacity verifier $Ver^{\mathcal{K}}(G)$ in Fig. 7.10. We can see that there is a node (in grey) where one compound state is with label "S" and the other satisfies that s_1 is included in the $(\mathcal{K}(s_1)+1)$-observation suffix of its string component.

By Theorem 7.1, we conclude that G is not event-opaque w.r.t. Σ_S and $\mathcal{K}$. Indeed, when we observe $w=aca$, we know for sure that a secret event (s_1 or s_3) is in its critical horizon. ♦

Remark 7.4: We analyze the complexity of the proposed verification method. It is $Z \subseteq 2^{\Sigma^*_{K\max+1} \times X \times La}$, where $\Sigma^*_{K\max+1}$ denotes the set of ($K_{\max}$+1)-observation suffixes of strings in Σ^*. Suppose that at most $a \in \mathbb{Z}^+$ continuous unobservable events appear in strings in $L(G)$. Then, the verifier $Ver^{\mathcal{K}}(G)$ contains at most $4^{(|\Sigma_o|+1)^{K\max+1} \times (|\Sigma_u|+1)^{a \times (K\max+1)} \times |X|}$ states and $|\Sigma_o| \times 4^{(|\Sigma_o|+1)^{K\max+1} \times (|\Sigma_u|+1)^{a \times (K\max+1)} \times |X|}$ transitions. The complexity of detecting a state in $Ver^{\mathcal{K}}(G)$ is linear with respect to the number of states of $Ver^{\mathcal{K}}(G)$. Thus, the overall complexity of the proposed verification method is

$$\mathcal{O}(|\Sigma_o| \times 4^{(|\Sigma_o|+1)^{K\max+1} \times (|\Sigma_u|+1)^{a \times (K\max+1)} \times |X|}).$$ ♣

Remark 7.5: In the case that $\Sigma_{S(+\infty)}=\varnothing$, i.e., there is no secret event in Σ_S whose critical horizon is infinite, the "label components" can be omitted when we construct the event-opacity verifier. In the case that $\Sigma_S=\Sigma_{S(+\infty)}$, i.e., every secret event in Σ_S has infinite critical horizon, the "string components" can be omitted. The second case is exactly the verification of infinite-observation event-opacity, which is discussed in more detail in the next subsection. ♣

7.5.2 Verification of *K*-observation and Infinite-observation Event-opacity

We know that K-observation and infinite-observation event-opacity are special cases of event-opacity. Thus, in this subsection, we show verification methods of such properties consisting in a reduction of the verification method for event-opacity.

Let us first focus on infinite-observation event-opacity. It holds that $\Sigma_S=\Sigma_{S(+\infty)}$. Thus, "string components" are omitted when we construct the event-opacity verifier. We rename the verifier as *infinite-observation event-opacity verifier*, denoted as $Ver^{\infty}(G)=(Q, \Sigma_o, \delta_Q, q_0)$. Then, we have the following corollary from Theorem 7.1.

Corollary 7.1: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S \subseteq \Sigma_u$ and the infinite-observation event-opacity verifier $Ver^{\infty}(G)=(Q, \Sigma_o, \delta_Q, q_0)$,

$$G \text{ is not infinite-observation event-opaque w.r.t. } \Sigma_S \Leftrightarrow \exists\, q \in Q, \text{ s.t. } \forall (x, l) \in q, l=\text{“}S\text{”}.$$

In words, the infinite-observation event-opacity can be verified by checking if there exists a node in the verifier where every label is “S”.

Consider K-observation event-opacity. It is the case that $\Sigma_{S(+\infty)}=\varnothing$ and $K_{\max}=K$. Thus, “label components” are omitted in the event-opacity verifier and the “string components” are the (K+1)-observation suffixes of all computed strings. We rename the verifier as *K-observation event-opacity verifier*, denoted as $Ver^K(G)=(Y, \Sigma_o, \delta_Y, y_0)$. Then, we have the following corollary from Theorem 7.1.

Corollary 7.2: Given a DFA $G=(X, \Sigma, \delta, x_0)$ with a set of secret events $\Sigma_S \subseteq \Sigma_u$, $K \in \mathbb{Z}^+$, and the K-observation event-opacity verifier $Ver^K(G)=(Y, \Sigma_o, \delta_Y, y_0)$,

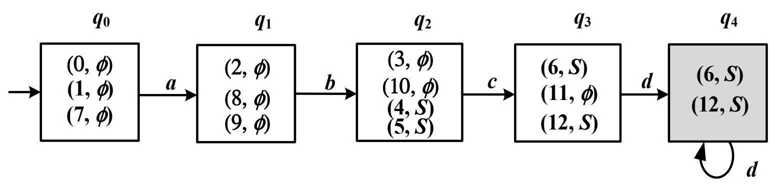
$$G \text{ is not } K\text{-observation event-opaque w.r.t. } \Sigma_S \Leftrightarrow \exists\, y \in Y, \text{ s.t. } \forall (\sigma, x) \in y, \sigma \cap \Sigma_S \neq \varnothing.$$

In words, K-observation event-opacity can be verified by checking if there exists a node in the verifier where every string contains a secret event.

Example 7.6: Consider again the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.1, where $\Sigma_o=\{a, b, c, d\}$, $\Sigma_u=\{u_1\text{-}u_4, s\}$ and $\Sigma_S=\{s\}$.

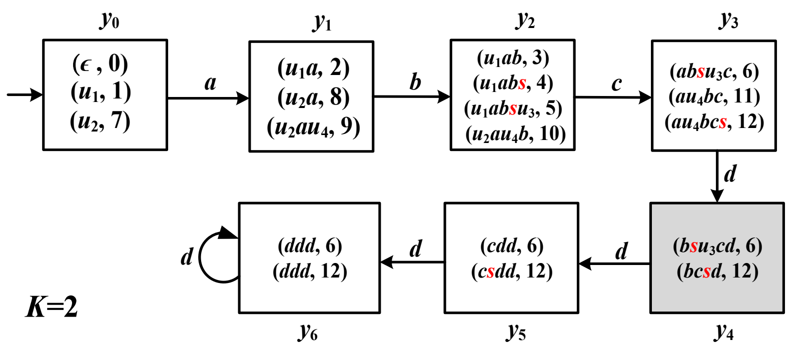
1) We verify if it is infinite-observation event-opaque w.r.t. Σ_S. Then, we construct its infinite-observation event-opacity verifier $Ver^{\infty}(G)=(Q, \Sigma_o, \delta_Q, q_0)$ shown in Fig. 7.8. We can see that there exists a node q_4 where the labels are both “S”. Thus, G is not infinite-observation event-opaque w.r.t. Σ_S.

