From Security Enforcement to Supervisory Control in Discrete Event Systems: Qualitative and Quantitative Analyses

by

Yiding Ji

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2019

Doctoral Committee:

Professor Stéphane Lafortune, Chair
Assistant Professor Jean-Baptiste Jeannin
Assistant Professor Necmiye Ozay
Professor Demosthenis Teneketzis

Yiding Ji
jiyiding@umich.edu
ORCID iD: 0000-0003-2678-7051

DEDICATION

To all the people who have helped me through the journey of my PhD study.

ACKNOWLEDGEMENTS

I would like to convey, through this acknowledgment, my sincere gratitude to all the people who have supported me, inspired me, instructed me, encouraged me, and accompanied me during the past years of my PhD study.

First, I would like to thank my advisor Professor Stéphane Lafortune, who opened the gate of Discrete Evnet Systems for me and guided me in exploring this interesting field. I am also very grateful for the freedom he gave me to conduct my research during the past years and I have learned a lot from him.

Next, I would like to say thanks to Professor Ozay and Professor Teneketzis at this moment. I benefited significantly from taking their courses and they gave me insightful instructions for completing this dissertation. I am so lucky to be Professor Teneketzis's last officially instructed doctoral student before his retirement. Here I would also like to thank Professor Jeannin for being a member of my dissertation committee and helping me to finish this dissertation.

I want to express my thanks to Professor Xiang Yin at Shanghai Jiao Tong University, who used to be my colleague and labmate at the Univesity of Michigan. Xiang contributed to many of my publications and is always a model for me towards being a mature researcher. I will never forget his helpful and insightful instructions.

During my graduate study, I have acquired vast knowledge from the excellent courses offered by the University of Michigan. This should be largely attributed to the instructors, and I want to thank you all.

While at the University of Michigan, my colleagues in UMDES group shared many great ideas with me about the research in Discrete Event Systems. I had a deeper understanding of my research from the fruitful discussions with you guys: Dr. Yichin Wu, Dr. Xiang Yin, Romulo Meira Góes, Dr. Lilian Carvalho, Dr. Richard Hill, Dr. Eunsuk Kang, Dr. Christoforos Keroglou, Dr. Balke C. Rawlings and Dr. Sahar Mohajerani. By the way, I want to thank all the undergraduate students who worked for our lab during the past years.

Then I would thank my faculty instructor Professor Brent Gillespie and my students at the course *Linear System Theory* during the 2018 fall semester. You gave me a wonderful experience of being a teaching assistant.

It is never enough to express my thanks to all of my dear friends at each period of my life. You are always there to help me out of the difficulties and selflessly back me no matter when or where. A true friendship never perishes, and I will cherish it forever. I am so sorry for not being able to mention all the names here as that is a long list. Thank you all, my friends!

I would like to acknowledge the financial aid from the National Science Foundation and the University of Michigan that allowed me to finish my study.

I am always grateful for the Depertment of Electrical Engineering and Computer Science, the College of Engineering and the University of Michigan for providing me a wonderful environment of study and research, which values equality, diversity, inclusiveness, valiance, social responsibility, and eagerness for truth. I feel respected and have a strong sense of belonging. Wherever I go, I will go blue!

Recalling my life, I hope to express my warmest gratitude to all my family members, especially my mother Haiyan Guo and my father Weiguang Ji. My parents devoted so much to me and they are the first teachers in my life. Their everlasting love and support lead me to where I am now. No word may eulogize your love and I may only wish to accompany you more in the future.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Cyber-physical systems are technological systems that involve physical components that are monitored and controlled by multiple computational units that exchange information through a communication network. Examples of cyber-physical systems arise in transportation, power, smart manufacturing, and other classes of systems that have a large degree of automation. Analysis and control of cyber-physical systems is an active area of research. The increasing demands for safety, security and performance improvement of cyber-physical systems put stringent constraints on their design and necessitate the use of formal model-based methods to synthesize control strategies that provably enforce required properties. This dissertation focuses on the higher level control logic in cyber-physical systems using the framework of discrete event systems. It tackles two classes of problems for discrete event systems. The first class of problems is related to system security. This problem is formulated in terms of the information flow property of opacity. In this part of the dissertation, an interface-based approach called insertion/edit function is developed to enforce opacity under the potential inference of malicious intruders that may or may not know the implementation of the insertion/edit function. The focus is the synthesis of insertion/edit functions that solve the opacity enforcement problem in the framework of qualitative and quantitative games on finite graphs. The second problem treated in the dissertation is that of performance optimization in the context of supervisory control under partial observation. This problem is transformed to a two-player quantitative game and an information structure where the game is played is constructed. A novel approach to synthesize supervisors by solving the game is developed.

The main contributions of this dissertation are grouped into the following five categories. (i) The transformation of the formulated opacity enforcement and supervisory control problems to games on finite graphs provides a systematic way of performing worst case analysis in design

of discrete event systems. (ii) These games have state spaces that are as compact as possible using the notion of information states in each corresponding problem. (iii) A formal model-based approach is employed in the entire dissertation, which results in provably correct solutions. (iv) The approaches developed in this dissertation reveal the interconnection between control theory and formal methods. (v) The results in this dissertation are applicable to many types of cyber-physical systems with security-critical and performance-aware requirements.

CHAPTER I

Introduction

I.1 Background and Motivation

Security and performance optimization are two important research topics in Discrete Event Systems. In modern large-scale cyber-physical systems, many components of the system are potentially vulnerable to attackers with malicious purposes to infer some confidential information about the system and inflict damage. Therefore, it is important to develop formal tools to preserve the security of the system. Meanwhile, it is also necessary to evaluate the performance of the system quantitatively and optimize relevant performance measures.

In the context of discrete event systems, opacity is an information-flow based security property that characterizes whether or not secrets of a given dynamic system can be inferred by an outside observer termed *intruder* with potentially malicious intentions. Due to its general formulation that is applicable to many security and privacy issues arising in networked systems, opacity has received significant attention in the literature on security and privacy since it was first introduced in [75]. In the setting of opacity, the external intruder is often modeled as an observer that knows the structure of the system and attempts to infer secrets of the system by passively observing the system's outputs. The system is called *opaque* if the intruder fails to determine system's secrets unambiguously from its observations. Opacity has been thoroughly discussed in discrete event systems, which provide a convenient and systematic way for problem modeling and analysis. Several notions of opacity have been proposed in discrete event systems and studied ever since [19, 20].

In practice, opacity may not always hold so that the problem of opacity enforcement naturally arises. In this dissertation, we mainly focus on the problem of enforcing opacity by *insertion functions* and *edit functions*, which serve as an interface between the output of the system and the intruder. The edit function may insert some strings into the output of the system or erase some events, so what the intruder observes is different from the actual output. In that sense, the intruder may be obfuscated and fails to infer critical information from its observations. Based on the intruder's *knowledge* about the implementation of the obfuscation methods, we consider both strong and weak attack scenarios in this dissertation, where the intruder may or may not know the implementation of insertion/edit functions. For both scenarios, we characterize the properties of insertion/edit functions and propose methods to synthesize them for opacity enforcement. We show the mechanism of insertion functions in Figure I.1, while the mechanism of edit functions is similar, which also includes event erasure.

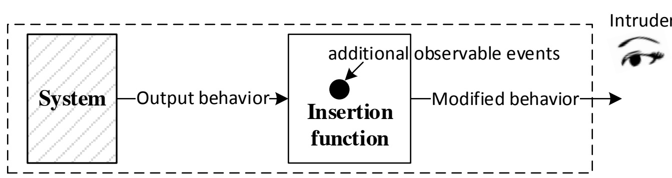

Figure I.1: The insertion mechanism

Along with qualitative analysis of opacity enforcement, we also extend our obfuscation methods to consider opacity enforcement under quantitative constraints. We assume that the system has several types of resources whose amounts are all fixed. The system's resource levels may change due to event occurrences and defense of secrets. Under this framework, our objective is to guarantee that secrets are not disclosed to the intruder while each type of resource is never depleted in the process of enforcing opacity.

Therefore, we consider opacity enforcement by leveraging the technique of insertion functions and further investigate it under a quantitative setting. This problem is inspired by the rapidly growing application of *location-based services (LBS)*. Suppose there is a device providing LBS, which sends personalized information to the user by exploiting the user's real time location. There may be a malicious eavesdropper which intends to infer some critical information of the user from the queries sent by the device, through the open communication network. To prevent the disclosure of secrets, some fictitious queries may be inserted to the ongoing queries if they are going to reveal the user's critical information. Then the resulting query sequences must be made consistent with some existing queries not revealing any secret information. This mechanism is shown in Figure I.2. Since inserting queries may cost certain resources like electricity, bandwidth and memory, the insertion functions may not insert arbitrary long or arbitrary many queries for obfuscation in practice. They should be properly designed so that the resource budget requirements are always satisfied and the resources are not consumed too sharply, i.e., the insertion functions work economically.

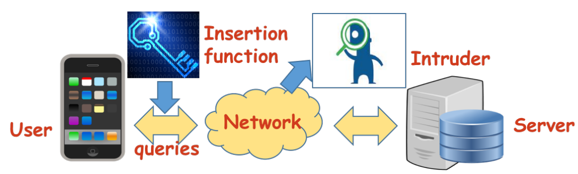

Figure I.2: Location-based service and insertion mechanism

Together with security obfuscation, this dissertation also studies another important research topic in discrete event systems, i.e., performance optimization by some quantitative measures. In many practical situations, the system may generate or consume some resources, e.g., energy, during the operation and over an arbitrarily long time horizon. In this circumstance, two requirements arise naturally. One is to ensure that the resource is never depleted as long as the system is operating, given a fixed amount of initial resource. The other requirement is to guarantee that the long run average rate of resource generation (consumption) is above (below) a given threshold. Furthermore, if the system does not terminate, it is preferable to optimize the above mentioned long run average rate so that the system works in an economical way. Those requirements motivate the problems discussed in this dissertation.

To achieve such objectives, proper supervisors are designed to restrict the behaviors of the system. The classic supervisory control theory in discrete event systems was initiated in [93] where the supervisor dynamically enables/disables events to ensure that the plant achieves certain specification. As it is not always feasible to sense every step of the operation of the plant, the supervisor may only have partial observation of the system. Given these considerations, we investigate the so called *energy-aware supervisory control problem* whose general mechanism is shown in Figure I.3. In the figure, IA stands for *information acquisition* which determines the supervisor's observation for the system. As is seen, the supervisor's commands are subject to quantitative energy/resource constraints. Specifically, we investigate optimal mean payoff supervisory control under partial observation in this dissertation, where our principal objective is optimize the long-run average resource payoff by supervisory control. To be more specific, we will transform the supervisory control problem to a two-player game and propose a novel information structure to encode the strategies for both players. Then we leverage results from quantitative graph game theory to further analyze the game. Finally we develop a systematic approach to synthesize the optimal supervisor by solving the game.

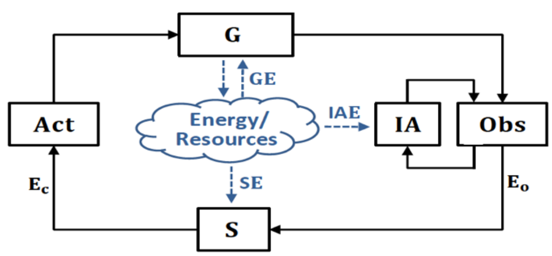

Figure I.3: The general mechanism of energy-aware supervisory control

I.2 Literature Review

In the context of discrete event systems, many problems related with opacity have been studied after it was first discussed in the computer security literature [19, 20]. Those problems may be categorized into two classes: proposing new notions of opacity and enforcing opacity. We will briefly review some representative works on both topics. Since two chapters in this dissertation are also inspired by quantitative graph game theory in theoretical computer science, which considers reactive synthesis under the game framework, we also do a brief literature review here on graph games with quantitative objectives. Finally, we list some relevant works on supervisory control theory, which is closely related to the last technical chapter of this dissertation.

I.2.1 Opacity Notions and Enforcement Methods

Since initiated by [20], opacity has received significant attention in the context of discrete event systems. Various representations of the system secret have been considered in the study of opacity. These representations have led to the formalization of several notions of opacity for event-driven models of dynamic systems. In the context of automata models, the notions of initial-state opacity, current-state opacity, language-based opacity, K-step opacity and infinite step opacity, have been proposed; see, e.g., [25,70,99,102,127]. Opacity has also been generalized to the settings of infinite state systems, see.,e.g., [35], modular systems, see.,e.g., [74] and timed systems, see.,e.g., [24, 117], while opacity under so-called Orwellian observers is investigated in [79]. Another important model in discrete event systems is Petri nets where opacity is discussed in many works such as [20, 112, 113]. In addition, system secrecy and opacity has been extend to quantitative settings [8, 36], while specifically, several stochastic notions of opacity have been defined and investigated; see, e.g., [6, 7, 61, 100]. In [132], an algorithm was proposed for verification of infinite-step opacity in stochastic discrete event system. A different framework was proposed in [131] to study opacity in networked control systems with insecure control channels. Some recent survey papers such as [52, 67] may be consulted for a detailed review of the literature on opacity in discrete event systems.

To alleviate the issue of heavy computation for opacity verification, some formal methods may be applied, like abstraction and composition. For abstraction, simulation and observation equivalence [76] are well-known methods to abstract the state space of an automaton. In general, bisimulation and observation equivalence do not preserve opacity properties. A variant called opacity-preserving bisimulation was discussed in [134] to reduce the state space of the system when verifying infinite-step opacity. A unified abstraction method called visible bisimmulation equivalence was proposed in [68] and then extended in [81] for abstraction-based opacity verification. Furthermore, the authors of [82] constructed observer of modular systems incrementally for verification and enforcement of current state opacity, which avoids the explosion of state space.

When a given notion of opacity is violated, researchers have proposed various methods for its enforcement. One popular approach is to design a minimally restrictive supervisor, which disables certain behaviors that violate opacity [38, 41, 101, 110]. A uniform approach was proposed in [126] to embed in a finite structure all feasible supervisors that enforce opacity and this structure is applied to synthesize supervisors with desired properties. The work in [133] also lies in this category but discusses the problem from the perspective of maximum information release. While [114] also adopts supervisory control for opacity enforcement, however it assumes that the intruder and the supervisor has incomparable observation. On the other hand, several works, such as [25, 124, 129, 130], apply another sensor activation framework to enforce opacity by building dynamic observers or most-permissive observers. Along with the above mentioned two popular techniques, a run-time method was discussed in [45] for enforcement of several notions of opacity.

In contrast to the above approaches, [119] introduces insertion functions as a new method, which insert fictitious events into the system's output to obfuscate the intruder. The insertion functions serve as an interface between the system's output and the intruder's observation. After that, [120] investigates opacity enforcement under the assumption that the intruder may or may not know the implementation of the insertion functions. To capture this situation, two concepts of private safety and public safety are defined and studied for evaluating the performance of insertion functions. As a following work, [121] discussed optimal insertion function in terms of the average insertion cost. Furthermore, the authors of [122] proceed to extend insertion functions to edit functions, which modify the system's output by inserting, erasing or replacing events. All these works enforce opacity in a deterministic setting, i.e., any string is mapped to a unique string.

I.2.2 Graph Games with Quantitative Objectives

In theoretical computer science, games on graphs with a quantitative objective [4] is a thoroughly investigated topic. Games provide a theoretical method to deal with logical requirements in reactive synthesis while games with quantitative objectives are natural models for design in resource-constrained environments. The specifications for such reactive systems usually have both a quantitative component specifying the resource constraints and a qualitative component specifying the logical goal. And some of the most intensively studied games include reachability games [2, 16, 27, 39], mean payoff games [17, 43, 135], energy games [12, 26], mean payoff and energy parity games [29, 32], etc.

Among all the above mentioned classes of games, we are especially interested in energy games and mean payoff games as they inspired some of our works in this dissertation. The energy game is a two-player quantitative game on weighted graphs, where the weights represent energy gain or consumption. The objective of the first player is to keep the energy level not below 0 while the other player intends to do the opposite. Depending on whether the initial-credit energy is fixed or not, the fixed initial energy problem studies whether the objective could be achieved given a certain amount of energy while the unknown initial energy problem asks whether there exists certain amount of initial energy to achieve the objective. The other way of classifying energy games is by the information available to the players. In the full observation case, both players have complete knowledge about the strategies and positions of each other [11, 30]. And partial information is reflected in one or both players being unable to determine the precise location of the other player [10, 31, 40, 51]. Considering partial observation in energy games results in enormous increase in the complexity of the problem, in terms of strategy synthesis [87, 95]. Some types of imperfect information energy games may be reduced to and solved as a reachability game with perfect information. Energy games with fixed initial energy is decidable with incomplete information, while they become undecidable when the initial energy is not fixed [51]. In general, mean payoff games with incomplete information are not decidable while some special decidable classes of games are presented in [51].

Energy games and mean payoff games have also been extended from one dimension to multiple dimensions to characterize different resource constraints [44, 60, 116], which are generally more complex than their single dimension counterpart. Recently, stochastic games have also been investigated [18, 28, 46], where each player's decisions are made with certain probability and their objective is evaluated with probability. Some researchers in DES also studied supervisory control by energy game with partial observation [90].

I.3 Qualitative and Quantitative Supervisory Control

Supervisory control under the framework of discrete event systems has been widely studied for qualitative specifications, such as safety and liveness, since it was initiated in [93]. The DES under control is modeled by an automaton with event set partitioned as controllable and uncontrollable event sets. The supervisor restricts the original behavior of the system so that a given specification is satisfied. Since then, supervisory control theory has been discussed under various settings in DES [23, 105, 118], such as Petri nets, see, e.g., [48], timed systems, see, e.g., [14], networked systems, see, e.g., [107], distributed systems, see, e.g., [63], decentralized systems, see, e.g., [71, 97], stochastic systems, see, e.g., [47, 64], and so on.

In the context of discrete event systems, due to the limited sensing capabilities, the plant is usually partially observed, which gives rise to supervisory control under partial observation [72]. Many works fall into this category, see, e.g., [1, 21, 22, 37, 49, 62, 96, 108, 111, 115, 128, 129], which discuss the problem from different perspectives. Recently, a novel approach was developed in [125] and extended in [126] to synthesize maximally permissive partial-observation supervisors for enforcement of a series of qualitative properties in discrete event systems without assumptions on the relation between controllable events and observable events. The following work [123] adopted this approach to investigate supervisory control for mealy automata with output functions.

Besides logical properties, supervisory control has also been investigated by introducing some quantitative performance measures. Optimal supervisory control is one problem of particular interest, starting with [86]. Since then, different frameworks of optimal supervisory control have been developed. For example, [106] defined both event enablement and disablement costs, then found the controller with minimum total costs by dynamic programming. This framework was extended in [73, 89] to consider partial observation of the system. Furthermore, [84] studied optimal supervisory control in probabilistic discrete event systems and [109] proposed a timed optimal supervisor. In [65], the authors viewed the weighted automaton as a flow network and solved the optimal supervisory control problem by leveraging the max-flow min-cut theorem. Besides, [94] defined a quantitative language measure and discussed the corresponding optimal supervisory control problem based on it. As a variant, the optimal stabilization problem under disturbances was investigated in [88]. All the above works evaluated the performance of the supervisor by considering finite behaviors. In contrast, [91] optimized the worst case limit average weight of the infinite sequences generated by the controlled system. The problem was formulated and solved as a mean payoff game between the supervisor and the environment, under full observation. In practice, optimal supervisory control has been applied to some engineering fields, see, e.g., [15, 85, 104].

I.4 Organization and Contributions of the Dissertation

I.4.1 Organization

The remaining chapters of the dissertation are organized as follows. Chapter II presents the work on opacity enforcement by insertion functions [54]. Chapter III presents the work on opacity enforcement by (nondeterministic) edit functions [53, 58]. Chapter IV presents the work on opacity enforcement by insertion functions under (multiple) energy constraints [56, 57]. Chapter V presents the work on optimal supervisory control with quantitative objectives and under partial observation [55, 59]. Finally, Chapter VI concludes the dissertation and presents some potential future research directions.

I.4.2 Main Contributions

This dissertation mainly concentrates on two problems: opacity enforcement by insertion/edit functions and optimal mean payoff supervisory control under partial observation. In terms of the methodologies, we transform both problems to the settings of qualitative or quantitative games and solve them on the games. In this manner, we find a proper way to deal with *worst-case analysis* in both problems, as we need to ensure that the synthesized insertion/edit functions and supervisors are reactive to all potentially possible circumstances imposed by the environment.

More specifically, for the opacity enforcement problem, this dissertation has the following major technical contributions: (i) it shows that publicly and privately safe insertion functions always exist when privately safe insertion functions exist; (ii) it provides a way of synthesizing publicly and privately safe insertion functions based on a two-player game structure called All Insertion Structure; (iii) it characterizes public safety for edit functions and proposes a novel three-player game structure called All Edit Structure to embed edit functions; (iv) it introduces nondeterministic edit functions and develops an approach to synthesize them; (v) it discusses insertion functions under multiple energy constrains and presents a way of synthesizing insertion functions for opacity enforcement without making the system's energy levels below 0; (vi) it proposes and solves the bounded cost rate insertion problem where the rate of insertion cost associated with each type or resource is bounded by certain threshold.

For the optimal supervisory control problem, the contributions are three-fold: (i) it discusses mean payoff supervisory control under partial observation for the first time in discrete event systems; (ii) a systematic approach is developed to transform the supervisory control problem to a two-player game by leveraging results from energy games and mean payoff games with incomplete information; (iii) an algorithm is given to synthesize the optimal supervisor on the game in a dynamic programming manner.

CHAPTER II

Enforcement of Opacity by Public and Private Insertion Functions

II.1 Introduction

In [119], it is assumed that the insertion function used by the system is always kept private from the intruder. Under this assumption, a method is presented on how to synthesize insertion functions that only output strings consistent with the non-secret behavior of the system and thus prevent the intruder from being certain that a secret behavior has occurred. In this chapter, we relax that assumption. While the implementation of the insertion function may be kept private at first, a sophisticated intruder may learn the full set of modified behaviors output by the insertion function, compare it with the system model, and potentially reverse engineer the insertion function. Also, if the intruder knows the system's optimality criteria, it may follow the optimal synthesis algorithm in [121] and discover the correct insertion function. It may also be the case that the system designers decide to make the insertion function public, as is done in public-key cryptography, for example. *Hence, there is a need to design insertion functions that enforce opacity even when their implementation becomes known to the intruder.* Under the same insertion mechanism as in Figure I.1, to enforce opacity regardless whether or not the intruder knows the implementation of the insertion function, we formally characterize a property called *public-and-private enforceability*, or *PP-enforceability* for short. A PP-enforcing insertion function is guaranteed to enforce opacity when the insertion function is kept private *and* when it becomes known to the intruder. In the former case, the insertion function outputs only behaviors consistent with non-secret behaviors of the system. In the latter case, the insertion function is designed such that for every secret behavior of the system, there is a non-secret behavior of the system that has the same modified output from the insertion function.

The main contributions of this chapter are as follows. First, we formally characterize the properties of public enforceability and of public-private (PP) enforceability, in the context of opacity enforcement by insertion functions. We present conditions for PP-enforceability and use them to derive an effective test under which opacity is public-private enforceable. It turns out that if there exists an insertion function that is privately enforcing, then there also exists a (potentially different) insertion function that is PP-enforcing. This result is established by defining a so-called greedy criterion for selecting insertion functions in the All Insertion Structure (AIS) introduced in [119]. These new results lead to an algorithmic procedure, called Algorithm INPRIVALIC-G, that is guaranteed to synthesize a PP-enforcing insertion function if one exists.

The remaining sections of this chapter are organized as follows. Section II.2 introduces the system model and the notion of opacity. Section II.3 formally introduces insertion functions and the notion of *public-and-private enforceability*, along with conditions under which private enforceability and public-private enforceability hold for a given insertion function. Section II.4 starts by reviewing the construction procedure of the All Insertion Structure (AIS) from [121] and then identifies relevant concepts and properties. In Section II.5, we first present a sufficient condition for insertion functions to be PP-enforcing, then define the greedy criterion and show that a greedy insertion function is PP-enforcing. Then, in Section II.6, the INPRIVALIC-G Algorithm is presented, which synthesizes PP-enforcing insertion functions by using a greedy-maximal insertion criterion within the AIS. Finally, Section II.7 concludes the chapter.

II.2 System Model

We consider opacity in the framework of discrete event systems modeled as finite-state automata [23]:

$$G = (X, E, f, X_0)$$

where X is the finite set of states, E is the finite set of events, $f : X \times E \to X$ is the partial state transition function and $X_0 \subseteq X$ is the set of initial states. Specifically, we denote $X_S \subset X$ as the set of *secret states*. The transition function is extended to domain $X \times E^*$ in the standard manner and we still denote the extended function by f. We denote by $s \preceq u$ if s is a prefix u, and $s \prec u$ if $s \preceq u, s \neq u$. Also, we denote by $t \in s$ if string t is a substring of s. In opacity problems, the initial state may not be known *a priori* by the intruder and thus we include a set of initial states X_0 in the definition of G. The language generated by G is defined as $\mathcal{L}(G) = \{s \in E^* : \exists x_0 \in X_0, \text{ s.t. } f(x_0, s)!\}$ where ! means "is defined".

In system G, given a string $s = e_1 e_2 \cdots e_{k-1}$, its corresponding *execution* is a sequence of the form $\langle x_1, e_1, \ldots, e_{k-1}, x_k \rangle$, where $x_i \in X$, $e_i \in E$ and $x_{i+1} = f(x_i, e_i)$, $\forall i \in \{1, 2, \ldots, k-1\}$. An execution forms a *cycle* if $x_1 = x_k$ and a cycle is an *elementary cycle* if $\forall i, j \in \{1, 2, \ldots, k-1\} : i \neq j \Rightarrow x_i \neq x_j$. Besides, string s contains a *cycle* if $\exists t \in s$, $t \neq \epsilon$, $\exists x \in X$, s.t. $f(x, t) = x$. Otherwise, we call s a *cycle-free string*.

We assume that the system G is partially observable and the event set E is partitioned as $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events. Given a string $t \in E^*$, its natural projection $P : E^* \to E_o^*$ is recursively defined as $P(t) = P(t'e) = P(t')P(e)$ where $t' \in E^*$ and $e \in E$. The projection of an event is $P(e) = e$ if $e \in E_o$ and $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string.

Given a set of states $q \subseteq X$ and an observable event $e_o \in E_o$, the *unobservable reach*, denoted as $UR(q)$, is defined as: $UR(q) = \{x \in X : \exists x' \in q, \exists s \in E_{uo}^*, \text{ s.t. } f(x', s) = x\}$. Besides, the *observable reach*, denoted by $Next(q, e_o)$, is defined as: $Next(q, e_o) = \{x \in X : \exists x' \in q, \text{ s.t. } f(x', e_o) = x\}$. Then, the *observer* of G is defined as: $Obs(G) = (X_{obs}, E_o, \delta, x_{obs,0})$ where $X_{obs} \subseteq 2^X$, $x_{obs,0} = UR(X_0)$

and for any $x_{obs} \in X_{obs}$, $e_o \in E_o$, $\delta(x_{obs}, e_o) = UR(Next(x_{obs}, e_o))$. We denote the state reached by $\delta(x_{obs,0}, s)$, $s \in P[\mathcal{L}(G)]$ as the *current state estimate* associated with s.

II.3 Insertion Mechanism and Opacity Notions

We first review the concept of current-state opacity.

Definition II.3.1 (Current-State Opacity (CSO))**.** *Given system $G = (X, E, f, X_0)$, projection P, and the set of secret states X_S, G is CSO if $\forall t \in L_S := \{t \in \mathcal{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap X_S \neq \emptyset\}$, $\exists t' \in L_{NS} := \{t \in \mathcal{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap (X \setminus X_S) \neq \emptyset\}$ such that $P(t) = P(t')$.*

In words, whenever the system generates a string t that ends at a secret state in X_S, there must exist a string t' such that t' ends at a state in $X \setminus X_S$ and $P(t) = P(t')$. Hence, the intruder cannot ascertain for sure that the current system state is in X_S.

An insertion function is defined as a (potentially partial) function $f_I : E_o^* \times E_o \to E_o^*$ that outputs a string with inserted events based on the past observed behavior and the current observed event. Given observable string $se_o \in P[\mathcal{L}(G)]$, $f_I(s, e_o) = s_I e_o$ when string $s_I \in E_o^*$ is inserted before e_o. We also define the *string-based* version of f_I, denoted by f_I^{str}, recursively from f_I: $f_I^{str}(\epsilon) = \epsilon$ and $f_I^{str}(se_o) = f_I^{str}(s) f_I(s, e_o)$. Given G, the modified language output by insertion function f_I is denoted by $f_I^{str}(P[\mathcal{L}(G)]) = \overline{\{\tilde{s} \in E_o^* : \exists s \in P[\mathcal{L}(G)], f_I^{str}(s) = \tilde{s}\}}$. When multiple events are inserted, we assume that they are inserted, hence observed, one by one. Notice that the insertion functions f_I (and corresponding f_I^{str}) considered in this chapter are deterministic.

We encode a given insertion function as an input/output (I/O) automaton

$$IA = (X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$$

and call it an *insertion automaton*. The state set X_{ia} of IA could potentially be infinite. The input set is E_o; the output set is a set of *strings* in $E_o^+ = E_o^* E_o$; the transition function f_{ia} defines the dynamics of IA; the output function q_{ia} is defined such that $q_{ia}(x, e_o) = s_I e_o$ where $f_{ia}(x_{ia,0}, s) = x$, if $f_I(s, e_o) = s_I e_o$; and finally $x_{ia,0}$ is the initial state. More details on I/O automata can be found in [23].

II.3.1 Private Enforceability

Admissibility is an input property for insertion functions; it requires insertion functions to be defined for all $P[\mathcal{L}(G)]$.

Definition II.3.2 (Admissibility)**.** *Consider G, P, L_S and L_{NS}. An insertion function f_I is admissible if: $\forall se_o \in P[\mathcal{L}(G)]$, where $s \in E_o^*, e_o \in E_o$, $\exists s_I \in E_o^*$ s.t. $f_I(s, e_o) = s_I e_o$.*

Private safety is an output property of insertion functions. We term this property "private" safety because it is under the assumption that the intruder has no knowledge of the insertion function at the outset. Consequently, the intruder is expecting to observe behaviors that are consistent with the system's transition structure. Notice that we consider insertion functions that are used to enforce opacity *online*. Hence, every modified output behavior from the insertion function should *always* be consistent with an original non-secret behavior from the system. Because of this "always" requirement, every modified output behavior should be observationally equivalent to a string in the safe language L_{safe}, which is the supremal prefix-closed sublanguage of $P(L_{NS})$ and is calculated by the equation:

$$L_{safe} = P[\mathcal{L}(G)] \setminus \{P[\mathcal{L}(G)] \setminus P(L_{NS})\}E_o^*$$

This equation is an application of a result in [66] and a similar expression was also proposed in [41]. Hereafter, we call a string $s \in P[\mathcal{L}(G)]$ *safe* if it is in L_{safe} and *unsafe* otherwise, so $L_{unsafe} = P[\mathcal{L}(G)] \setminus L_{safe}$. From the definition of safe language, if a string is unsafe, then all its continuations are unsafe.