2) We verify if it is 2-observation event-opaque w.r.t. Σ_S. Then, we construct its 2-observation event-opacity verifier $Ver^2(G)=(Y, \Sigma_o, \delta_Y, y_0)$ shown in Fig. 7.9. We can see that there exists a node y_4 where both strings contain a secret event. Thus, G is not 2-observation event-opaque w.r.t. Σ_S. ♦



q_0
(0, ϕ) (1, ϕ) (7, ϕ)
a
q_1
(2, ϕ) (8, ϕ) (9, ϕ)
b
q_2
(3, ϕ) (10, ϕ) (4, S) (5, S)
c
q_3
(6, S) (11, ϕ) (12, S)
d
q_4
(6, S) (12, S)
d

Fig. 7.8 Infinite-observation event-opacity verifier of G in Fig. 7.1



y_0
(ϵ, 0) (u_1, 1) (u_2, 7)
a
y_1
(u_1a, 2) (u_2a, 8) (u_2au_4, 9)
b
y_2
(u_1ab, 3) (u_1abs, 4) (u_1absu_3, 5) (u_2au_4b, 10)
c
y_3
($absu_3c$, 6) (au_4bc, 11) (au_4bcs, 12)
d
y_4
(bsu_3cd, 6) ($bcsd$, 12)
d
y_5
(cdd, 6) ($csdd$, 12)
d
y_6
(ddd, 6) (ddd, 12)
d
K=2

Fig. 7.9 2-observation event-opacity verifier of G in Fig. 7.1

Remark 7.6: The infinite-observation event-opacity verifier is similar to the *diagnoser* [83]. However, in the diagnoser, "A" is introduced as a label, indicating the "ambiguous" situation of a state, while in the infinite-observation event-opacity verifier, no ambiguous state is present but it could happen that there are two labeled states only differing for their labels even in the same node. ♣

7.5.3 Verification of Combinational Event-opacity

Combinational event-opacity can be verified by checking the event-opacity of each class of secret events, which means that it can be verified based on the construction of multiple event-opacity verifiers. In this subsection, we introduce a verification method for combinational event-opacity based on a verifier, which differs from the event-opacity verifier for the extension of the label set.

Specifically, we extend the label set such that $La=2^{\{S^1,S^2,\ldots,S^m\}}$, where each S^i indicates a class of secret events. Accordingly, given a set of secret events $\Sigma_S{}'\subseteq\Sigma_S$ with

$\Sigma_S'=\Sigma_S'^{(1)}\cup\Sigma_S'^{(2)}\cup\ldots\cup\Sigma_S'^{(m)}$, the *label evolution function* is updated as $\eta_{\Sigma_S'}$: $La\times\Sigma^*\to La$ such that

$$\forall l\in La,\ \forall\sigma\in\Sigma^*,\ \eta_{\Sigma_S'}(l,\sigma)=l\cup\{S^i:\Sigma_S'^{(i)}\cap\sigma\neq\varnothing\}.$$

By using the updated label set and the label evolution function, the verifier in Definition 7.7 can be used to verify the combinational event-opacity of G w.r.t. (Σ_S, $\mathcal{K}$, $\mathcal{P}_m$). We denote the verifier as $Ver^{\mathcal{K},\mathcal{P}_m}(G)$ with no need to formalize its definition.

In order to present the verification method, we assume that, without loss of generality, each class Σ_S^i, $i\in\{1, 2, \ldots, m\}$ under the function $\mathcal{K}$: $\Sigma_S\to\mathbb{Z}^+\cup\{+\infty\}$, is divided into n^i+1 categories with different critical horizons, that is,

$$\Sigma_S^i=\Sigma_{S1}^i\dot{\cup}\Sigma_{S2}^i\dot{\cup}\ldots\dot{\cup}\Sigma_{S(n^i)}^i\dot{\cup}\Sigma_{S(+\infty)}^i,$$

where $\forall s\in\Sigma_{Sj}^i$, $\mathcal{K}(s)=K_j^i\in\mathbb{Z}^+$, $j\in\{1, 2, \ldots, n^i\}$, and $\forall s\in\Sigma_{S(+\infty)}^i$, $\mathcal{K}(s)=+\infty$.

Theorem 7.2: Given a DFA $G=(X, \Sigma, \delta, x_0)$, a set of secret events $\Sigma_S\subseteq\Sigma_u$, $\mathcal{K}$: $\Sigma_S\to\mathbb{Z}^+\cup\{+\infty\}$, a partition $\mathcal{P}_m$ of Σ_S, and the verifier $Ver^{\mathcal{K},\mathcal{P}_m}(G)=(Z, \Sigma_o, \delta_Z, z_0)$,

G is not combinational event-opaque w.r.t. (Σ_S, $\mathcal{K}$, $\mathcal{P}_m$)

$$\Leftrightarrow(\exists i\in\{1, 2, \ldots, m\})(\exists z\in Z)(\forall(\sigma, x, l)\in z)$$
$$[(S^i\in l)\vee(\exists j\in\{1, 2, \ldots, n^i\}:\ sufob_{K_j^i+1}(\sigma)\cap\Sigma_{Sj}^i\neq\varnothing)].$$

Proof: Similar to Lemma 7.1, it is $P(L(G))=L(Ver^{\mathcal{K},\mathcal{P}_m}(G))$ and $\forall w\in L(Ver^{\mathcal{K},\mathcal{P}_m}(G))$, $\delta_Z(z_0, w)=\{(sufob_{K_{max}+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma))|\ \sigma\in P_G^{-1}(w)\}$. Thus, the theorem can be proved similarly to Theorem 7.1 based on Definition 7.6. ■

In words, the combinational event-opacity can be verified by checking if there is a class i of secret events and there is a node in the verifier $Ver^{\mathcal{K},\mathcal{P}_m}(G)$ such that for every compound state in the node, either it contains a label indicating the class i or there exists a subclass Σ_{Sj}^i such that the (K_j^i+1)-observation suffix of its "string component" contains a secret event in Σ_{Sj}^i.

Example 7.7: Recall Example 7.5, where we consider the DFA $G=(X, \Sigma, \delta, x_0)$ in Fig. 7.7 with $\Sigma_o=\{a, b, c, d, e\}$, $\Sigma_u=\{u_1\text{-}u_3, s_1\text{-}s_5\}$, $\Sigma_S=\{s_1\text{-}s_5\}$, and $\mathcal{K}$: $\Sigma_S\rightarrow \mathbb{Z}^+\cup\{+\infty\}$ such that

$$\mathcal{K}(s_1)=1,\ \mathcal{K}(s_2)=\mathcal{K}(s_4)=2,\ \mathcal{K}(s_3)=\mathcal{K}(s_5)=+\infty.$$

Now, suppose that there exists a partition $\mathcal{P}_2$ of Σ_S such that $\Sigma_S=\Sigma_S{}^1\dot{\cup}\Sigma_S{}^2$, where $\Sigma_S{}^1=\{s_1, s_4, s_5\}$ and $\Sigma_S{}^2=\{s_2, s_3\}$. We verify the combinational event-opacity of G w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_2)$.

Since there are two classes of secret events, the label set is now $La=2^{\{S^1,S^2\}}$. Specifically, "S^1" records the occurrence of s_5 since $\Sigma_S{}^1{}_{(+\infty)}=\{s_5\}$ and "S^2" records the occurrence of s_3 since $\Sigma_S{}^2{}_{(+\infty)}=\{s_3\}$. The combinational event-opacity verifier $Ver^{\mathcal{K},\mathcal{P}_2}(G)$ is shown in Fig. 7.11. Indeed, except the label components, it is the same as $Ver^{\mathcal{K}}(G)$ in Fig. 7.10. Looking at the node reached by the observation *aca*, it reveals that a secret event is in its critical horizon, but it does not reveal which class it is. In other words, neither for class 1 nor 2, the node satisfies the violation condition of combinational event-opacity in Theorem 7.2. Moreover, it can be checked that, no matter for class 1 or 2, none of nodes in $Ver^{\mathcal{K},\mathcal{P}_2}(G)$ satisfies the violating condition in Theorem 7.2. Consequently, G is combinational event-opaque w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_2)$.