Definition II.3.3 (Private Safety)**.** *Consider G with P, L_S and L_{NS}. An insertion function f_I is privately safe if $\forall s \in P[\mathcal{L}(G)]$, $f_I^{str}(s) \in L_{safe}$; equivalently, $f_I^{str}(P[\mathcal{L}(G)]) \subseteq L_{safe}$.*

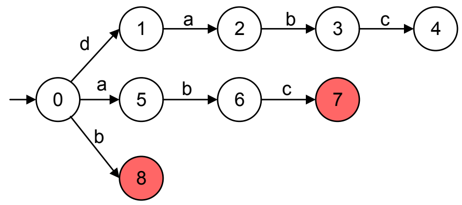

Figure II.1: Current-state estimator $\mathscr{E}$; states 7 and 8 contain only secret states

If we delete all states violating CSO from $Obs(G)$ and take the accessible part, the resulting automaton just generates L_{safe}. We define it as *desired observer* $Obs_d(G) = (X_{obsd}, E_o, \delta_d, x_{obsd,0})$.

II.3.2 Private and Public Enforceability

Privately enforcing insertion functions enforce opacity by insuring that the intruder never observes an unsafe string. A naive intruder, with no knowledge of the insertion function at the outset, would therefore never be certain about the secret being revealed; in fact, the intruder would have no reason to suspect the existence of an insertion function. However, a privately enforcing insertion function may fail if it becomes known to the intruder, as illustrated by the following example.

Example II.3.1. *Consider the current-state estimator in Figure II.1. These estimator states represent sets of system states; they are numbered from* 0 *to* 8 *for simplicity. Assume that states* 7 *and* 8 *contain only secret states; i.e., these estimator states reveal the secret. Suppose that opacity is enforced by the privately enforcing insertion function where* $f_I^{str}(b) = ab, f_I^{str}(a) = da$ *and no other insertions are made. If the intruder has no knowledge of* f_I*, then it would never conclude that the secret is revealed, as the output from* f_I *is always safe; here,* $L_{safe} = \overline{\{dabc, ab\}}$*. However, if the intruder knows the implementation of* f_I*, then it would be able to conclude that the state estimate is state* 8 *when it observes ab. This is because if ab were the genuine output behavior from the system, then it would have been modified to dab; and the intruder knows that. Hence, the only system output that would produce ab is string b.*

Example II.3.1 shows how an intruder can infer the secret if it knows the implementation of the insertion function. Indeed, there are ways for intruders to learn the implementation of the insertion function. For example, the intruder could use learning algorithms, such as in [3], to learn the modified system $\tilde{G}$, which is the parallel composition of G with insertion automaton IA, and then use $\tilde{G}$ and G to reverse engineer IA. This type of parallel composition of a regular automaton with an I/O one is sometimes called "input parallel composition"; we refer the reader to [119] for its formal definition. Alternatively, if the intruder knows the optimality criteria used by the system's designer, it could follow certain synthesis algorithm and construct the correct insertion function. In either case, we wish to use an insertion function that still enforces opacity when its implementation becomes known. In this manner, the system designers may be able to eventually reveal the structure of f_I, if so desired.

PP-enforceability is a specification that we characterize under the assumptions that: (i) the intruder does not know about the implementation of the insertion function at the outset; but (ii) the intruder can possibly learn or be told the correct implementation. Consequently, to enforce opacity under assumption (i), insertion functions should be privately safe. Also, under assumption (ii), insertion functions should be defined so that the intruder is still not able to determine the occurrence of the secret even if it knows about the insertion function's implementation. The second requirement is formally characterized as a property called *public safety*, defined as follows.

Definition II.3.4 (Public Safety)**.** *Consider G with P, L_S and L_{NS}. An insertion function f_I is publicly safe if* $\forall \tilde{s} \in f_I^{str}(P[\mathcal{L}(G)]), \exists t \in L_{safe}$ *s.t.* $f_I^{str}(t) = \tilde{s}$*; equivalently,* $f_I^{str}(P[\mathcal{L}(G)]) \subseteq f_I^{str}(L_{safe})$.

In contrast to Definition 4 in [120], we use L_{safe} instead of $P(L_{NS})$ in the above definition to better capture the on-line operation of the system, where public safety must be preserved for every prefix of a safe string. The idea behind public safety is that no matter what the insertion function outputs, this output could have been obtained from a safe string; hence opacity holds.

When an insertion function is admissible and publicly safe, we say that it is *publicly enforcing*. Moreover, we say that an insertion function satisfies the property of *private-and-public enforceability*, or *PP-enforceability*, if it is admissible, privately safe, and publicly safe.

Definition II.3.5 (PP-Enforceability)**.** *Insertion function f_I is PP-enforcing if it is admissible, privately safe, and publicly safe.*

Example II.3.2. *In Example II.3.1, insertion function f_I is privately enforcing but not PP-enforcing. Specifically, for $\tilde{s} = ab$, there is no $t \in L_{safe}$ for which $f_I^{str}[P(t)] = ab$. Let us define another insertion function: $f_I'(\epsilon, a) = da, f_I'(\epsilon, b) = dab$, and $f_I'(s, e_o) = e_o, \forall se_o \in P[\mathcal{L}(G)] \setminus \{a, b\}$. One can verify that f_I' is PP-enforcing. Specifically, f_I' is admissible because it is defined for every $P[\mathcal{L}(G)]$; it is privately safe as $f_I'(P[\mathcal{L}(G)]) = \overline{\{dabc\}} \subseteq L_{safe}$; also, f_I' is publicly safe since for every $\tilde{s} \in \overline{\{dabc\}}$, there exists $t \in L_{safe}$ that is observationally equivalent and is unmodified by f_I', which is sufficient to ensure the condition in Definition II.3.4.*

It may be tempting to think that a publicly enforcing insertion function should also be privately enforcing, as if we deprive the intruder from the knowledge of the insertion function, it should make its inference task harder. However, this is not true in general, as shown in the following example.

Example II.3.3. *Consider the current-state estimator with strings $\overline{\{ab, b\}}$, where string ab is secret. Consider the insertion function f_I: $f_I(\epsilon, b) = ab$ and $f_I(s, e_o) = e_o, \forall se_o \in \overline{\{ab\}}$. This insertion function is publicly enforcing since it is admissible and the only unsafe behavior ab is now observationally equivalent to safe behavior b. However, if the intruder does not know the implementation of f_I, it would always believe that the secret has occurred. Hence, the secret will be revealed when the system indeed outputs ab.*

The issue in the preceding example is that a publicly safe insertion function is free to map strings to anything, as long as the condition in Definition II.3.4 holds. It is not required that the output string be safe. This explains our choice of using PP-enforceability as our specification for insertion functions. We do not wish to make any assumptions about the intruder's knowledge, either at the outset or as it keeps observing the system. Thus, insertion functions should enforce opacity regardless what the intruder knows about the implementation of insertion function, including nothing. Hence, by also requiring private safety, PP-enforceability ensures that only safe strings will be output.

Our goal is to develop a synthesis algorithm for PP-enforcing insertion functions. For this purpose, we use the discrete structure called "All Insertion Structure".

II.4 All Insertion Structure and Analysis

We originally developed in [119] a procedure to synthesize privately enforcing insertion functions based on a special discrete structure called the *All Insertion Structure* (AIS). In this section, we start by reviewing the process of building the AIS, but following the procedure in [121], which is more efficient than the one in [119, 120].

II.4.1 Construction of the AIS

The review of the construction procedure of the AIS herein is necessary in order to explain how we employ this structure for the purposes of this chapter and also to define relevant notations. The AIS is a game-like bipartite structure between the system and the insertion function, with so-called Y states and Z states. When the system plays, it outputs an observable event e_0 that is defined at the current Y-state y of the AIS, and it leads to a Z-state $z = (y, e_0)$ in the AIS. On the other hand, when the insertion function plays, certain insertion decisions are made at Z-state z corresponding to strings that can be inserted before the last observed event e_0. As shown in [119], the AIS embeds in its transition structure *all* privately enforcing insertion functions.

There are three steps in the construction of the AIS: (1) building the i-verifier; (2) building the unfolded verifier; (3) obtaining the AIS. We start by describing step (1). First, we build the *desired* estimator $\mathscr{E}^d$ by deleting all the secret states from the original estimator $\mathscr{E}$ and taking the accessible part. As was mentioned earlier, $\mathscr{E} = (M, E_o, \delta, m_o)$ is the standard observer automaton of G with $M \subseteq 2^X$. Therefore, by construction, $\mathscr{E}^d$ generates exactly the safe language L_{safe}. We define the resulting sub-automaton of $\mathscr{E}$ as $\mathscr{E}^d = (M_d, E_o, \delta_d, m_o)$.

Next, we build the *feasible* estimator $\mathscr{E}^f$, which includes *all* possible insertions: we insert a self-loop at each state for each observable event, unless that self-loop is already defined in $\mathscr{E}$. We will use the new transition function δ_{sl} to denote those inserted self-loop transitions, and only those, in $\mathscr{E}^f$. Therefore, we obtain $\mathscr{E}^f = (M, E_o, \delta, \delta_{sl}, m_o)$. Hereafter, we wish to distinguish between two sets of transitions, normal and inserted ones, in $\mathscr{E}^f$; this is why we use two transition functions in its definition.

Finally, we synchronize $\mathscr{E}^d$ and $\mathscr{E}^f$ by a special type of parallel composition called *verifier parallel composition*, resulting in a new automaton called the *verifier*. All possible insertion functions are included in this automaton. The verifier parallel composition is denoted by $\|_v$. It is a synchronization between two kinds of automata, one with only "normal" transitions and the other with both "normal" and "inserted" self-loop transitions. Since we wish to again distinguish between these two sets of transitions, we use two transition functions in the definition of the i-verifier V, as was done above in $\mathscr{E}^f$.

Definition II.4.1 (Verifier parallel composition $\|_v$)**.** *The verifier parallel composition is a special kind of parallel composition between automata $\mathscr{E}^d$ and $\mathscr{E}^f$. Two kinds of transition functions, $\delta_{vs} : (M_d \times M) \times E_o \to (M_d \times M)$ and $\delta_{vd} : (M_d \times M) \times E_o \to (M_d \times M)$, are defined for synchronization:*

$$V := (M_v, E_o, \delta_{vd}, \delta_{vs}, m_{v0}) = \mathscr{E}^d \|_v \mathscr{E}^f =$$

$$Ac(M_d \times M, E_0, \delta_{vd}, \delta_{vs}, (m_0, m_0))$$

where the transition functions are defined as

$$\delta_{vs}((m_d, m_f), e) := (\delta_d(m_d, e), \delta(m_f, e))$$

$$\delta_{vd}((m_d, m_f), e) := (\delta_d(m_d, e), \delta_{sl}(m_f, e)) = (\delta_d(m_d, e), m_f)$$

The first equation corresponds to a normal transition labeled by e in both $\mathscr{E}^d$and $\mathscr{E}^f$; the second equation corresponds to a normal transition labeled by e in $\mathscr{E}^d$ and an inserted self-loop transition labeled by e in $\mathscr{E}^f$.

Hereafter, we assume that the two transition functions δ_{vs} and δ_{vd} are extended to strings of events in E_o.

In step (2) of the AIS construction, we "unfold" all deterministic insertion decisions from the i-verifier resulting in a game structure between the "system player" G and the "insertion function player"; we call this structure the *unfolded verifier*. This unfolding procedure is given in Algorithm 1 in [121]. The essence of the construction is to: (i) include all possible system plays, i.e., newly-generated observable events, at a given Y-state, and (ii) include all insertions that are possible *before* that observable event at a given Z-state, based on existing paths of inserted transitions in the i-verifier.

In order to synthesize admissible insertion functions, in step (3) of the AIS construction, we follow Algorithm 2 in [121] to prune away all the inadmissible insertion decisions (i.e., those that lead to deadlock at Z-states, since the insertion function should always play) from the unfolded i-verifier and call the final bipartite structure the AIS. This iterative pruning and associated trimming is described in Algorithm 2 in [121]. As explained in [121], it can be interpreted as a supremal controllable sublanguage calculation. Notice that there may be multiple paths of inserted events between two states m_v and m_v' in V and this is captured by the function $Ins(m_v, m_v') = \{s_I \in E_o^* : \delta_{vd}(m_v, s_I) = m_v'\}$ in Section IV.A of [121]. (In contrast with [121], we do not use the notation E_i in this chapter since it is the same as E_o.) Notice that $Ins(m_v, m_v')$ may be an infinite set if there is a cycle of inserted events in the path from m_v to m_v'. In this chapter, we make the assumption that such cycles are redundant (from the viewpoint of event insertion) and extract only the finite set of cycle-free paths from m_v to m_v', i.e., cycles of inserted events are replaced by ϵ.

The function Ins is used in line 5 of Algorithm 2 in [121] to label transitions from Z-states to Y-states in the AIS as sets of admissible strings that can be inserted when such transitions are taken. For the sake of simplicity of notation, we denote hereafter these sets by $L(z, y)$ for a given transition between state z and state y. It can be shown from the construction of V and of the AIS that any two $L(z, y_i)$ and $L(z, y_j)$ are disjoint for any two distinct successors y_i and y_j of z. Moreover, these sets are all finite since cycles of inserted events have been removed as mentioned above. As defined, the AIS does not pre-specify which string in an $L(z, y)$ set is to be selected and thus all the possible insertion choices are encoded in it. The reader is referred to [119, 121] for further details. As shown in [119], *opacity is privately enforceable if and only if the AIS is not empty.*

For the sake of completeness, we formally define this bipartite transition system. Let $I =$

$M_d \times M$ denote the set of all information states.

Definition II.4.2 (All Insertion Structure)**.** *The All Insertion Structure w.r.t. current-state estimator $\mathscr{E}$ is the tuple: $AIS = (Y, Z, E_o, 2^{E_o^*}, f_{AIS,yz}, f_{AIS,zy}, \Gamma, y_0)$, where*

- *$E_o \subseteq E$ is the set of observable events.*
- *$Y \subseteq I$ is the set of Y-states.*
- *$Z \subseteq I \times E_0$ is the set of Z-states. Let $I(z)$ denote the information state component in Z; then $z = (I(z), e)$ for some $e \in E_o$.*
- *$f_{AIS,yz} : Y \times E_0 \to Z$ is the transition function from Y-states to Z-states.*
- *$f_{AIS,zy} : Z \times 2^{E_o^*} \to Y$ is the transition function from Z states to Y states.*
- *$\Gamma : Z \to 2^{E_o^*}$ is the set of insertion choices at Z states defined as follows:*
  $\Gamma(z) = \bigcup\{L(z,y) : f_{AIS,zy}(z, L(z,y))$ *is defined* $\}$
- *$y_0 \subseteq Y$ is the initial Y state where $y_0 = (m_0, m_0)$ and m_0 is the initial state of $\mathscr{E}$.*

Example II.4.1. *Here we show an example to illustrate the whole construction process of the AIS. The current state estimator $\mathscr{E}$ is the same as in Example II.3.1 and is shown in Figure II.1. In this example, states 7 and 8 are secret states, so we delete them as well as transitions leading to them and then obtain the desired estimator $\mathscr{E}^d$ in Figure II.2. Next, we add self-loops for events $\{a, b, c, d\}$ at each state of $\mathscr{E}$ and obtain the feasible estimator $\mathscr{E}^f$ in Figure II.3. After that, we do the verifier parallel composition between $\mathscr{E}^d$ and $\mathscr{E}^f$ and obtain verifier V in Figure II.4. Notice that dashed transitions that are not followed by any solid transition are not shown in the figure. Those transitions do not indicate valid insertions and play no role in building the unfolded verifier. By the insertion mechanism, events are inserted before the occurrence of the next observable event, thus every δ_{vd} transition should be followed by a δ_{vs} transition somewhere in the verifier. Then we construct the unfolded verifier in Figure II.5, where the rectangular states are Y states and the oval states are Z states. As is seen in the figure, Z state $((6,6), c)$ is a deadlock state and should*

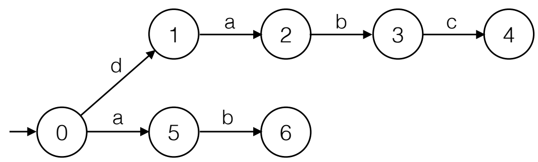

Figure II.2: Desired estimator $\mathscr{E}^d$

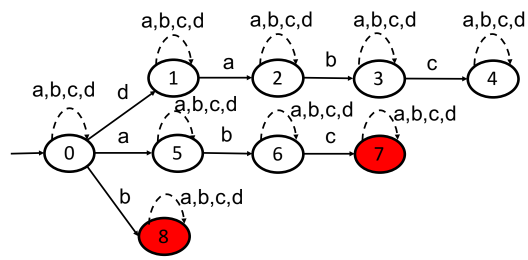

Figure II.3: Feasible estimator $\mathscr{E}^f$

be pruned away in the next step of building the AIS. Following Algorithm 2 in [121], the shaded path in V_u *is pruned away. Finally, we obtain the AIS in Figure II.6. The game starts at the initial Y-state* $(0,0)$ *where the system plays; initially the system can output a, b, or d. If the system outputs b, the game then reaches Z-state* $((0,0),b)$*, where the insertion function plays. The transition a between states* $((0,0),b)$ *and* $(6,8)$ *stands for insertion of event a and all the other transitions from Z states to Y states can be interpreted similarly. The insertion function can choose to insert a or da, leading the system to state* $(6,8)$ *or* $(3,8)$*, respectively.*

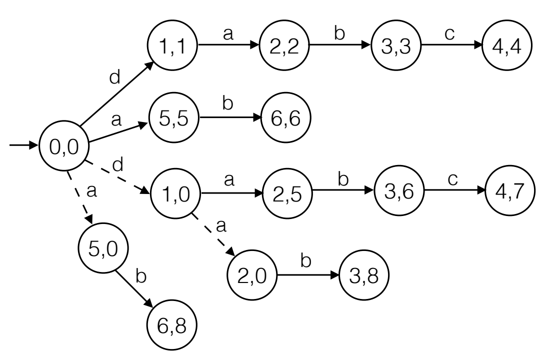

Figure II.4: Verifier V without dangling δ_{vd} transitions

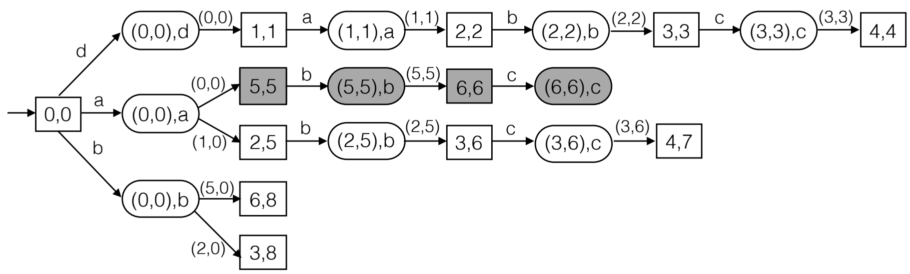

Figure II.5: Unfolded verifier V_u

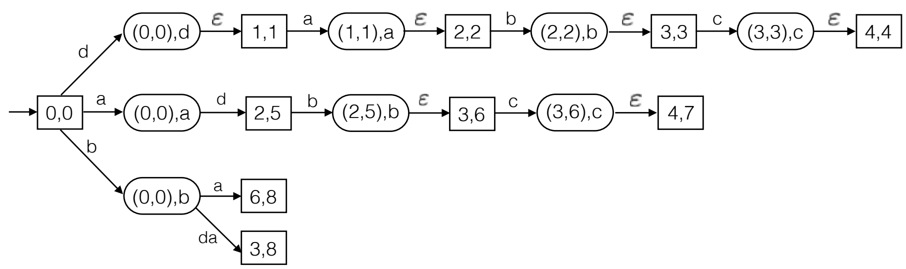

Figure II.6: AIS in Example II.4.1

II.4.2 Analysis of AIS

In the AIS, the insertion function works as follows: it observes some events and then makes a decision to insert a specific string before the observed event. This process continues as long as the system generates new observations. In order to better characterize this fact, we define the notion of *run* in the AIS:

Definition II.4.3 (Run)**.** *A run ω in the AIS is a sequence of alternating states, observable events and insertion decisions.*

$$\omega = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \cdots y_{n-1} \xrightarrow{e_{n-1}} z_{n-1} \xrightarrow{s_{n-1}} y_n \rangle$$

where $n \in \mathbb{N}$, y_0 is the initial state of the AIS, $e_i \in E_o$, $s_i \in E_o^*$, s.t., $f_{AIS,yz}(y_i, e_i) = z_i$, $s_i \in L(z_i, y_{i+1})$ where $f_{AIS,zy}(z_i, L(z_i, y_{i+1})) = y_{i+1}$, $\forall i,\ 0 \leq i < n$. The set of runs is denoted by Ω.

In the definition of run, the insertion choice is determined at each Z state, so we explicitly use an insertion string from the set of strings labeling a transition out of the Z-state. The length n of a run can be arbitrarily long. We require that a run of finite length could only end at Y-states, since these are the only possible terminating states in the AIS and this structure embeds only admissible insertion functions. A Y-state y is *terminating* if $f_{AIS,yz}(y, e_o)$ is undefined for all $e_o \in E_o$.

If we erase all the states from a run and swap every consecutive e_i and s_i pair, then by construction of the AIS, we get a *string generated by a run.*

Definition II.4.4 (String generated by a run)**.** *The string generated by run $\omega \in \Omega$ is defined as: $S(\omega) = s_0 e_0 s_1 e_1 \cdots s_{n-1} e_{n-1}$, given $\omega = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \cdots y_{n-1} \xrightarrow{e_{n-1}} z_{n-1} \xrightarrow{s_{n-1}} y_n \rangle$.*

From the definition of safe language, we observe that some safe strings are prefixes of unsafe strings while others are not. Based on this observation, the safe language is partitioned as follows:

Definition II.4.5 (Partition of safe language)**.** *Safe language L_{safe} is partitioned as:*

(1) $L^1_{safe} = \overline{\tilde{L}_{safe}}$ where $\tilde{L}_{safe} = \{s \in L_{safe} : \nexists u \in L_{unsafe},\ s.t.,\ s < u\}$.

(2) $L^2_{safe} = L_{safe} \setminus L^1_{safe}$.

Clearly, it is a partition of the safe language. Also L^1_{safe} is prefix-closed by definition but L^2_{safe} may not be prefix-closed. For strings in L^1_{safe}, we can choose not to insert in the AIS since they are already safe and we could also choose to insert as long as the insertion is feasible in the AIS. However, for strings in L^2_{safe}, we have to insert somewhere to obtain a string in L^1_{safe}, otherwise the secret states would be ultimately reached and private opacity would be violated. We already know that $L_{safe} \neq \emptyset$ if private safety is enforceable. Furthermore, the following proposition shows the non-emptiness of L^1_{safe} when private safety is enforceable.

Proposition II.4.1. *$L^1_{safe} \neq \emptyset$ if private safety is enforceable.*

Proof. Proof by contradiction. If $L^1_{safe} = \emptyset$, then $\forall s \in P[\mathcal{L}(G)]$, $\exists u \in L_{usafe}$, s.t. $s < u$. Since all the continuations of unsafe strings are also unsafe, we can never map an unsafe string to a string in L^1_{safe}. Then there always exists a string $u' \in L_{unsafe}$, such that no matter what the privately safe insertion function f_I is and what it inserts, $f_I(u') \in L_{unsafe}$, which violates private enforceability. □

II.5 PP-Enforcing Insertion Functions

Our goal is to exploit the AIS to synthesize PP-enforcing insertion functions. In that regard, we establish a necessary and sufficient condition for the existence of PP-enforcing insertion functions. We will proceed in two steps, first establishing preliminary results in Section II.5.1 before presenting the main necessary and sufficient condition in Section II.5.2.

II.5.1 A Sufficient condition for PP-enforcing Insertion Functions

Based on the definitions, a privately safe f_I maps all strings in $P[\mathcal{L}(G)]$ to a subset of L_{safe}. However, in general, $f_I^{str}[P(L_S)]$ may not be a subset of $f_I^{str}(L_{safe})$. In this case, the intruder, when knowing the implementation of f_I, could determine the occurrence of the secret when it observes strings in $f_I^{str}[P(L_S)] \setminus f_I^{str}(L_{safe})$. If, on the other hand, $f_I^{str}[P(L_S)]$ is contained in $f_I^{str}(L_{safe})$, then $f_I^{str}(P[\mathcal{L}(G)]) = f_I^{str}(L_{safe})$ and thus f_I is PP-enforcing. A special case where $f_I^{str}[P(L_S)]$ is guaranteed to be contained in $f_I^{str}(L_{safe})$ is when $f_I^{str}(L_{safe})$ is the entire set L_{safe}. Based on this special case, Lemma II.5.1 and Theorem II.5.1 below show sufficient conditions for a privately enforcing f_I to be PP-enforcing.

Lemma II.5.1. *Consider privately enforcing insertion function f_I. If $f_I^{str}(L_{safe}) = L_{safe}$, then f_I is also publicly enforcing; that is, f_I is PP-enforcing.*

Proof. Because a privately enforcing insertion function f_I is admissible, we can prove this Lemma using the definition of PP-enforceability. We will show that if $f_I^{str}(L_{safe}) = L_{safe}$, then the definition is satisfied. First, f_I is admissible and privately safe from the statement. We then show f_I is also publicly safe to complete the proof: if $f_I^{str}(L_{safe}) = L_{safe}$, then $f_I^{str}(P[\mathcal{L}(G)]) \subseteq L_{safe} = f_I^{str}(L_{safe})$. So f_I is PP-enforcing. $\square$

We now replace L_{safe} with a subset $L \subseteq L_{safe}$ and follow the argument in the proof of Lemma II.5.1 to derive a more general condition in Theorem II.5.1 (proof omitted since similar to that of Lemma II.5.1).

Theorem II.5.1. *Consider privately enforcing insertion function f_I, if there is $L \subseteq L_{safe}$ such that $f_I^{str}(P[\mathcal{L}(G)]) = L$ and $f_I^{str}(L) = L$, then f_I is also publicly enforcing; i.e., f_I is PP-enforcing.*

The condition in Theorem II.5.1 is sufficient and the following example shows a case when the theorem does not hold. Thus it remains to be seen whether a PP-enforcing insertion function can always be synthesized from the AIS.

Example II.5.1. *Consider system G with observable event set $E_o = \{a,b,c,d\}$ and observable language $P[\mathcal{L}(G)] = \overline{\{dabc, abc, bc, c\}}$, where $L_{safe} = \overline{\{dabc, abc, c\}}$. Define f_I so that $f_I(\epsilon, a) = da$, $f_I(\epsilon, b) = ab$, $f_I(\epsilon, c) = abc$ and $f_I(s, e_o) = e_o$ otherwise. Because $f_I^{str}[\mathcal{L}(G)] = \overline{\{abc, dabc\}} \subseteq L_{safe}$, f_I is privately enforcing. One can also check that f_I is publicly enforcing. However, the only set $L \subseteq L_{safe}$ satisfying $f_I^{str}(L) = L$ is $\overline{\{dabc\}}$, which is not equal to $f_I^{str}[\mathcal{L}(G)] = \overline{\{abc, dabc\}}$. Hence, f_I is a PP-enforcing insertion function such that no $L \subseteq L_{safe}$ satisfies $f_I^{str}[\mathcal{L}(G)] = L$ and $f_I^{str}(L) = L$.*

II.5.2 Greedy PP-enforcing Insertion Functions

In this section, we introduce the notion of a *greedy-maximal* PP-enforcing insertion function and then leverage the results in Section II.4.2 together with Theorem II.5.1.

First, we partition the set of Z states in the AIS into three subsets: (i) Z_1, defined as the Z states where the only insertion defined is ϵ; (ii) Z_2, defined as the Z states where both ϵ and non-ϵ transitions are defined; (iii) Z_3, defined as the remaining Z states, where no ϵ transitions are defined. If we track the runs generating L_{safe}^1, all the Z states should belong to Z_1 or Z_2, while for the runs generating L_{safe}^2 and L_{unsafe}, they should contain some Z_3 states.

Definition II.5.1 (Greedy-maximal criterion)**.** *(i) At any $z \in Z_1 \cup Z_2$ in the AIS, choose ϵ insertion; (ii) At any $z \in Z_3$ in the AIS, choose for insertion choice any string $s_{max} \in \arg\max[\ |s_i|,\ s_i \in \Gamma(z)]$ where $|\cdot|$ denotes the length of the string.*

Any insertion function that satisfies the greedy-maximal criterion at every Z-state that it visits in the AIS is called a *greedy-maximal insertion function*, denoted as f_{greedy}. By this criterion,

$f_{greedy}(L^1_{safe}) = L^1_{safe}$ since ϵ is chosen at every Z state. Moreover, $f_{greedy}(L^2_{safe} \cup L_{unsafe}) \subseteq L^1_{safe}$, a fact established below in the proof of Theorem II.5.2. In order to prove that theorem, we fist give definition of a particular projection P_e.

Definition II.5.2 (Projection P_e)**.** *Given a run* $\omega = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \cdots y_{n-1} \xrightarrow{e_{n-1}} z_n \xrightarrow{s_{n-1}} y_n \rangle$ *where* y_0 *is the initial state of the AIS, the edit projection* P_e *returns the string* $P_e(\omega) = s = e_0 e_1 \cdots e_{n-1}$.

Intuitively speaking, this projection just erases all the insertion choices from a run, and recovers the original string corresponding to the run. We can now state one of the main results in this chapter.

Theorem II.5.2. *A greedy-maximal insertion function is PP-enforcing.*

Proof. Consider greedy-maximal insertion function f_{greedy}. First, by Proposition II.4.1, $L^1_{safe} \neq \emptyset$. We also know that $\forall s \in L^1_{safe}$, $f_{greedy}(s) = s$, i.e., $f_{greedy}(L^1_{safe}) = L^1_{safe}$ by our greedy criterion.