Considering the practical scenario, the example together with Example 7.5 implies that in the case that the competitor has to distinguish between two sets of secret events $\Sigma_S{}^1=\{s_1, s_4, s_5\}$ and $\Sigma_S{}^2=\{s_2, s_3\}$, the security of the decision-making model is guaranteed, while in the case that the competitor does not distinguish among secret events, the security of the decision-making model is not guaranteed. ♦

Remark 7.7: Since $La=2^{\{S^1,S^2,\ldots,S^m\}}$, the complexity of the proposed verification method for combinational event-opacity is $\mathcal{O}(|\Sigma_o|\times 2^{(|\Sigma_o|+1)^{K\max+1}\times(|\Sigma_u|+1)^{a\times(K\max+1)}\times|X|\times 2^m})$.

♣

Remark 7.8: The verification method can be simplified for special cases.

Case 1: Secret events in each class Σ_S^i have the same critical horizon. Without loss of generality, we assume that there are $g\in\mathbb{Z}$ $(0\leq g\leq m)$ classes of secret events with infinite critical horizon and others with finite critical horizon. Moreover, it is assumed that $\forall i\in\{1, 2, \ldots, g\}$, $\forall s\in\Sigma_S^i$, $\mathcal{K}(s)=+\infty$ and $\forall i\in\{g, g+1, \ldots, m\}$, $\forall s\in\Sigma_S^i$, $\mathcal{K}(s)=K^i\in\mathbb{Z}^+$. Then, based on the verifier $Ver^{\mathcal{K},\mathcal{P}_m}(G)$, the following implication holds:

G is not combinational event-opaque w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_m)$

$$\Leftrightarrow(\exists i\in\{1, 2, \ldots, g\})(\exists z\in Z)(\forall(\sigma, x, l)\in z)[(S^i\in l)]$$
$$\vee(\exists i\in\{g, g+1, \ldots, m\})(\exists z\in Z)(\forall(\sigma, x, l)\in z)\,[(sufob_{K^i+1}(\sigma)\cap\Sigma_S^{\,i}\neq\varnothing)].$$

Case 2: All secret events in Σ_S have the same critical horizon and it is $K\in\mathbb{Z}^+$. Then, based on the verifier $Ver^{\mathcal{K},\mathcal{P}_m}(G)$, the following implication holds:

G is not combinational event-opaque w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_m)$

$$\Leftrightarrow(\exists i\in\{1, 2, \ldots, m\})(\exists z\in Z)(\forall(\sigma, x, l)\in z)\,[sufob_{K+1}(\sigma)\cap\Sigma_S^{\,i}\neq\varnothing)].$$

Case 3: All secret events in Σ_S have the same critical horizon and it is $K=+\infty$. Then, based on the verifier $Ver^{\mathcal{K},\mathcal{P}_m}(G)$, the following implication holds:

G is not combinational event-opaque w.r.t. $(\Sigma_S, \mathcal{K}, \mathcal{P}_m)$

$$\Leftrightarrow(\exists i\in\{1, 2, \ldots, m\})(\exists z\in Z)(\forall(\sigma, x, l)\in z)\,[(S^i\in l)].$$

Finally, in the case that $\Sigma_{S(+\infty)}=\varnothing$, i.e., there is no secret event in Σ_S whose critical horizon is infinite, the "label components" can be omitted in $Ver^{\mathcal{K},\mathcal{P}_m}(G)$ and in the case that $\Sigma_S=\Sigma_{S(+\infty)}$, i.e., all secret events in Σ_S are with infinite critical horizon, the "string components" can be omitted in $Ver^{\mathcal{K},\mathcal{P}_m}(G)$. ♣