Next, we show that $f_{greedy}(L^2_{safe} \cup L_{unsafe}) \subseteq L^1_{safe}$. $\forall s \in L^2_{safe} \cup L_{unsafe}$, let $f_{greedy}(s) = s'$, where we know that $\exists \omega \in \Omega$ s.t., $P_e(\omega) = s$ and $S(\omega) = s'$. Then we claim that $\exists \omega' \in \Omega$, s.t., $(P_e(\omega') = s') \wedge (f_{greedy}(s') = s')$, which we prove by contradiction. We know that actually $(f_{greedy}(s') = s') \Rightarrow (P_e(\omega') = s')$, and we focus on showing $f_{greedy}(s') = s'$. Suppose this is not the case, then $f_{greedy}(s') = s'' \neq s'$ and $S(\omega') \neq s'$. So $\exists z \in Z_3$ in ω' where only non-ϵ insertion is feasible. However, the AIS embeds all admissible insertion choices and this implies f_{greedy} does not choose a longest insertion choice at certain $z \in Z_3$ in ω, which leaves the possibility for non-ϵ insertion in ω'. This contradicts with the insertion mechanism of f_{greedy}. Therefore, $\forall z \in \omega'$, $z \in Z_1 \cup Z_2$, $f_{greedy}(s') = s' \in L^1_{safe}$, in other words, $f_{greedy}(L^2_{safe} \cup L_{unsafe}) \subseteq L^1_{safe}$. Overall, $f_{greedy}(P[\mathcal{L}(G)]) = L^1_{safe}$ and this implies f_{greedy} and L^1_{safe} satisfy Theorem II.5.1, thus f_{greedy} is PP-enforcing.

□

This theorem demonstrates that as long as the AIS is not empty, then there exists at least one greedy-maximal insertion function that is also PP-enforcing. This leads to the following corollary.

Corollary II.5.1. *Opacity is PP-enforceable if and only if it is privately enforceable.*

Proof. The only if part is true since the definition of PP-enforceability implies private enforceability.

For the if part, as long as the AIS is not empty, we could always make insertion choices by this greedy criterion at every Z state and get a PP-enforcing insertion function. □

This result is a direct improvement of the preliminary work [120] in the sense that PP-enforcing insertion function always exists as long as privately safe insertion function exists. Let us revisit Example II.5.1: it is clear that f_I is not greedy-maximal since $f_I(\epsilon, c) \neq dabc$. If we set $f_I(\epsilon, c) = dabc$, then we obtain a greedy-maximal insertion function that is PP-enforcing.

II.6 The INPRIVALIC-G Algorithm

In this section, we develop a new algorithm that synthesizes a PP-enforcing insertion function by leveraging Theorem II.5.2. We first build the AIS, which embeds all privately enforcing insertion functions. The strategy of the proposed algorithm is to identify L^1_{safe} and modify all other strings to strings in L^1_{safe} by using the greedy-maximal criterion. As a result, any insertion function synthesized in that manner is guaranteed to be PP-enforcing by Theorem II.5.2.

Because this algorithm synthesizes INsertion functions with PRIVAte-and-pubLIC-enforceability property using Greedy-maximal criterion, we call it the INPRIVALIC-G Algorithm. Hereafter, we denote a greedy-maximal insertion function by f_{greedy}.

Algorithm II.1: INPRIVALIC-G ALGORITHM

Input : $G = (X, E, f, X_0)$, projection P, $X_s \subseteq X$
Output: A PP-enforcing IA
1 Build $\mathscr{E}, \mathscr{E}^d, \mathscr{E}^f$;
2 $V = \mathscr{E}^d \|_v \mathscr{E}^f$;
3 Construct All Insertion Structure (AIS) by algorithms in [121];
4 Synthesize a greedy insertion function from AIS;

The INPRIVALIC-G Algorithm is not meant to synthesize *all* PP-enforcing insertion functions, but it is guaranteed to find one (unless the AIS is empty).

We discuss the steps of the algorithm, as a way of summarizing the methodology developed in this chapter. Steps 1 to 3 construct the AIS. These steps were already discussed earlier in Section II.4.1 and will not be repeated here. After that, step 4 synthesizes an insertion automaton from the AIS using the greedy-maximal criterion. The main idea is that at each Z-state in the AIS, a greedy-maximal insertion choice is selected according to Definition II.5.1 and this process proceeds until: (1) a terminating Y is reached; or (2) a previously visited Y state is visited again. It is implemented in Algorithm II.2, which builds the reachable part of the AIS for the selections made, until a complete IA is obtained.

Algorithm II.2: Synthesize a greedy insertion function

Input : $AIS = (Y, Z, E_o, 2^{E_o^*}, f_{AIS,yz}, f_{AIS,zy}, \Gamma, y_0)$

Output: IA=$(X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$

$x_{ia,0} := y_o, X_{ia} := \{x_{ia,0}\}$;

for $x_{ia} \in X_{ia}$ *that has not been examined* **do**

    for $e \in E_o$ *s.t.* $f_{AIS,yz}(x_{ia}, e)$ *is defined and where* $z = f_{AIS,yz}(x_{ia}, e) = (x_{ia}, e)$ **do**

        if $\epsilon \in \Gamma(z)$ **then**

            $x'_{ia} := f_{AIS,zy}(z, L(z, x'_{ia}))$ where $\epsilon \in L(z, x'_{ia})$;

            $f_{ia}(x_{ia}, e) = x'_{ia}$;

            $q_{ia}(x_{ia}, e) = e$;

            else

                pick one $s_{max} \in \arg\max[|s_i|, s_i \in \Gamma(z)]$;

                $x'_{ia} := f_{AIS,zy}(z, L(z, x'_{ia}))$ where $s_{max} \in L(z, x'_{ia})$;

                $f_{ia}(x_{ia}, e) = x'_{ia}$;

                $f_{ia}(x_{ia}, e) = x'_{ia}$;

        $X_{ia} := X_{ia} \cup \{x'_{ia}\}$;

return IA

The following running example shows all the steps of the INPRIVALIC-G Algorithm.

Example II.6.1. *Let automaton G with observable events* $E_o = \{a, b, c, d, e\}$ *have the state estimator shown in Figure II.7, where estimator state 7 reveals the secret. We use this example to illustrate all the steps of the INPRIVALIC-G Algorithm. Following the algorithm, we build the AIS and synthesize a PP-enforcing insertion function encoded by an I/O automation.*

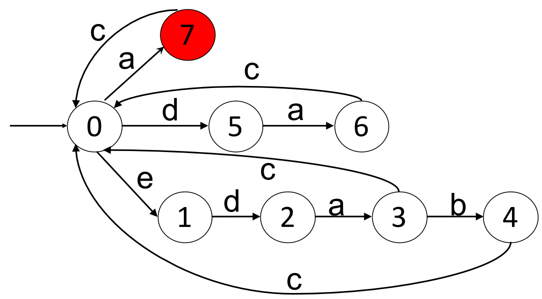

Figure II.7: $\mathscr{E}$ with secret-revealing state 7

In step 1, we build $\mathscr{E}^d$ by removing state 7 and we obtain $\mathscr{E}^f$ by adding self-loops for a,b,c,d,e at each state.

In step 2, we perform the verifier parallel composition of $\mathscr{E}^d$ and $\mathscr{E}^f$ and obtain V, which is not shown here.

In step 3, we unfold the insertions in V for every system output, and build the game structure V_u. Since there is no inadmissible insertion in V_u, no state will be pruned away and the AIS is immediately obtained in Figure II.8. There are two types of states in the AIS: square states where the system plays and round states where the insertion function plays.

With the AIS built, we proceed to the synthesis part. By the greedy-maximal criterion, at state $((0,0),a)$, ed should be inserted and at state $((3,7),c)$, ϵ should be inserted. Similarly for the other Z-states: we insert ϵ if it is defined. In Figure II.8 we use bold red lines to indicate the greedy-maximal criterion in the AIS. Finally, the insertion automaton in Figure II.9 encodes the constructed PP-enforcing insertion function.

We conclude with a brief discussion of the computational complexity of the INPRIVALIC-G Algorithm. Consider a system with estimator $\mathscr{E}$; as shown in [119], the AIS has at most $(|E_o|+1)|X_{\mathscr{E}}|^2$ states, where $|X_{\mathscr{E}}|$ is the number of states in $\mathscr{E}$. The time complexity for building the AIS is of $O(|X_{\mathscr{E}}|^6)$ according to [121]. Finally, the greedy-maximal synthesis step is done by performing a breadth-first search on the AIS, which requires time complexity linear in its size. In all, the computational complexity of the INPRIVALIC-G Algorithm is therefore of $O(|X_{\mathscr{E}}|^6)$. In the worst case, $|X_{\mathscr{E}}|$ may be $2^{|X|}$ and the complexity is exponential in terms of $|X|$. We refer the reader to [119]

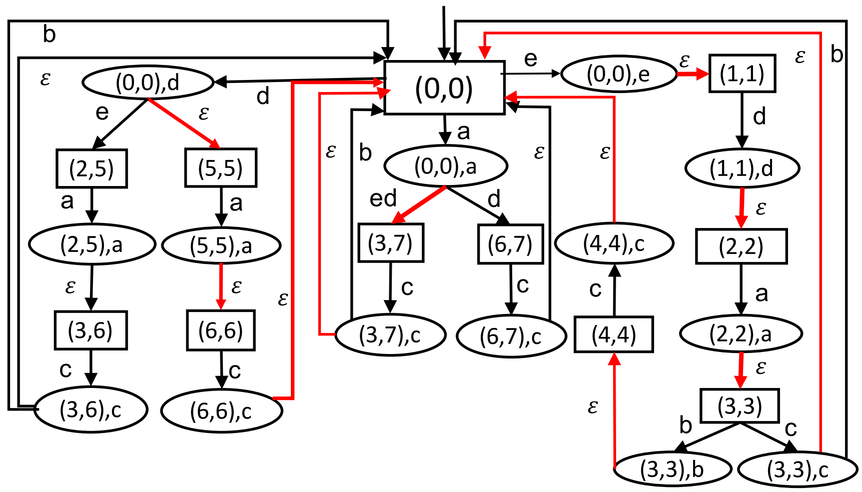

Figure II.8: All insertion structure with greedy-maximal criterion

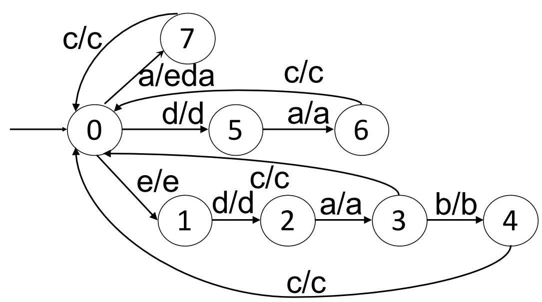

Figure II.9: A PP-enforcing insertion function encoded as an I/O automaton

for numerical tests on the construction of the AIS using an explicit representation, and to [122] for a symbolic implementation of the AIS construction using binary decision diagrams, which achieves greater scalability.

Remark II.6.1. *The INPRIVALIC-G Algorithm is sound and complete, unlike the INPRIVALIC Algorithm in [?], which was provably sound only.*

II.7 Conclusion

This chapter extends prior works on opacity enforcement by insertion functions to the case where the insertion function may become known to the intruder. To handle this situation, we defined the notion of public-private (PP) opacity and investigated its enforcement by so-called PP-enforcing insertion functions. We showed that while not all insertion functions that are privately-enforcing may be PP-enforcing, if private safety is enforceable, then so is public-private safety. In this regard, we identified a necessary and sufficient condition for PP-enforceability and then developed an algorithmic procedure for synthesizing insertion functions that are provably PP-enforcing. This algorithm (INPRIVALIC-G) is based on a greedy-maximal insertion mechanism.

This chapter also opens several avenues for future investigations. First, it would be of interest to extend the results herein to the case of *edit functions*, a generalized form of insertion functions. This problem will be discussed in the next chapter. Second, it would be worthwhile to identify other synthesis strategies than the greedy-maximal one of Algorithm INPRIVALIC-G to synthesize PP-enforcing insertion functions. Finally, it would be of interest to study instances where the intruder has partial knowledge of the insertion function, as opposed to the full-knowledge or no-knowledge scenarios considered in this chapter.

CHAPTER III

Opacity Enforcement using Nondeterministic Publicly-Known Edit Functions

III.1 Introduction

In last chapter, we assume that the edit function's implementation is known to the intruder and discuss how to defend secrets by insertion functions under such an adversary. As an extension, we try to solve the same problem by edit functions in this chapter. We further improve the results in [53, 54, 119, 122] by considering opacity enforcement using nondeterministic edit functions, whose outcome is randomly chosen from a pre-calculated set and the intruder does not know the result a priori. Both private safety and public safety are defined for edit functions to characterize their performance. Although nondeterministic edit functions seem to release more information to the intruder by allowing more potential outcomes, they essentially provide the system more plausible denial of secret disclosure, which contributes to opacity enforcement. It is shown that a nondeterministic edit function may still achieve private and public safety even when its deterministic counterpart fails to do so. To the best of our knowledge, this chapter for the first time considers nondeterminism of the defender in opacity enforcement. We introduce a three-player game structure termed *All Edit Structure (AES)* to embed edit functions. An algorithm is developed to synthesize privately and publicly safe nondeterministic edit functions based on the AES.

The remaining sections are organized as follows. Section V.2 presents the system model. Section III.3 formally introduces the notions of nondeterministic edit functions, private safety and public safety. Section III.4 defines the three-player observer (TPO), discusses its properties and introduces edit constraints. Section III.5 defines a special TPO called All Edit Structure (AES) and presents its construction algorithm. Section III.6 develops an algorithm for synthesizing nondeterministic publicly and privately safe edit functions based on the reachability tree of the AES. Finally, Section V.7 concludes the chapter.

III.2 System Model

We consider opacity in the framework of discrete event systems modeled as deterministic finite-state automata [23]:

$$G = (X, E, f, x_0)$$

where X is the finite set of states, E is the finite set of events, $f : X \times E \to X$ is the partial state transition function and $x_0 \in X$ is the initial state. Specifically, we denote by $X_S \subset X$ the set of *secret states*. The transition function is extended to domain $X \times E^*$ in the standard manner [23]. Given two strings s, u, we denote by $s \leq u$ if s is a prefix u and $t \in s$ if t is a substring of s. The language generated by G is defined as $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined". Notice that the system model here is very similar to that in Chapter II, except that the initial state here is unique.

For simplicity, we write $x \xrightarrow{e} x'$, if $x' = f(x, e)$ for $x, x' \in X$ and $e \in E$. Given system G, a *run* is a sequence of alternating states and events $x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$, where $\forall i \leq n$, $x_i \in X$ and $e_i \in E$. A run contains a *cycle* if $\exists 1 \leq i < j \leq n$, s.t. $x_i = x_j$.

The system is partially observed with the event set E partitioned as $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events. Given a string $t \in E^*$, its natural projection $P : E^* \to E_o^*$ is recursively defined as $P(t) = P(t'e) = P(t')P(e)$ where $t' \in E^*$

and $e \in E$. The projection of an event is $P(e) = e$ if $e \in E_o$ and $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string. Then by the standard technique in [23], the *observer* of G is defined as: $Obs(G) = (X_{obs}, E_o, \delta, x_{obs,0})$, where $X_{obs} \subseteq 2^X$ is the state space, E_o is the set of observable events, $\delta : X_{obs} \times E_o \to X_{obs}$ is the transition function and $x_{obs,0} \in X_{obs}$ is the initial state. An observer state can be viewed as an *estimate* of the system's current states. Therefore, the observer is often called "state estimator" in the literature, e.g., [119].

III.3 Edit Functions and Opacity Notions

In this section, we formally define *nondeterministic* edit functions and discuss the edit mechanism. We also define *private safety* and *public safety* to further characterize how the edit function defends the secrets of the system against intruders with different knowledge.

III.3.1 Edit Mechanism

We first review the concept of deterministic edit function in [53]: $f_e : E_o^* \times E_o \to E_o^* E_o^\epsilon$ where $E_o^\epsilon = E_o \cup \{\epsilon\}$. Given $s \in P[\mathscr{L}(G)]$, $e_o \in E_o$, $f_e(s, e_o) = s_I e_o$ if s_I is inserted before e_o; $f_e(s, e_o) = \epsilon$ if e_o is erased; $f_e(s, e_o) = s_I$ if s_I is inserted and e_o is erased.

By definition, the outcome of a deterministic edit function is unique. Then we extend it and define a *nondeterministic edit function*: $f_{ne} : E_o^* \times E_o \to 2^{E_o^* E_o^\epsilon}$ that outputs a string nondeterministically from a set of potential outcomes. Its output is based on the past observed string and the current observed event. Given an observable string $s \in P[\mathscr{L}(G)]$ and an observable event $e_o \in E_o$, a potential outcome of a nondeterministic edit function may be $s_I e_o$ if s_I is inserted before e_o or ϵ if e_o is erased or s_I if s_I is inserted and e_o is erased. In contrast to deterministic edit functions in [53], the outcome is not pre-calculated and is chosen randomly when the nondeterministic edit function is implemented. Notice that s_I may be ϵ so that nothing is inserted. The outcome of such a function is not known by the intruder before it is observed. With a slight abuse of notation, we also define a string based nondeterministic edit function f_{ne} recursively as: $f_{ne}(\epsilon) = \{\epsilon\}$ and

$f_{ne}(se_o) = \{l_p l_s \in E_o^* : l_p \in f_{ne}(s), l_s \in f_{ne}(s, e_o)\}$.

An edit function is an interface between the system's output and the outside world, which includes the intruder eavesdropping on the system. The edit function works as follows: upon observing a string, it makes a decision to insert fictitious events before the last observed event or to erase the last observed event; then the edited string is emitted as the actual output. We assume that all observable events E_o are allowed to be inserted or erased, and the intruder cannot distinguish between an inserted event and its genuine counterpart. We define $E_o^r = \{e_o \to \epsilon : e_o \in E_o\}$ to be the set of "event erasure" events. In this chapter, if we concatenate an "event erasure" event $e_o \to \epsilon$ with the observable event e_o, the result is simply ϵ.

Given a nondeterministic edit function f_{ne}, the intruder infers secrets from its *current state estimate* $\mathcal{E}_{f_{ne}} : P[\mathcal{L}(G)] \to 2^{X_{obs}}$ and $\mathcal{E}_{f_{ne}}(s) = \{x_{obs} \in X_{obs} : \exists t \in f_{ne}(s),\ \text{s.t. } x_{obs} = \delta(x_{obs,0}, t)\}$. Since f_{ne} is nondeterministic, $\mathcal{E}_{f_{ne}}(s)$ is generally a set of states in X_{obs}.

III.3.2 Private Safety and Public Safety

In this subsection, we first review the well-studied concept of *current-state opacity* (Definition II.3.1) and then derive two concepts from it.

A system is current-state opaque if for every string reaching a secret state, there exists another string reaching a non-secret state and both strings share the same projection. CSO can be verified by building the observer and checking whether any observer state contains solely secret states. If CSO is violated, an edit function may be used to enforce opacity, which is the problem studied in this chapter

Based on CSO, we define the *safe language* [119] as: $L_{safe} = P[\mathcal{L}(G)] \setminus \{[P[\mathcal{L}(G)] \setminus P(L_{NS})]E_o^*\}$, which is prefix-closed. While the *unsafe language* is $L_{unsafe} = P[\mathcal{L}(G)] \setminus L_{safe}$. Intuitively, we view all observable continuations of $P[\mathcal{L}(G)] \setminus P(L_{NS})$ as "unsafe". If we delete all states violating CSO from $Obs(G)$, i.e., all observer states that solely contain secret states, and then take the accessible part, the resulting automaton just generates L_{safe}. We call it *desired observer*: $Obs_d(G) = (X_{obsd}, E_o, \delta_d, x_{obsd,0})$, see [54, 119] for more details.

Inspired by *private safety* and *public safety* of insertion functions in [54], we redefine those two concepts for nondeterministic edit functions and call them *nondeterministic (ND)-private safety* and *nondeterministic (ND)-public safety*, respectively.

Definition III.3.1 (ND-Private Safety)**.** *Consider system G with P, L_{safe} and $Obs_d(G)$, a nondeterministic edit function f_{ne} is privately safe, if $\forall s \in P[\mathscr{L}(G)]$, $f_{ne}(s) \subseteq L_{safe}$.*

If f_{ne} is privately safe, we denote it by $f_{ne} \vDash \varphi_{ndpri}$ where φ_{ndpri} stands for ND-private safety. ND-private safety is based on the assumption that the intruder does not know about the implementation of edit functions. Thus, as long as for a given string s and an edit function f_{ne}, every element in $f_{ne}(s)$ is also in L_{safe}, then the intruder's state estimate would never reveal the secrets of the system.

Definition III.3.2 (ND-Public Safety)**.** *Consider a system G, L_{safe} and L_{unsafe}, a nondeterministic edit function f_{ne} is publicly safe, if $\forall s \in L_{unsafe}$, $\forall \tilde{s} \in f_{ne}(s)$, $\exists t \in L_{safe}$, s.t. $\tilde{s} \in f_{ne}(t)$.*

If f_{ne} is publicly safe, we denote it by $f_{ne} \vDash \varphi_{ndpub}$ where φ_{ndpub} stands for ND-public safety. ND-public safety is based on the assumption that the implementation of edit functions is known to the intruder. A sophisticated intruder may learn the implementation of the edit function and potentially does some reverse engineering to infer the source of the edited string. Thus, for ND-public safety, we require that no matter how an unsafe string is edited, it should share the same edited behavior with some safe string. As the intruder does not know how a string is edited before it makes an observation, ND-public safety and ND-private safety guarantee that the system's secrets are never disclosed. A nondeterministic edit function f_{ne} is *ND-public-private enforcing* (ND-PP-enforcing), denoted by $f_{ne} \vDash \varphi_{ndpp}$, if $f_{ne} \vDash \varphi_{ndpri}$ and $f_{ne} \vDash \varphi_{ndpub}$. In this chapter, we require that an edit function should be able to map every string in $P[\mathcal{L}(G)]$ to some strings and we term this property as *admissibility*.

III.4 Three-Player Observer

In this section, we propose the *Three-Player Observer (TPO)*, which is a three-player game structure that provides a systematic way of embedding edit functions and evaluating their performance. Then we discuss some properties of the TPO and define *edit constraints*.

The TPO is an *information-state-based* structure, whose *current* state contains enough information for analysis of opacity enforcement and no *future* information is necessary. We denote the set of *information states* as I. The formal definition is as follows:

Definition III.4.1 (Three-Player Observer)**.** *Given a system G, its observer $Obs(G)$ and desired observer $Obs_d(G)$, let $I \subseteq X_{obsd} \times X_{obs}$ be the set of information states. A three-player observer is the tuple $T = (Q_Y, Q_Z, Q_W, E_o, E_o^r, \Theta, f_{yz}, f_{zz}, f_{zw}^{in}, f_{zw}^{er}, f_{wy}^{in}, f_{wy}^{er}, y_0)$, where*

- *$Q_Y \subseteq I$ is the set of information states.*
- *$Q_Z \subseteq I \times E_o$ is the set of information states augmented with observable events. Let $I(z)$, $E(z)$ denote the information state component and observable event component of $z \in Q_Z$ respectively, so that $z = (I(z), E(z))$.*
- *$Q_W \subseteq I \times (E_o \cup E_o^r)$ is the set of information states augmented with observable events or event erasure events. Let $I(w)$, $A(w)$ denote the information state component and edit action component of $w \in Q_W$ respectively, so that $w = (I(w), A(w))$.*
- *$E_o \subseteq E$ is the set of observable events.*
- *E_o^r is the set of "event erasure" events.*
- *$\Theta \subseteq E_o \cup \{\epsilon\} \cup E_o^r$ is the set of edit decisions at Q_Z-states.*
- *$f_{yz} : Q_Y \times E_o \to Q_Z$ is the transition function from Q_Y states to Q_Z states. For $y = (x_d, x_f) \in Q_Y$, $e_o \in E_o$, we have:*

$$f_{yz}(y, e_o) = z \Rightarrow [\delta(x_f, e_o)!] \wedge [I(z) = y] \wedge [E(z) = e_o]$$

- $f_{zz}: Q_Z \times \Theta \to Q_Z$ *is the transition function from* Q_Z*-states to* Q_Z*-states. For* $z = ((x_d, x_f), e_o) \in Q_Z$*,* $\theta \in \Theta$*, we have:*

$$\begin{aligned} f_{zz}(z,\theta) = z' \Rightarrow & [\theta \in E_o] \wedge [I(z') = (x'_d, x_f)] \\ & \wedge [x'_d = \delta_d(x_d,\theta)] \wedge [E(z') = e_o] \end{aligned}$$

- $f^{in}_{zw}: Q_Z \times \Theta \to Q_W$ *is the* ϵ*-insertion transition from* Q_Z*-states to* Q_W*-states. For* $z = ((x_d, x_f), e_o) \in Q_Z$*,* $\theta \in \Theta$ *we have:*

$$\begin{aligned} f^{in}_{zw}(z,\theta) = w \Rightarrow & [\theta = \epsilon] \wedge [I(w) = I(z)] \wedge [A(w) = e_o] \\ & \wedge [\delta_d(x_d, e_o)!] \wedge [\delta(x_f, e_o)!] \end{aligned}$$

- $f^{er}_{zw}: Q_Z \times \Theta \to Q_W$ *is the event erasure transition from* Q_Z*-states to* Q_W*-states. For* $z = ((x_d, x_f), e_o) \in Q_Z$*,* $\theta \in \Theta$*, we have:*

$$\begin{aligned} f^{er}_{zw}(z,\theta) = w \Rightarrow & [\theta = e_o \to \epsilon] \wedge [I(w) = I(z)] \\ & \wedge [A(w) = e_o \to \epsilon] \wedge [\delta(x_f, e_o)!] \end{aligned}$$

- $f^{in}_{wy}: Q_W \times E_o \to Q_Y$ *is the transition function from* Q_W*-states whose edit action component is in* E_o *to* Q_Y*-states. For* $w = ((x_d, x_f), e_o) \in Q_W$*, we have:*

$$\begin{aligned} f^{in}_{wy}(w, e_o) = y \Rightarrow & [y = (x'_d, x'_f)] \wedge [x'_d = \delta_d(x_d, e_o)] \\ & \wedge [x'_f = \delta(x_f, e_o)] \end{aligned}$$

- $f^{er}_{wy}: Q_W \times E_o \to Q_Y$ *is the transition function from* Q_W*-states whose edit action component is in*

E_o^r to Q_Y-states. For $w = ((x_d, x_f), e_o \to \epsilon) \in Q_W$, we have:

$$f_{wy}^{er}(w, e_o) = y \Rightarrow [y = (x_d, x_f')] \wedge [x_f' = \delta(x_f, e_o)]$$

- *$y_0 \in Q_Y$ is the initial Q_Y-state where $y_0 = (x_{obsd,0}, x_{obs,0})$. $x_{obsd,0}$ and $x_{obs,0}$ are the initial states of $Obs_d(G)$ and $Obs(G)$.*

The three-player observer is defined to describe the game among a "dummy" player, "edit function" and "system/environment". All three players have *complete information* in the sense that they know exactly the actions of each other at any moment of the game.

A Q_Y-state (Y-state) is an information state, from which the "dummy" player executes observable events. A Y-state contains both the intruder's estimate and the system's estimate. Actually, the events from Y-states do not really occur and they are the events to be observed by the edit function player. f_{yz} is defined only to help determine what edit decisions can be made by the edit function in the next step. That is why we call this player a dummy player.

A Q_Z-state (Z-state) is an information state augmented with the event executed by the dummy player, where the edit function makes decisions. If the edit function chooses to insert an event, a succeeding Z-state will be reached under an f_{zz} transition. If another event is inserted following the last inserted event, then another succeeding Z-state is reached until the edit function stops inserting. This corresponds to insertion of multiple events. If the edit function keeps inserting events, we can expect that a cycle of Z-states and f_{zz} transitions is formed in the TPO. When an event is inserted, only the intruder's estimate is updated while the system's estimate remains the same, which is reflected in defining f_{zz}. This is consistent with the edit function's mechanism as the edit function serves as an interface to modify the intruder's observation but does not interfere with the system's operation. When the edit function decides to stop insertion or to erase the last observed event, the turn of the game is passed to the system/environment player by f_{zw}^{in} and f_{zw}^{er} transitions. We denote by $f_{zw} = f_{zw}^{in} \cup f_{zw}^{er}$ where f_{zw}^{in} stands for ϵ-insertion (termination of insertion) and f_{zw}^{er} stands for erasure of the observable event executed by the dummy player. We will use f_{zw} for simplicity in the following discussion if there is no confusion. There may be multiple transitions defined out of a Z-state, i.e., multiple edit decisions, and we let $\Theta(z)$ be the set of edit decisions defined at $z \in Q_Z$ in a TPO.

A Q_W-state (W-state) is an information state augmented with an observable event or an "event erasure" event, from which the system plays. If a W-state contains an observable event, that means the edit function player has inserted ϵ from its preceding Z-state. When that event is executed, it will be observed by the intruder. Thus, an f_{wy}^{in} transition leads to a Y-state, whose first and second state components are both updated. If a W-state contains an "event erasure" event, that means the edit function has decided to erase the observable event. So when the event is executed, it will not be observed by the intruder. Hence, an f_{wy}^{er} transition leads to a Y-state, whose first state component (intruder's estimate) is updated while the second state component (system's estimate) remains unchanged. We just denote by $f_{wy} = f_{wy}^{in} \cup f_{wy}^{er}$ and will use f_{wy} when there is no confusion.

Given two TPOs T_1 and T_2, T_1 is a *subsystem* of T_2, denoted by $T_1 \sqsubseteq T_2$, if $Q_Y^{T_1} \subseteq Q_Y^{T_2}$, $Q_Z^{T_1} \subseteq Q_Z^{T_2}$, $Q_W^{T_1} \subseteq Q_W^{T_2}$ and $\forall y \in Q_Y^{T_1}$, $\forall z, z' \in Q_Z^{T_1}$, $\forall w \in Q_W^{T_1}$, $\forall e_o \in E_o$, $\forall \theta, \theta' \in \Theta$, we have: (1) $f_{yz}^{T_1}(y, e_o) = z \Rightarrow f_{yz}^{T_2}(y, e_o) = z$; (2) $f_{zz}^{T_1}(z, \theta) = z' \Rightarrow f_{zz}^{T_2}(z, \theta) = z'$; (3) $f_{zw}^{T_1}(z, \theta') = w \Rightarrow f_{zw}^{T_2}(z, \theta') = w$; (4) $f_{wy}^{T_1}(w, e_o) = y \Rightarrow f_{wy}^{T_2}(w, e_o) = y$.