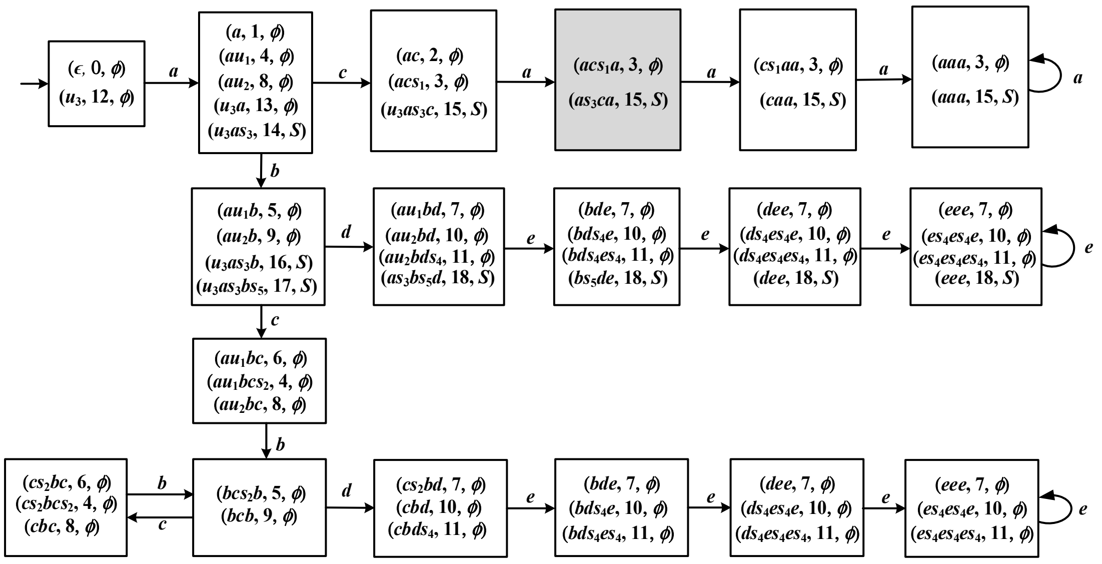


Fig. 7.10 Verifier $Ver^{\mathcal{K}}(G)$ in Example 7.5

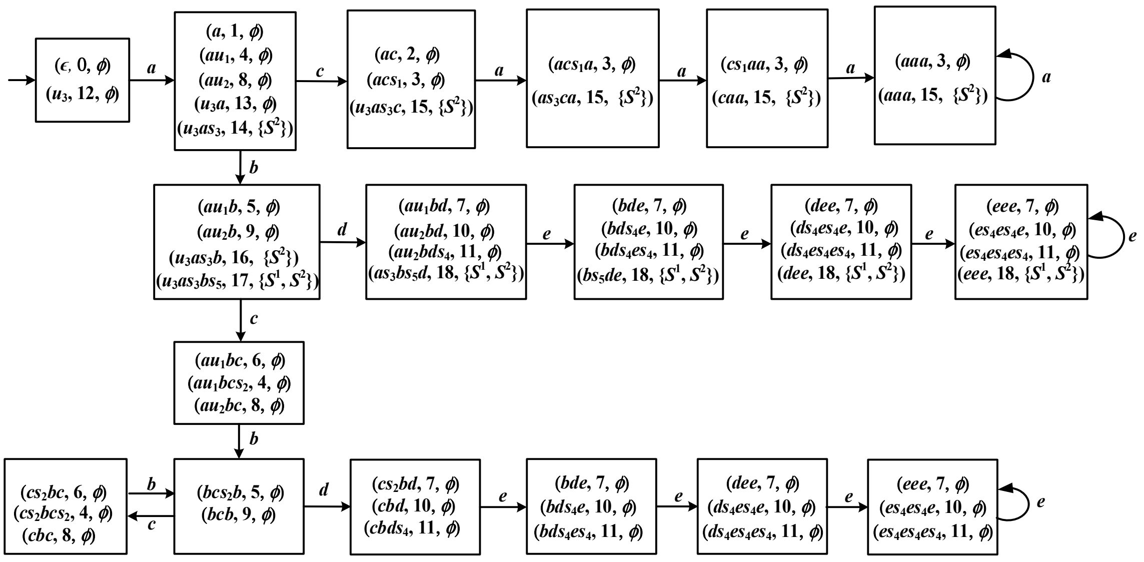


Fig. 7.11 Verifier $Ver^{\mathcal{K},\mathcal{P}_2}(G)$ in Example 7.7

7.6 Conclusions

In this chapter, new notions of opacity, namely, *K-observation event-opacity*, *infinite-observation event-opacity*, *event-opacity* and *combinational event-opacity*, are proposed. The relationship between *K*-observation/infinite-observation event-opacity and (*K*-step) diagnosabililty is presented. Event-opacity is a notion generalized from *K*-observation/infinite-observation event-opacity considering that different secret events may have different critical horizons and the combinational event-opacity is proposed in the case that the intruder distinguishes among secret events. Appropriate verifiers are introduced, which allow us to verify the proposed properties by checking their nodes.

The work of this chapter is now under review by *IEEE Transactions on Automatic Control*.

7.7 Appendix: Proof of Lemma 7.1

1) First, we consider $\epsilon \in L(Ver^{\mathcal{K}}(G))$. Clearly, $|P_G^{-1}(\epsilon)| \neq \varnothing$. By Definition 7.7, $\delta_Z(z_0, \epsilon) = z_0 = \{(sufob_{Kmax+1}(\sigma), x, l) | (\sigma, x, l) \in UR_{\Sigma_{S(+\infty)}}(\{(\epsilon, x_0, \varnothing)\})\}$. Thus, it is trivial to see that

$$\delta_Z(z_0, \epsilon) = \{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma)) | \sigma \in P_G^{-1}(\epsilon)\}.$$

Next, we consider $w' \in L(Ver^{\mathcal{K}}(G))$ with $w' = wv$, where $v \in \Sigma_o$ and $w \in L(Ver^{\mathcal{K}}(G))$ satisfying $|P_G^{-1}(w)| \neq \varnothing$ and

$$\delta_Z(z_0, w) = \{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma)) | \sigma \in P_G^{-1}(w)\}. \quad (4)$$

Clearly, $\delta_Z(z_0, w) \neq \varnothing$. Let $z = \delta_Z(z_0, w)$. By Definition 7.7, $v \in En_o(z)$ and

$$\delta_Z(z, v) = \{(sufob_{Kmax+1}(\sigma), x, l) | (\sigma, x, l) \in UR_{\Sigma_{S(+\infty)}}(Next(z, v))\}.$$

Due to (4), since $v \in En_o(z)$, it follows that

$$\exists \sigma \in P_G^{-1}(w), \text{ s.t. } \delta(x_0, \sigma v)! \quad (5)$$

and it holds that

$$Next(z, v) = \{(sufob_{Kmax+1}(\sigma)v, \delta(x_0, \sigma v), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma v)) | \sigma \in P_G^{-1}(w), \delta(x_0, \sigma v)!\}.$$

Furthermore, it is

$$UR_{\Sigma_{S(+\infty)}}(Next(z, v)) = \{(sufob_{Kmax+1}(\sigma)vu, \delta(x_0, \sigma vu), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma vu)) | \sigma \in P_G^{-1}(w), u \in \Sigma_u^*, \delta(x_0, \sigma vu)!\}.$$

Trivially, it is

$$\delta_Z(z, v) = \{(sufob_{Kmax+1}(\sigma vu), \delta(x_0, \sigma vu), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma vu)) | \sigma \in P_G^{-1}(w), u \in \Sigma_u^*, \delta(x_0, \sigma vu)!\}. \quad (6)$$

Note that it holds

$$P_G^{-1}(w') = P_G^{-1}(wv) = \{\sigma vu | \sigma \in P_G^{-1}(w), u \in \Sigma_u^*, \delta(x_0, \sigma vu)!\}. \quad (7)$$

Thus, $|P_G^{-1}(w')| \neq \varnothing$ due to (5). In addition, by (6) and (7), it follows

$$\delta_Z(z, v) = \{(sufob_{Kmax+1}(\sigma'), \delta(x_0, \sigma'), \eta_{\Sigma_{S(+\infty)}}(\varnothing, \sigma')) | \sigma' \in P_G^{-1}(w')\}.$$

Since $z = \delta_Z(z_0, w)$ and $w' = wv$, it is

$$\delta_Z(z_0, w')=\delta_Z(z, v)=\{(sufob_{Kmax+1}(\sigma'), \delta(x_0, \sigma'), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma'))|\sigma'\in P_G^{-1}(w')\}.$$

Consequently, we conclude that $\forall w\in L(Ver^{\mathcal{K}}(G))$, $|P_G^{-1}(w)|\neq\varnothing$ and

$$\delta_Z(z_0, w)=\{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma))|\ \sigma\in P_G^{-1}(w)\}.$$

2) It is proved that $\forall w\in L(Ver^{\mathcal{K}}(G))$, $|P_G^{-1}(w)|\neq\varnothing$. It means $\forall w\in L(Ver^{\mathcal{K}}(G))$, $w\in P(L(G))$. Thus, $L(Ver^{\mathcal{K}}(G))\subseteq P(L(G))$. Next, we prove $L(Ver^{\mathcal{K}}(G))\supseteq P(L(G))$.

First, we consider $\epsilon\in P(L(G))$. It is clear that $\epsilon\in L(Ver^{\mathcal{K}}(G))$.

Next, we consider $w'\in P(L(G))$ s.t. $w'=wv$, where $v\in\Sigma_o$ and $w\in P(L(G))\wedge w\in L(Ver^{\mathcal{K}}(G))$. We prove $w'\in L(Ver^{\mathcal{K}}(G))$.

Since $wv\in P(L(G))$, $w\in P(L(G))$ and $v\in\Sigma_o$, it holds that

$$\exists\sigma\in P_G^{-1}(w), \text{ s.t. } \sigma v\in L(G). \tag{8}$$

Since $w\in L(Ver^{\mathcal{K}}(G))$, there exists $z\in Z$ s.t.

$$z=\delta_Z(z_0, w)=\{(sufob_{Kmax+1}(\sigma), \delta(x_0, \sigma), \eta_{\Sigma_{S(+\infty)}}(\varnothing,\sigma))|\ \sigma\in P_G^{-1}(w)\}. \tag{9}$$

Due to (8) and (9), it is $v\in En_o(z)$. Thus, $w'\in L(Ver^{\mathcal{K}}(G))$ since $w\in L(Ver^{\mathcal{K}}(G))$.

Consequently, it holds that $\forall w\in P(L(G))$, $w\in L(Ver^{\mathcal{K}}(G))$, i.e., $L(Ver^{\mathcal{K}}(G))\supseteq P(L(G))$.

Therefore, $P(L(G))=L(Ver^{\mathcal{K}}(G))$. ■

CHAPTER VIII

Conclusions and Future Work

8.1 Conclusions

This thesis includes two topics on partially-observed discrete event systems (DES), namely, supervisory control and analysis.

In the topic of supervisory control, Petri nets (PN) are used as a reference formalism. First, we investigate the forbidden state problem in PN with both unobservable and uncontrollable transitions, assuming that unobservable transitions are uncontrollable. An optimal on-line control policy with polynomial complexity is proposed for ordinary PN subject to an admissible Generalized Mutual Exclusion Constraint (GMEC) such that the observation subnet satisfies certain conditions in structure. How to obtain an optimal on-line control policy for PN subject to an arbitrary GMEC is also discussed. Next, the forbidden state problem in PN vulnerable to sensor-reading disguising attacks (SD-attacks) is studied. Given a control specification in terms of a GMEC, three methods to derive on-line control policies are provided. Finally, we still consider PN vulnerable to SD-attacks but deal with the liveness-enforcing problem. In this problem, by restricting the plant within a bounded system, an off-line method that synthesizes a liveness-enforcing supervisor tolerant to an SD-attack is proposed.

In the topic of analysis, we focus on two properties that are related to system security, namely, fault-predictability and event-based opacity. In the framework of labeled PN, a verification approach for fault-predictability by characterizing the structure of the constructed Predictor Graph is developed together with two rules that reduce the size of a PN without changing its fault-predictability. In the framework of deterministic finite-state automata, we propose four notions of event-based opacity, namely, *K*-observation event-opacity, infinite-observation event-opacity, event-opacity and combinational event-opacity. Moreover, appropriate verifiers are proposed to verify the proposed four properties.

8.2 Future Work

We present the future work of the thesis in terms of the following four parts.

8.2.1 Forbidden State Problem of DES With Unobservable and Uncontrollable Events

The control policy in Chapter III has limited application due to restrictive assumptions made on the structures of PN. In order to extend the control policy to more general PN, the future work may be focused on the following two issues:

1) study the efficient computation of the unobservable minimal decrease in PN with more general structures; and

2) develop efficient techniques to equivalently transform an arbitrary GMEC into an admissible one for more general PN structures.

8.2.2 Supervisory Control of DES Vulnerable to Network Attacks

In this thesis, we consider sensor-reading disguising attacks (SD-attacks) only and assume that transitions in the considered PN are all controllable and observable. Thus, one future direction is extending the proposed methods in Chapters IV and V to more general types of attacks including erasing and/or adding observations and to more general systems containing unobservable and/or uncontrollable transitions. Concerning the liveness-enforcing problem studied in Chapter V, the computed liveness-enforcing supervisor is not guaranteed to be maximally permissive. Thus, in the future work, we plan to investigate how to guarantee the solution to be maximally permissive. Furthermore, we are interested in extending the proposed method to unbounded systems based on the construction of coverability graphs.

8.2.3 Fault-predictability

As a future work we first plan to use the recent results on reachability analysis of unbounded PN in [27, 52] to reduce the complexity of verifying fault-predictability. Besides, we plan to study on-line fault prediction, namely we want to design an efficient tool that, once an observation is done, allows us to conclude if a fault will occur for sure, or it will not occur for sure, or we are in an uncertain condition. Finally, we may study the problem of fault-predictability enforcement, namely, given a system that is fault-unpredictable, we want to modify the labeling function (adding new sensors) to guarantee that the resulting system is fault-predictable.

8.2.4 Event-based Opacity

In the future work, we plan to study the enforcement of the proposed properties, considering the supervisory control and sensor activation techniques, respectively. Also, we are interested in extending the properties to distributed and decentralized settings.

BIBLIOGRAPHY

[1] Z. Achour, N. Rezg, and X. Xie, "Supervisory control of partially observable marked graphs," *IEEE Transactions on Automatic control,* vol. 49, no. 11, pp. 2007-2011, 2004.

[2] R. Ammour, E. Leclercq, E. Sanlaville, and D. Lefebvre, "Fault prognosis of timed stochastic discrete event systems with bounded estimation error," *Automatica,* vol. 82, pp. 35-41, 2017.

[3] R. Ammour, E. Leclercq, E. Sanlaville, and D. Lefebvre, "Faults prognosis using partially observed stochastic petri-nets: An incremental approach," *Discrete Event Dynamic Systems,* vol. 28, no. 2, pp. 247-267, 2018.

[4] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau, "Concurrent secrets," *Discrete Event Dynamic Systems,* vol. 17, no. 4, pp. 425-446, 2007, doi: 10.1007/s10626-007-0020-5.

[5] K. Barkaoui and J.-F. Pradat-Peyre, "On liveness and controlled siphons in petri nets," in *International Conference on Application and Theory of Petri Nets*, 1996: Springer, pp. 57-72.

[6] F. Basile, M. P. Cabasino, and C. Seatzu, "Diagnosability analysis of labeled time petri net systems," *IEEE Transactions on Automatic Control,* vol. 62, no. 3, pp. 1384-1396, 2016.

[7] F. Basile, P. Chiacchio, and A. Giua, "Suboptimal supervisory control of petri nets in presence of uncontrollable transitions via monitor places," *Automatica,* vol. 42, no. 6, pp. 995-1004, 2006.

[8] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *International Journal of Information Security,* vol. 7, no. 6, pp. 421-435, 2008, doi: 10.1007/s10207-008-0058-x.

[9] J. W. Bryans, M. Koutny, and P. Y. A. Ryan, "Modelling opacity using petri nets," *Electronic Notes in Theoretical Computer Science,* vol. 121, pp. 101-115, 2005.

[10] M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu, "A new approach for diagnosability analysis of petri nets using verifier nets," *IEEE Transactions on Automatic Control,* vol. 57, no. 12, pp. 3104-3117, 2012, doi: 10.1109/tac.2012.2200372.

[11] M. P. Cabasino, A. Giua, and C. Seatzu, "Fault detection for discrete event systems using petri nets with unobservable transitions," *Automatica,* vol. 46, no. 9, pp. 1531-1539, 2010.

[12] M. P. Cabasino, C. N. Hadjicostis, and C. Seatzu, "Marking observer in labeled petri nets with application to supervisory control," *IEEE Transactions on Automatic Control,* vol. 62, no. 4, pp. 1813-1824, 2017.

[13] K. Cai, R. Zhang, and W. M. Wonham, "Relative observability of discrete-event systems and its supremal sublanguages," *IEEE Transactions on Automatic Control,* vol. 60, no. 3, pp. 659-670, 2014.

[14] L. K. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune, "Detection and mitigation of classes of attacks in supervisory control systems," *Automatica,* vol. 97, pp. 121-133, 2018.

[15] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2008.

[16] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods in System Design,* vol. 40, no. 1, pp. 88-115, 2012.

[17] F. Cassez and A. Grastien, "Predictability of event occurrences in timed systems," in *International Conference on Formal Modeling and Analysis of Timed Systems*, 2013: Springer, pp. 62-76.

[18] E. Chanthery and P. Ribot, "An integrated framework for diagnosis and prognosis of hybrid systems," *arXiv preprint arXiv:1308.5332,* 2013.

[19] C. Chen and H. Hu, "Liveness-enforcing supervision in ams-oriented hamgs: An approach based on new characterization of siphons using petri nets," *IEEE Transactions on Automatic Control,* vol. 63, no. 7, pp. 1987-2002, 2017.

[20] J. Chen and R. Kumar, "Stochastic failure prognosability of discrete event systems," *IEEE Transactions on Automatic Control,* vol. 60, no. 6, pp. 1570-1581, 2014.

[21] Y. Chen and Z. Li, "Design of a maximally permissive liveness-enforcing supervisor with a compressed supervisory structure for flexible manufacturing systems," *Automatica,* vol. 47, no. 5, pp. 1028-1034, 2011.