A run in a three-player observer is of the form: $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, where y_0 is the initial state of T, $e_i \in E_o$, $\theta_i^j \in \Theta(z_i^j)$, $\forall 0 \leq i \leq n$, $1 \leq j \leq m_i$ and $n \in \mathbb{N}$, $m_i \in \mathbb{N}^+$. It characterizes the information flow in a TPO and we denote the set of runs in a TPO T by $Run(T)$. We also write $y_i \in r$ ($z_i \in r$ or $w_i \in r$) if y_i (z_i or w_i) is a state in r. A run corresponds to an unedited string and an edited string, then we have the following definitions.

Definition III.4.2 (String Generated by a Run)**.** *Given a run* $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, *the string generated by r is defined as:* $l_g(r) = \theta_0^1 \theta_0^2 \cdots \theta_0^{m_0-1} \theta_0^{m_0} e_0 \theta_1^1 \cdots \theta_1^{m_1} e_1 \cdots e_{n-1} \theta_n^1 \cdots \theta_n^{m_n} e_n$, *where* $\forall i \leq n$, $\theta_i^{m_i} e_i = \epsilon$ *if* $\theta_i^{m_i} = e_i \rightarrow \epsilon$.

Definition III.4.3 (Edit Projection). *In a TPO T, given a run* $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, *edit projection* $P_e : Run(T) \to P[\mathcal{L}(G)]$ *is defined such that* $P_e(r) = e_0 e_1 \cdots e_n$.

That is, the edit projection projects away the edit decisions in a run and "recovers" the unedited string. While the generated string of a run is just the string after considering the edit decisions.

From a given TPO, we may extract an edit function from it and we define the *edit function embedded in a TPO*. With a slight abuse of notation, we write $f_{ne} \in T$ if f_{ne} is embedded in T.

Definition III.4.4 (ND-Edit Function embedded in TPO). *Given a TPO T, nondeterministic edit function f_{ne} is embedded in T if* $\forall s \in P[\mathcal{L}(G)]$, $\forall \tilde{s} \in f_{ne}(s)$, $\exists r \in Run(T)$, *s.t.* $P_e(r) = s$ *and* $l_g(r) = \tilde{s}$.

In a TPO, $y \in Q_Y$ is a *terminating state* if $\nexists e_o \in E_o$, s.t. $f_{yz}(y, e_o)!$. And $w \in Q_W$ is a *deadlocking* state if $\nexists e_o \in E_o$, s.t. $f_{wy}(w, e_o)!$. Also $z \in Q_Z$ is a *deadlocking* state if $\nexists \theta \in \Theta$, s.t. $f_{zz}(z, \theta)!$ or $f_{zw}(z, \theta)!$. We call a TPO *complete* if: (1) there are no deadlocking W or Z states; (2) $\forall s \in P[\mathcal{L}(G)]$, $\exists r \in Run(T)$, s.t. $P_e(r) = s$. In a complete TPO, all embedded edit functions are admissible and they can always make a decision no matter what event occurs; also the events executed by the system can not be blocked from happening. From now on, we will only consider complete TPOs. Notice that a complete TPO only terminates at Y-states, being consistent with the definition of run.

In practice, the edit functions may be constrained by the outside environment or the preference of the system's designer so that certain edit decisions may not be taken and some Y-states may not be preferred. Thus, we introduce *constraints on edit decisions* and *constraints on Y-states*, both in a generic form.

Definition III.4.5 (Constraints on Edit Decisions). *The constraint on edit decisions is a binary function* $\phi_{dec} : \Theta \to \{0, 1\}$ *and an edit decision* $\theta \in \Theta$ *satisfies the constraint if* $\phi_{dec}(\theta) = 1$.

Definition III.4.6 (Constraints on Y-States). *The constraint on Y-states is a binary function* $\phi_y : Q_Y \to \{0, 1\}$ *and a Y-state* $y \in Q_Y$ *satisfies the constraint if* $\phi_y(y) = 1$.

Both constraints are problem-dependent and will be specified when a problem is discussed. They will reduce the state space of the TPO and bring in deadlocking states. In the following section, we will define the "largest" TPO satisfying both constraints.

III.5 All Edit Structure

In this section, we define a complete TPO such that: $[\forall y \in Q_Y : \phi_y(y) = 1] \wedge [\forall \theta \in \Theta : \phi_{dec}(\theta) = 1]$ and T is "as large as possible". We call this structure the All Edit Structure (AES). The property of being as large as possible is as follows: if T_1 and T_2 are two TPOs satisfying edit constraints, then their union, in the graph merging sense, is also a TPO satisfying edit constraints. The union of T_1 and T_2 is defined as: (1) $Q_Y^{T_1 \cup T_2} = Q_Y^{T_1} \cup Q_Y^{T_2}$, $Q_Z^{T_1 \cup T_2} = Q_Z^{T_1} \cup Q_Z^{T_2}$, $Q_W^{T_1 \cup T_2} = Q_W^{T_1} \cup Q_W^{T_2}$; (2) $\forall y \in Q_Y^{T_1 \cup T_2}$, $\forall z, z' \in Q_Z^{T_1 \cup T_2}$, $\forall w \in Q_W^{T_1 \cup T_2}$, $\forall \theta, \theta' \in \Theta$, $\forall e_o \in E_o$, we have: $f_{yz}^{T_1 \cup T_2}(y, e_o) = z \Leftrightarrow \exists i \in \{1,2\} : f_{yz}^{T_i}(y, e_o) = z$, $f_{zz}^{T_1 \cup T_2}(z, \theta') = z' \Leftrightarrow \exists i \in \{1,2\} : f_{zz}^{T_i}(z, \theta') = z'$, $f_{zw}^{T_1 \cup T_2}(z, \theta) = w \Leftrightarrow \exists i \in \{1,2\} : f_{zw}^{T_i}(z, \theta) = w$ and $f_{wy}^{T_1 \cup T_2}(w, e_o) = y \Leftrightarrow \exists i \in \{1,2\} : f_{zw}^{T_i}(w, e_o) = y$.

Definition III.5.1 (All Edit Structure)**.** *Given system G, edit constraints ϕ_{dec} and ϕ_y, the All Edit Structure (AES) is the largest complete TPO:*

$$AES = (Q_Y^A, Q_Z^A, Q_W^A, E_o, E_o^r, \Theta, f_{yz}^A, f_{zz}^A, f_{zw}^A, f_{wy}^A, y_0)$$

where $\forall y \in Q_Y^A : \phi_y(y) = 1$ and $\forall \theta \in \Theta : \phi_{dec}(\theta) = 1$. The largest TPO is such that: for all TPO T satisfying the above two conditions, $T \sqsubseteq AES$.

Algorithm III.1: Construction of the AES

Input : $Obs(G)$, $Obs_d(G)$, E_o^r, ϕ_{dec}, ϕ_y
Output : AES

1 $Q_Y^A = \{y_0\} = \{(x_{obsd,0}, x_{obs,0})\}$, $Q_Z^A = \emptyset$, $Q_W^A = \emptyset$;
2 $AES_{pre} = DoDFS(y_0, \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r)$;
3 $AES = Prune(AES_{pre})$;

Algorithm III.2: DoDFS

Input : $y, \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r$

Output : AES_{pre}

for $e_o \in E_o$, *s.t.* $f_{yz}(y, e_o)!$ *by Definition III.4.1* **do**
 $z = ((x_{obsd}, x_{obsf}), e_o) = f_{yz}(y, e_o)$;
 add transition $y \xrightarrow{e_o} z$ to f_{yz}^A;
 if $z \notin Q_Z^A$ **then**
  $Q_Z^A = Q_Z^A \cup \{z\}$;
  $\Theta(z) = \emptyset$;
  $Z_{ext}(z) = \{z\}$;
  $EXTEND - Z(Z_{ext}(z), \phi_{dec})$;
  for $z' \in Z_{ext}(z)$ **do**
   if $\exists \theta \in \Theta$, *s.t.* $f_{zw}(z', \theta)!$ *by Definition III.4.1 and* $\phi_{dec}(\theta) = 1$ **then**
    $w = f_{zw}(z', \theta)$;
    $\Theta(z') = \Theta(z') \cup \{\theta\}$;
    add transition $z' \xrightarrow{\theta} w$ to f_{zw}^A;
    if $w \notin Q_W^A$ **then**
     $Q_W^A = Q_W^A \cup \{w\}$;
    for $e_o \in E_o$, *s.t.* $f_{wy}(w, e_o)!$ *by Definition III.4.1* **do**
     $y' = f_{wy}(w, e_o)$;
     if $\phi_y(y') = 1$ **then**
      add transition $w \xrightarrow{e_o} y'$ to f_{wy}^A ;
      if $y' \notin Q_Y^A$ **then**
       $Q_Y^A = Q_Y^A \cup \{y'\}$;
       $DoDFS(y', \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r)$;

Procedure: $EXTEND - Z(Z_{ext}(z), \phi_{dec})$

while $\exists z \in Z_{ext}(z)$, $\exists \theta \in \Theta$, *s.t.* $f_{zz}(z, \theta)!$ *by Definition III.4.1 and* $\phi_{dec}(\theta) = 1$ **do**
 $z' = f_{zz}(z, \theta)$;
 $\Theta(z) = \Theta(z) \cup \{\theta\}$;
 add transition $z \xrightarrow{\theta} z'$ to f_{zz}^A;
 if $z' \notin Q_Z^A$ **then**
  $Q_Z^A = Q_Z^A \cup \{z'\}$;
  $\Theta(z') = \emptyset$;
  $Z_{ext}(z) = Z_{ext}(z) \cup \{z'\}$;
  $EXTEND - Z(Z_{ext}(z), \phi_{dec})$;

Algorithm III.1 shows a general procedure for constructing the AES and it calls Algorithms III.2 and III.3 in its operation. In Algorithm III.2, we start searching from $y_0 = (x_{obsd,0}, x_{obs,0})$ and expand the state space recursively by computing all possible successors of the current state. We

Algorithm III.3: Prune

Input : A three-player observer

Output: A three-player observer without deadlocking states

while *there exist deadlocking W-states or Z-states* **do**
    for *deadlocking W-state w* **do**
        remove w from the structure
    for *deadlocking Z-state z* **do**
        if *there exist Y-state y and $e_o \in E_o$, s.t. z is reachable from y through e_o* **then**
            remove y and z from the structure;
        else
            remove z from the structure;
    take the accessible part of the structure;

terminate searching on a path when a Y-state violates the edit constraint, i.e., $\phi_y(y) = 0$ or an edit decision is not allowed by the constraints, i.e., $\phi_{dec}(\theta) = 0$. This is an iterative procedure, which allows us to build the whole reachable state space. We also add transitions in this process.

Specifically, at a newly added Z-state, we need to determine feasible edit decisions. There may be consecutive Z-states between a Y-state and a W-state. Then we search them in the procedure $EXTEND-Z$, which is also a depth-first search process. In $EXTEND-Z$, we add succeeding Z-states until no more f_{zz} transitions are defined and no more insertions are made. In this process, for each $z \in Q_Z^A$, we define $Z_{ext}(z)$ to be the set of Z-states that can be reached from z through f_{zz} transitions. We keep growing $Z_{ext}(z)$ until no more Z-states are added and no new f_{zz} transitions are defined at states in $Z_{ext}(z)$. Consecutive Z-states may form a cycle in the AES, which indicates that a loop is inserted by the edit function. Since the information state component of a Z-state comes from $2^X \times 2^X$ and its event component comes from E_o, both of which are finite sets, then only a finite number of Z-states are added in each iterate and $EXTEND-Z$ always terminates. Similarly, the information state components of Y-states and W-states also come from $2^X \times 2^X$, while the edit action components of W-states come from E_o or E_o^r. All of them are finite sets. Overall, only finite states will be added to AES_{pre} until some states or transitions violate the edit constraints. Thus, Algorithm III.2 terminates after a finite number of steps and returns a finite structure.

We denote the output of Algorithm III.2 by AES_{pre}, which may contain deadlocking states since edit constraints preclude transitions out of them or their succeeding states. We prune away deadlocking states as well as their predecessor states in Algorithm III.3 in an iterative manner until the structure converges. If a state is deadlocking, then the edit decisions leading to it should not be considered for synthesizing edit functions. Thus, we also prune away its preceding states. This process is similar to calculating the supremal controllable sublanguage in non-blocking supervisory control under full observation [23], by viewing the deadlocking states as undesired marked states and f^A_{yz}, f^A_{wy} transitions as uncontrollable while f^A_{zz}, f^A_{zw} transitions as controllable. Algorithm III.3 also terminates after a finite number of steps when no more states are to be removed, then it returns the AES after it is called in Algorithm III.1. The following theorem reveals the correctness and completeness of the AES, namely, the AES embeds all ND-privately safe edit functions satisfying the edit constraints.

Theorem III.5.1. *Given system G, a nondeterministic edit function f_{ne} is ND-privately safe if and only if $f_{ne} \in AES$.*

Proof. ($\Rightarrow$) By contradiction. Suppose $f_{ne} \vDash \varphi_{ndpri}$ but $f_{ne} \notin AES$. Then there should exist a TPO T such that $f_{ne} \in T$. This means that $\exists s \in P[\mathcal{L}(G)]$, $\exists r \in Run(T)$, s.t. $P_e(r) = s$, $l_g(r) \in f_{ne}(s)$ but $r \notin Run(AES)$. Thus, there are some states or transitions in r that are not in the AES. However, this implies that the union of T and the AES is strictly larger than the AES, which contradicts with the definition that the AES is the largest TPO satisfying the edit constraints.

($\Leftarrow$) Suppose that $f_{ne} \in AES$, then $\forall s \in P[\mathcal{L}(G)]$, $\forall \tilde{s} \in f_{ne}(s)$, $\exists r \in Run(T)$, s.t. $P_e(r) = s \wedge l_g(r) = \tilde{s}$. Since $\forall y = (x_d, x_f) \in r$, $x_d \in X_{obsd}$, we know $f_{ne}(s) \subseteq \mathcal{L}(Obs_d(G)) = L_{safe}$ and f_{ne} is privately safe. □

Remark III.5.1. *We briefly analyze the complexity of constructing the AES. First, we evaluate the complexity of Algorithm III.2. Here we define $Q^{ent}_Z = \{z \in Q^A_Z : \exists y \in Q^A_Y, \exists e_o \in E_o$ s.t. $f_{yz}(y, e_o) = z\}$ as the Z-states which can be reached from certain Y-states by f_{yz} transitions. Given system G with $|X|$ states, its observer $Obs(G)$ has at most $|X_{obs}| = 2^{|X|}$ states. Since $Q^A_Y \subseteq X_{obsd} \times X_{obs}$, $|Q^A_Y| \leq |X_{obs}|^2$. Also, each Y-state can execute at most $|E_o|$ observable events in line 1, so $|Q^{ent}_Z| \leq |E_o||X_{obs}|^2$.*

In $DoDFS$, we apply procedure $EXTEND-Z$ at each Q_Z^{ent} in line 8 to determine edit choices step by step. This procedure creates at most $(|X_{obs}|-1)$ *states for each Q_Z^{ent} state. Thus,* $|Q_Z^A| \leq |E_o||X_{obs}|^2(|X_{obs}|-1+1) = |E_o||X_{obs}|^3$. *Furthermore, every Z-state may lead to a W-state by f_{zw} transition, so* $|Q_W^A| \leq |E_o||X_{obs}|^3$. *Thus, the state space complexity of AES_{pre} is* $O(|X_{obs}|^3)$. *The complexity of Algorithm III.3 is quadratic in the size of AES_{pre} as one state is visited at most once in an iteration. Overall, the space complexity of constructing the AES is polynomial in terms of* $|X_{obs}|$.