[22] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE transactions on automatic control,* vol. 33, no. 3, pp. 249-260, 1988.

[23] D. E. Comer and R. E. Droms, *Computer networks and internets*. Prentice-Hall, Inc., 2003.

[24] X. Cong, M. P. Fanti, A. M. Mangini, and Z. Li, "Decentralized diagnosis by petri nets and integer linear programming," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 48, no. 10, pp. 1689-1700, 2017.

[25] P. Darondeau, H. Marchand, and L. Ricker, "Enforcing opacity of regular predicates on modal transition systems," *Discrete Event Dynamic Systems,* vol. 25, no. 1-2, pp. 251-270, 2014.

[26] V. Deverakonda and R. Sreenivas, "On a sufficient information structure for supervisory policies that enforce liveness in a class of general petri nets," *IEEE Transactions on Automatic Control,* vol. 7, no. 60, pp. 1915-1920, 2015.

[27] Z. Ding, M. Pan, R. Yang, C. Jiang, and M. Zhou, "Fully expanded tree for property analysis of one-place-unbounded petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 47, no. 9, pp. 2574-2585, 2016.

[28] J. Dubreil, P. Darondeau, and H. Marchand, "Opacity enforcing control synthesis," in *the 9th international workshop on discrete event systems (WODES)*, 2008, pp. 28–35, doi: 10.1109/WODES.2008.4605918.

[29] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory control for opacity," *IEEE Transactions on Automatic Control,* vol. 55, no. 5, pp. 1089-1100, 2010.

[30] Y. Falcone and H. Marchand, "Enforcement and validation (at runtime) of various notions of opacity," *Discrete Event Dynamic Systems: Theory & Applications,* pp. 1-40, 2014.

[31] R. Fritz and P. Zhang, "Modeling and detection of cyber attacks on discrete event systems," *IFAC-PapersOnLine,* vol. 51, no. 7, pp. 285-290, 2018.

[32] S. Genc and S. Lafortune, "Distributed diagnosis of place-bordered petri nets," *IEEE Transactions on Automation science and Engineering,* vol. 4, no. 2, pp. 206-219, 2007.

[33] S. Genc and S. Lafortune, "Predictability of event occurrences in partially-observed discrete-event systems," *Automatica,* vol. 45, no. 2, pp. 301-311, 2009.

[34] A. Giua, F. DiCesare, and M. Silva, "Generalized mutual exclusion contraints on nets with uncontrollable transitions," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, Chicago, IL, USA, 1992: IEEE, pp. 974-979.

[35] R. M. Góes, E. Kang, R. Kwong, and S. Lafortune, "Stealthy deception attacks for cyber-physical systems," in *IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017: IEEE, pp. 4224-4230.

[36] C. Gu, Z. Li, N. Wu, M. Khalgui, T. Qu, and A. Al-Ahmari, "Improved multi-step look-ahead control policies for automated manufacturing systems," *IEEE Access,* vol. 6, pp. 68824-68838, 2018.

[37] N. B. Hadj-Alouane, S. Lafortune, and F. Lin, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *Discrete Event Dynamic Systems,* vol. 6, no. 4, pp. 379-427, 1996.

[38] L. E. Holloway, B. H. Krogh, and A. Giua, "A survey of petri net methods for controlled discrete event systems," *Discrete Event Dynamic Systems,* vol. 7, no. 2, pp. 151-190, 1997.

[39] B. Huang, M. Zhou, Y. Huang, and Y. Yang, "Supervisor synthesis for fms based on critical activity places," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 49, no. 5, pp. 881-890, 2017.

[40] M. V. Iordache and P. J. Antsaklis, "Synthesis of supervisors enforcing general linear constraints in petri nets," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL,* vol. 48, no. 11, pp. 2036-2039, 2003.

[41] R. Jacob, J.-J. Lesage, and J.-M. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annual Reviews in Control,* vol. 41, pp. 135-146, 2016, doi: 10.1016/j.arcontrol.2016.04.015.

[42] T. Jéron, H. Marchand, S. Genc, and S. Lafortune, "Predictability of sequence patterns in discrete event systems," *IFAC Proceedings Volumes,* vol. 41, no. 2, pp. 537-543, 2008.

[43] Y. Ji, X. Yin, and S. Lafortune, "Enforcing opacity by insertion functions under multiple energy constraints," *Automatica,* vol. 108, 2019, doi: 10.1016/j.automatica.2019.06.028.

[44] Y. Ji, X. Yin, and S. Lafortune, "Opacity enforcement using nondeterministic publicly-known edit functions," *IEEE Transactions on Automatic Control,* pp. 1-1, 2019, doi: 10.1109/tac.2019.2897553.

[45] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control,* vol. 46, no. 8, pp. 1318-1321, 2001.

[46] G. Jiroveanu and R. K. Boel, "A distributed approach for fault detection and diagnosis based on time petri nets," *Mathematics and computers in simulation,* vol. 70, no. 5-6, pp. 287-313, 2006.

[47] C. Keroglou and C. N. Hadjicostis, "Probabilistic system opacity in discrete event systems," *Discrete Event Dynamic Systems,* vol. 28, no. 2, pp. 289-314, 2018.

[48] A. Khoumsi and H. Chakib, "Conjunctive and disjunctive architectures for decentralized prognosis of failures in discrete-event systems," *IEEE transactions on automation science and engineering,* vol. 9, no. 2, pp. 412-417, 2012.

[49] B. H. Krogh and L. E. Holloway, "Synthesis of feedback control logic for discrete manufacturing systems," *Automatica,* vol. 27, no. 4, pp. 641-651, 1991.

[50] R. Kumar and S. Takai, "Decentralized prognosis of failures in discrete event systems," *IEEE Transactions on Automatic Control,* vol. 55, no. 1, pp. 48-59, 2010, doi: 10.1109/tac.2009.2034216.

[51] D. Lefebvre, "Fault diagnosis and prognosis with partially observed stochastic petri nets," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability,* vol. 228, no. 4, pp. 382-396, 2014.

[52] J. Li, X. Yu, M. Zhou, and X. Dai, "Lean reachability tree for unbounded petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 48, no. 2, pp. 299-308, 2016.

[53] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on petri nets—a literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 42, no. 4, pp. 437-462, 2012.

[54] Y. E. Lien, "Termination properties of generalized petri nets," *SIAM Journal on Computing,* vol. 5, no. 2, pp. 251-265, 1976.

[55] P. M. Lima, "Security against network attacks in supervisory control systems," Master's thesis, Federal University of Rio de Janeiro, 2017.

[56] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, "Security against communication network attacks of cyber-physical systems," *Journal of Control, Automation and Electrical Systems,* vol. 30, no. 1, pp. 125-135, 2019.

[57] P. M. Lima, L. K. Carvalho, and M. V. Moreira, "Detectable and undetectable network attack security of cyber-physical systems," *IFAC-PapersOnLine,* vol. 51, no. 7, pp. 179-185, 2018.

[58] F. Lin, "Opacity of discrete event systems and its applications," *Automatica,* vol. 47, no. 3, pp. 496-503, 2011.

[59] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Information sciences,* vol. 44, no. 3, pp. 173-198, 1988.

[60] J. Luo and K. Nonami, "Approach for transforming linear constraints on petri nets," *IEEE Transactions on Automatic Control,* vol. 56, no. 12, pp. 2751-2765, 2011.

[61] J. Luo, H. Shao, K. Nonami, and F. Jin, "Maximally permissive supervisor synthesis based on a new constraint transformation method," *Automatica,* vol. 48, no. 6, pp. 1097-1101, 2012.

[62] J. Luo, W. Wu, H. Su, and J. Chu, "Supervisor synthesis for enforcing a class of generalized mutual exclusion constraints on petri nets," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans,* vol. 39, no. 6, pp. 1237-1246, 2009.

[63] J. Luo and M. Zhou, "Petri-net controller synthesis for partially controllable and observable discrete event systems," *IEEE Transactions on Automatic Control,* vol. 62, no. 3, pp. 1301-1313, 2017.

[64] Z. Ma, Z. Li, and A. Giua, "Characterization of admissible marking sets in petri nets with conflicts and synchronizations," *IEEE Transactions on Automatic Control,* vol. 62, no. 3, pp. 1329-1341, 2017, doi: 10.1109/tac.2016.2585647.

[65] Z. Ma, Y. Tong, Z. Li, and A. Giua, "Basis marking representation of petri net reachability spaces and its application to the reachability problem," *IEEE Transactions on Automatic Control,* vol. 62, no. 3, pp. 1078-1093, 2017, doi: 10.1109/tac.2016.2574120.

[66] A. Madalinski and V. Khomenko, "Predictability verification with parallel ltl-x model checking based on petri net unfoldings," *IFAC Proceedings Volumes,* vol. 45, no. 20, pp. 1232-1237, 2012.

[67] L. Mazar´e, "Using unification for opacity properties," presented at the WITS: 4, 2004. [Online]. Available: http://www- verimag.imag.fr/TR/TR- 2004- 24.pdf.

[68] J. O. Moody and P. J. Antsaklis, "Petri net supervisors for des with uncontrollable and unobservable transitions," *IEEE Transactions on Automatic Control,* vol. 45, no. 3, pp. 462-476, 2000.

[69] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE,* vol. 77, no. 4, pp. 541-580, 1989.

[70] F. Nouioua, P. Dague, and L. Ye, "Predictability in probabilistic discrete event systems," in *International Conference on Soft Methods in Probability and Statistics*, 2016: Springer, pp. 381-389.

[71] A. Philippot, M. Sayed-Mouchaweh, and V. Carré-Ménétrier, "Discrete event model-based approach for fault detection and isolation of manufacturing systems," *IFAC Proceedings Volumes,* vol. 42, no. 5, pp. 69-74, 2009.

[72] N. Ran, H. Su, A. Giua, and C. Seatzu, "Codiagnosability analysis of bounded petri nets," *IEEE Transactions on Automatic Control,* vol. 63, no. 4, pp. 1192-1199, 2017.

[73] N. Ran, H. Su, and S. Wang, "An improved approach to test diagnosability of bounded petri nets," *IEEE/CAA Journal of Automatica Sinica,* vol. 4, no. 2, pp. 297-303, 2017.

[74] L. Rozé and M.-O. Cordier, "Diagnosing discrete-event systems: Extending the “diagnoser approach” to deal with telecommunication networks," *Discrete Event Dynamic Systems,* vol. 12, no. 1, pp. 43-81, 2002.

[75] Y. Ru, M. P. Cabasino, A. Giua, and C. N. Hadjicostis, "Supervisor synthesis for discrete event systems under partial observation and arbitrary forbidden state specifications," *Discrete Event Dynamic Systems,* vol. 24, no. 3, pp. 275-307, 2014.

[76] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *Decision and Control, 2007 46th IEEE Conference on*, 2007, pp. 5056–5061, doi: 10.1109/CDC.2007.4434515.

[77] A. Saboori and C. N. Hadjicostis, "Verification of k-step opacity and analysis of its complexity," *IEEE Transactions on Automation Science and Engineering,* vol. 8, no. 3, pp. 549-559, 2011, doi: 10.1109/tase.2011.2106775.

[78] A. Saboori and C. N. Hadjicostis, "Opacity-enforcing supervisory strategies via state estimator constructions," *IEEE Transactions on Automatic Control,* vol. 57, no. 5, pp. 1155-1165, 2012, doi: 10.1109/tac.2011.2170453.

[79] A. Saboori and C. N. Hadjicostis, "Verification of infinite-step opacity and complexity considerations," *IEEE Transactions on Automatic Control,* vol. 57, no. 5, pp. 1265-1269, 2012, doi: 10.1109/tac.2011.2173774.

[80] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of discrete event systems," *Information Sciences,* vol. 246, pp. 115-132, 2013, doi: 10.1016/j.ins.2013.05.033.

[81] A. Saboori and C. N. Hadjicostis, "Current-state opacity formulations in probabilistic finite automata," *IEEE Transactions on Automatic Control,* vol. 59, no. 1, pp. 120-133, 2014, doi: 10.1109/tac.2013.2279914.

[82] M. Sampath, A. Godambe, E. Jackson, and E. Mallow, "Combining qualitative & quantitative reasoning-a hybrid approach to failure diagnosis of industrial systems," *IFAC Proceedings Volumes,* vol. 33, no. 11, pp. 487-494, 2000.

[83] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on automatic control,* vol. 40, no. 9, pp. 1555-1575, 1995.

[84] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE transactions on control systems technology,* vol. 4, no. 2, pp. 105-124, 1996.

[85] R. S. Sreenivas, "On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled petri nets," *IEEE Transactions on Automatic Control,* vol. 42, no. 7, pp. 928-945, 1997.

[86] R. Su, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations," *Automatica,* vol. 94, pp. 35-44, 2018.

[87] S. Takai, "Robust prognosability for a set of partially observed discrete event systems," *Automatica,* vol. 51, pp. 123-130, 2015.

[88] S. Takai and R. Kumar, "Verification and synthesis for secrecy in discrete-event systems," in *2009 American Control Conference*, 2009: IEEE, pp. 4741-4746.

[89] S. Takai and R. Kumar, "Distributed failure prognosis of discrete event systems with bounded-delay communications," *IEEE Transactions on Automatic Control,* vol. 57, no. 5, pp. 1259-1265, 2011.

[90] S. Takai and T. Ushio, "Effective computation of an lm (g)-closed, controllable, and observable sublanguage arising in supervisory control," *Systems & Control Letters,* vol. 49, no. 3, pp. 191-200, 2003.

[91] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing,* vol. 1, no. 2, pp. 146-160, 1972.

[92] D. Thorsley and D. Teneketzis, "Intrusion detection in controlled discrete event systems," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006: IEEE, pp. 6047-6054.

[93] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Current-state opacity enforcement in discrete event systems under incomparable observations," *Discrete Event Dynamic Systems,* vol. 28, no. 2, pp. 161-182, 2017, doi: 10.1007/s10626-017-0264-7.

[94] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Decidability of opacity verification problems in labeled petri net systems," *Automatica,* vol. 80, pp. 48-53, 2017, doi: 10.1016/j.automatica.2017.01.013.

[95] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Verification of state-based opacity using petri nets," *IEEE Transactions on Automatic Control,* vol. 62, no. 6, pp. 2823-2837, 2017, doi: 10.1109/tac.2016.2620429.

[96] T. Ushio and S. Takai, "Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions," *IEEE Transactions on Automatic Control,* vol. 61, no. 3, pp. 799-804, 2016.

[97] M. Wakaiki, P. Tabuada, and J. P. Hespanha, "Supervisory control of discrete-event systems under attacks," *arXiv preprint arXiv:1701.00881,* 2017.

[98] F.-Y. Wang, Y. Gao, and M. Zhou, "A modified reachability tree approach to analysis of unbounded petri nets," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 34, no. 1, pp. 303-308, 2004.

[99] S. Wang, M. Gan, and M. Zhou, "Macro liveness graph and liveness of ω-independent unbounded nets," *Science China Information Sciences,* vol. 58, no. 3, pp. 1-10, 2015.

[100] S. Wang, C. Wang, and M. Zhou, "Design of optimal monitor-based supervisors for a class of petri nets with uncontrollable transitions," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 43, no. 5, pp. 1248-1255, 2013.

[101] S. Wang, D. You, and C. Seatzu, "A novel approach for constraint transformation in petri nets with uncontrollable transitions," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 48, no. 8, pp. 1403-1410, 2018, doi: 10.1109/tsmc.2017.2665479.

[102] S. Wang, D. You, and C. Wang, "Optimal supervisor synthesis for petri nets with uncontrollable transitions: A bottom-up algorithm," *Information Sciences,* vol. 363, pp. 261-273, 2016, doi: 10.1016/j.ins.2015.11.003.

[103] S. Wang, D. You, M. Zhou, and C. Seatzu, "Characterization of admissible marking sets in petri nets with uncontrollable transitions," *IEEE Transactions on Automatic Control,* vol. 61, no. 7, pp. 1953-1958, 2016, doi: 10.1109/tac.2015.2480233.

[104] S. Wang, M. Zhou, Z. Li, and C. Wang, "A new modified reachability tree approach and its applications to unbounded petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 43, no. 4, pp. 932-940, 2013.

[105] S. G. Wang, Y. Dan, and M. C. Zhou, "A necessary and sufficient condition for a resource subset to generate a strict minimal siphon in s ^{4} pr," *IEEE Transactions on Automatic Control,* vol. PP, no. 99, pp. 1-1, 2017.

[106] Y.-C. Wu and S. Lafortune, "Comparative analysis of related notions of opacity in centralized and coordinated architectures," *Discrete Event Dynamic Systems,* vol. 23, no. 3, pp. 307-339, 2013.

[107] Y.-C. Wu and S. Lafortune, "Synthesis of insertion functions for enforcement of opacity security properties," *Automatica,* vol. 50, no. 5, pp. 1336-1348, 2014.

[108] S. Xu and R. Kumar, "Discrete event control under nondeterministic partial observation," in *IEEE International Conference on Automation Science and Engineering*, 2009: IEEE, pp. 127-132.

[109] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of petri nets based on place invariants," *Automatica,* vol. 32, no. 1, pp. 15-28, 1996.

[110] R. Yang, Z. Ding, M. Pan, C. Jiang, and M. Zhou, "Liveness analysis of $\omega $-independent petri nets based on new modified reachability trees," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 47, no. 9, pp. 2601-2612, 2016.

[111] L. Ye, P. Dague, and F. Nouioua, "A predictability algorithm for distributed discrete event systems," in *International conference on formal engineering methods*, 2015: Springer, pp. 201-216.

[112] X. Yin, "Supervisor synthesis for mealy automata with output functions: A model transformation approach," *IEEE Transactions on Automatic Control,* vol. 62, no. 5, pp. 2576-2581, 2017.

[113] X. Yin, "Verification of prognosability for labeled petri nets," *IEEE Transactions on Automatic Control,* vol. 63, no. 6, pp. 1828-1834, 2017.

[114] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control,* vol. 61, no. 5, pp. 1239-1254, 2015.

[115] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control,* vol. 61, no. 8, pp. 2140-2154, 2015.

[116] X. Yin and S. Lafortune, "Synthesis of maximally-permissive supervisors for the range control problem," *IEEE Transactions on Automatic Control,* vol. 62, no. 8, pp. 3914-3929, 2016.

[117] X. Yin and S. Lafortune, "A new approach for the verification of infinite-step andk-step opacity using two-way observers," *Automatica,* vol. 80, pp. 162-171, 2017, doi: 10.1016/j.automatica.2017.02.037.

[118] X. Yin and S. Li, "Synthesis of dynamic masks for infinite-step opacity," *IEEE Transactions on Automatic Control,* pp. 1-1, 2019, doi: 10.1109/tac.2019.2916940.

[119] X. Yin and Z. Li, "Reliable decentralized fault prognosis of discrete-event systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 46, no. 11, pp. 1598-1603, 2015.

[120] X. Yin and Z. Li, "Decentralized fault prognosis of discrete event systems with guaranteed performance bound," *Automatica,* vol. 69, pp. 375-379, 2016.

[121] X. Yin, Z. Li, W. Wang, and S. Li, "Infinite-step opacity and k-step opacity of stochastic discrete-event systems," *Automatica,* vol. 99, pp. 266-274, 2019, doi: 10.1016/j.automatica.2018.10.049.

[122] T.-S. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *IEEE Transactions on automatic control,* vol. 47, no. 9, pp. 1491-1495, 2002.

[123] D. You, S. Wang, Z. Li, and C. Wang, "Computation of an optimal transformed linear constraint in a class of petri nets with uncontrollable transitions," *IEEE Access,* vol. 5, pp. 6780-6790, 2017, doi: 10.1109/access.2017.2696029.

[124] D. You, S. Wang, and C. Seatzu, "Supervisory control of a class of petri nets with unobservable and uncontrollable transitions," *Information Sciences,* vol. 501, pp. 635-654, 2019, doi: 10.1016/j.ins.2018.10.018.

[125] D. You, S. Wang, and C. Seatzu, "Verification of fault-predictability in labeled petri nets using predictor graphs," *IEEE Transactions on Automatic Control,* vol. 64, no. 10, pp. 4353-4360, 2019, doi: 10.1109/tac.2019.2897272.

[126] D. You, S. Wang, and M. Zhou, "Synthesis of monitor-based liveness-enforcing supervisors for $\rm {s}^{3} $ pr with ${\boldsymbol {\xi}} $-resources," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 45, no. 6, pp. 967-975, 2015.

[127] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for discrete event systems," *Annual Reviews in Control,* vol. 37, no. 2, pp. 308-320, 2013.

[128] B. Zhang, S. Shu, and F. Lin, "Maximum information release while ensuring opacity in discrete event systems," *IEEE Transactions on Automation Science and Engineering,* vol. 12, no. 3, pp. 1067-1079, 2015.

[129] H. Zhang, L. Feng, N. Wu, and Z. Li, "Integration of learning-based testing and supervisory control for requirements conformance of black-box reactive systems," *IEEE Transactions on Automation Science and Engineering,* vol. 15, no. 1, pp. 2-15, 2017.

[130] P. Zhang, S. Shu, and M. Zhou, "An online fault detection model and strategies based on svm-grid in clouds," *IEEE/CAA Journal of Automatica Sinica,* vol. 5, no. 2, pp. 445-456, 2018.

[131] Q. Zhang, Z. Li, C. Seatzu, and A. Giua, "Stealthy attacks for partially-observed discrete event systems," in *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, vol. 1: IEEE, pp. 1161-1164.

[132] G. Zhu, Z. Li, N. Wu, and A. Al-Ahmari, "Fault identification of discrete event systems modeled by petri nets with unobservable transitions," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 49, no. 2, pp. 333-345, 2017.