Remark III.5.2. *It can be shown by induction on the length of strings that if the AES is not empty, then all edit functions embedded in it are admissible. This is a consequence of the pruning process in Algorithm 3 and we omit the proof here. By the same argument, no admissible edit function exists if the AES is empty. Hence, we will rule out this situation in the remainder of the chapter.*

Example III.5.1. *We show an* All Edit Structure. *The observer of system G is depicted in Figure V.2. All events* $\{a,b,c,d\}$ *are observable and observer state* 4 *is solely composed of secret states from G. The desired observer $Obs_d(G)$ is simply without state* 4 *and we omit its figure here. To begin with, we follow the first two steps of Algorithm III.1 and build AES_{pre} in Figure III.2, where squared states, oval states and diamond states stand for Y, Z and W states, respectively.*

The game is initialized at $y_0 = (0,0)$ *where the dummy player executes b and d since both events are defined at state* 0 *in $Obs(G)$. If b is executed, Z-state* $((0,0),b)$ *is reached, where the edit function plays and there are two edit decisions. At* $((0,0),b)$*, if the edit function chooses to erase b, then the system plays at W-state* $((0,0),b \to \epsilon)$*; if the edit function inserts d, then Z-state* $((1,0),b)$ *is reached since* $\delta_d(0,d) = 1$. *If a is also inserted after d is inserted, then another Z-state* $((2,0),b)$ *is reached. Then at Z-state* $((2,0),b)$*, if the edit function decides to stop inserting, W-state* $((2,0),b)$ *is reached. When the system plays, say, at* $((0,0),b \to \epsilon)$*, b occurs and leads to Y-state* $(0,4)$*, since b is not observed by the intruder and the first state component is not updated. When the system plays at* $((2,0),b)$*, b occurs and leads to Y-state* $(3,4)$ *since* $\delta_d(2,b) = 3$*,* $\delta(0,b) = 4$. *The whole structure is interpreted in a similar way.*

In this example, the edit constraints prohibit the edit function from erasing b at $((1,0),b)$ *and*

$((3,2),b)$, also $\phi_y((0,1)) = \phi_y((1,2)) = \phi_y((2,3)) = 0$. We use dashed lines in Figure III.2 to indicate the transitions and states that violate edit constraints. Those transitions/states are not in AES_{pre}. In Figure III.2, there are some deadlocking W-states such as $((0,0), d \to \epsilon)$, $((1,0), a \to \epsilon)$ and $((2,2), b \to \epsilon)$ and no deadlocking Z-states exist. Then we prune away those deadlocking states by Algorithm III.3 and finally obtain the AES in Figure III.3.

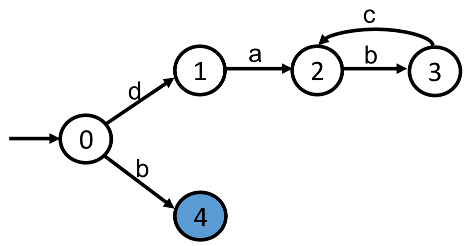

Figure III.1: The observer in Example III.5.1

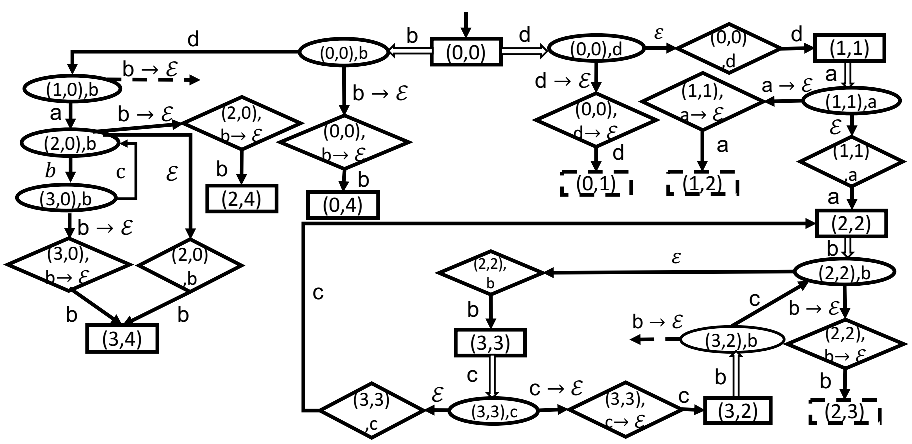

Figure III.2: AES_{pre} in Example III.5.1 (without dashed states and transitions)

Then it is natural to ask when there exists an ND-PP-enforcing edit function in the given AES. The key point is *every unsafe string shares the same edited behavior with some safe string*. However, the state information in the AES is insufficient to verify this condition as a Y-state may appear in multiple runs and different strings may be edited to the same one by different edit decisions. Therefore additional analysis is necessary, which is discussed in the next section.

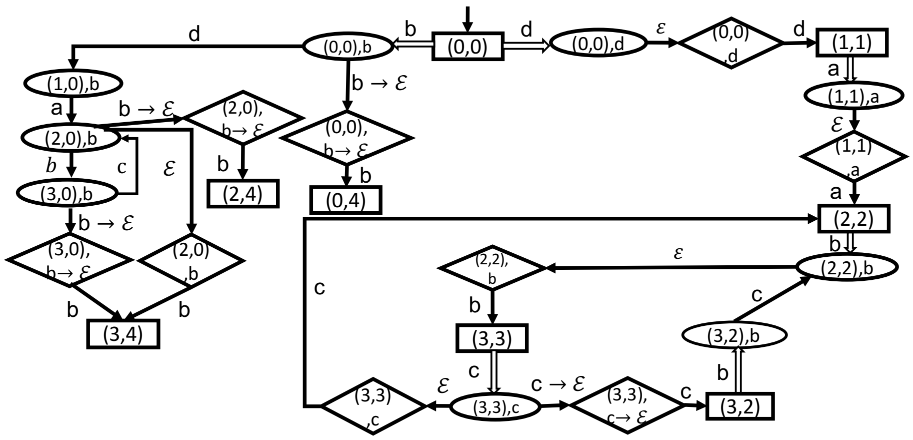

Figure III.3: The AES in Example III.5.1

III.6 Synthesis of Nondeterministic Privately Safe and Publicly Safe Edit Functions

In this section, we synthesize nondeterministic PP-enforcing edit functions. From Theorem III.5.1, any edit function embedded in the AES is ND-privately safe so we only need to consider ND-public safety. Unfortunately, we cannot only consider the state information in the AES for synthesis. Thus, we introduce the *reachability tree* of the AES, which is the "unfolded" AES with respect to unedited strings and edited strings. Then we have access to strings before/after edit and develop a synthesis algorithm based on the tree. The terminology of *reachability tree* is from the Petri net literature; it is employed here as it is well-suited to the construction procedure in this chapter.

III.6.1 Reachability Tree of the AES

The reachability tree of the AES is denoted by

$$AES_t = (Q_Y^{AT}, Q_Z^{AT}, Q_W^{AT}, E_o, E_o^r, \Theta, f_{yz}^{AT}, f_{zz}^{AT}, f_{zw}^{AT}, f_{wy}^{AT}, y_0)$$

and constructed in Algorithm III.4. It is built by unfolding the state space in a breadth-first search manner in line 2. The AES$_t$ is an acyclic structure by construction, so all its runs are finite. The transitions in the AES_t are defined in a similar way as in the AES. Within $DoDFS$, if an examined state is visited again, we stop searching on the current path and know there is a cycle in the AES. Since the number of states in the AES is finite, $DoBFS$ stops after a finite number of steps when all states in the AES are examined. In line 3, we call Algorithm III.3 and achieve two goals: (1) all leaf states in the AES_t are Y-states; (2) no deadlocking states exist in the AES_t. We denote by Q^{AT}_{Y-leaf} the leaf states in the AES_t. Since states are completely split in terms of state and string components, there is a unique run from the root y_0 to every state in the AES_t. Finally, we label each Y-state in the tree with both the edited string and the original string in line 6.

Algorithm III.4: Build labeled reachability tree of the AES

Input : AES
Output : AES_t
1 $Q^{AT}_Y = \{y_0\}$, $Q^{AT}_Z = Q^{AT}_W = \emptyset$;
2 $AES^{pre}_t = DoBFS(y_0, AES)$;
3 call Algorithm III.3, $Prune(AES^{pre}_t)$;
4 **for** *Y-state y in the remaining structure* **do**
5   specify the run r from y_0 to y in the remaining structure;
6   use $(l(r), P_e(r))$ to label y;
7 **return** AES_t;
Procedure: $DoBFS(q, AES)$
8 **while** *there exists state q in AES that has not been examined* **do**
9   evaluate all transitions defined at q in AES;
10   **if** *no transition is defined at q in AES* **then**
11   terminate searching on the current path from q;
12   **else**
13   **for** *a transition defined at q in AES* **do**
14   add state q' reached by the transition as a new state in the tree AES^{pre}_t;
15   **if** *q' equals a state on the path from y_0 to q* **then**
16   stop searching from on the current path q';

Edit functions embedded in the AES_t only make finite insertion choices. However, this does not compromise the performance of edit functions in opacity enforcement. We use Example III.5.1 to illustrate this point. If we build the reachability tree for this example, the cycle between Z-states $((3,0), b)$ and $((2,0), b)$ is broken and the transition c is removed. Thus, if we consider edit functions embedded in the AES_t, then string b can only be mapped to dab. However, all strings of the form $da(bc)^n b$ where $n \geq 1$ reach state 2. It does not really matter whether b is edited to a string containing a loop or not.

In the following discussion, we let the edit function make the same decisions every time a Z-state in the AES is reached. Hence, if there exists a cycle in the AES, the edit function does not change decisions whenever the cycle is visited. Therefore no information is lost if we consider edit functions embedded in the AES$_t$ and repeat the same edit decisions when two states share the same state components.

Remark III.6.1. *We briefly analyze the space complexity of the AES$_t$. First we have the notion* $Q = \max\{|Q_Y^{AT}|, |Q_Z^{AT}|, |Q_W^{AT}|\}$. *The number of nodes reached by the initial state in one step transition in the AES is at most Q. Also each node may have at most Q succeeding nodes by one step transition in the AES. Thus, the number of states reached by y_0 by two transitions is at most Q^2. The same process goes on and we know that there may be at most $|Q_Y^{AT}| + |Q_Z^{AT}| + |Q_W^{AT}|$ states between the root y_0 and any leaf state in the tree. Thus, the number of states in the AES$_t$ is at most in the order of $Q^{|Q_Y^{AT}|+|Q_Z^{AT}|+|Q_W^{AT}|+1}$. From last section's discussion, we know that the complexities of Q and $|Q_Y^{AT}| + |Q_Z^{AT}| + |Q_W^{AT}|$ are both of the order $(O(|X_{obs}|^3))$. Therefore, the complexity of the AES$_t$ does not exceed $O(|X_{obs}|^{3(|X_{obs}|^3+1)})$.*

In the AES$_t$, some Y-states are labeled by an unsafe string and a safe string while others by two safe strings. We partition Y-states as:

$$Q_Y^{AT1} = \{((x_d, x_f), (t, s)) \in Q_Y^{AT} : t \in L_{safe}, s \in L_{unsafe}\}$$
$$Q_Y^{AT2} = \{((x_d, x_f), (t, s)) \in Q_Y^{AT} : t, s \in L_{safe}\}$$

Next we define the *last preserved Q_Y^{AT2} state* as: $Q_{Y-lp}^{AT2} = \{y_t^2 \in Q_Y^{AT2} : \exists y_t^1 \in Q_Y^{AT1}, \exists \theta_1, \cdots \theta_m \in \Theta, \exists e_o \in E_o,$ s.t. $f_{wy}^{AT}(f_{zw}^{AT}(f_{zz}^{AT} \cdots (f_{zz}^{AT}(f_{yz}^{AT}(y_t^1, e_o), \theta_1), \cdots \theta_{m-1}), \theta_m), e_o) = y_t^2\}$, which serves as the "boundary" between Q_Y^{AT1} and Q_Y^{AT2} states.

Define $Q_{Y-leaf}^{AT1} = Q_{Y-leaf}^{AT} \cap Q_Y^{AT1}$ and $Q_{Y-leaf}^{AT2} = Q_{Y-leaf}^{AT} \cap Q_Y^{AT2}$ as leaf states that contain and do not contain unsafe string components. Besides, we define $Q_{Y-l}^{AT2} = Q_{Y-leaf}^{AT2} \cup Q_{Y-lp}^{AT2}$. Then we define

$$
\begin{aligned}
L_{leaf}^{u} &= \{l \in L_{unsafe} : \exists y_{leaf}^{1} = ((x_d, x_f), (t, s)) \in Q_{Y-leaf}^{AT1}, \text{ s.t. } s = l\} \\
L_{leaf}^{s} &= \{l \in L_{safe} : \exists y_{leaf}^{2} = ((x_d, x_f), (t, s)) \in Q_{Y-leaf}^{AT2}, \text{ s.t. } s = l\} \\
L_{lp}^{s} &= \{l \in L_{safe} : \exists y_{lp}^{2} = ((x_d, x_f), (t, s)) \in Q_{Y-lp}^{AT2}, \text{ s.t. } s = l\}
\end{aligned}
$$

as the set of unsafe strings appearing in Q_{Y-leaf}^{AT1}, the set of safe strings appearing in Q_{Y-leaf}^{AT2} and Q_{Y-lp}^{AT2}, respectively. We further group some Y-states by their components of original strings (safe or unsafe):

$$
\begin{aligned}
Q_{Y-leaf}^{AT1}(l) &= \{((x_d, x_f), (t, s)) \in Q_{Y-leaf}^{AT1} : s = l \in L_{leaf}^{u}\} \\
Q_{Y-leaf}^{AT2}(l) &= \{((x_d, x_f), (t, s)) \in Q_{Y-leaf}^{AT2} : s = l \in L_{leaf}^{s}\} \\
Q_{Y-lp}^{AT2}(l) &= \{((x_d, x_f), (t, s)) \in Q_{Y-lp}^{AT2} : s = l \in L_{lp}^{s}\} \\
Q_{Y-l}^{AT2}(l) &= \{((x_d, x_f), (t, s)) \in Q_{Y-l}^{AT2} : s = l \in L_{lp}^{s} \cup L_{leaf}^{s}\}
\end{aligned}
$$

In this chapter, we assume that events are inserted or erased one by one, so observed one at a time. Also both the observer's language and the safe language are prefix-closed. Therefore, if a string s is mapped to string l, then all the prefixes of s are mapped to some prefixes of string l. This result is formally stated as follows:

Lemma III.6.1. *Consider a nondeterministic edit function f_{ne}, if $s, t \in P[\mathcal{L}(G)]$ satisfy $f_e(s) \subseteq f_e(t)$, then $\forall s' \leq s$, $\exists t' \leq t$, s.t. $f_e(s') \subseteq f_e(t')$.*

This lemma has the implication that we can restrict attention to unsafe strings in L_{leaf}^{u} since all the other unsafe strings in the AES$_t$, being their prefixes, can be mapped to safe strings if strings in L_{leaf}^{u} can be mapped to safe strings. Besides, we can focus on safe strings in $L_{lp}^{s} \cup L_{leaf}^{s}$ for opacity enforcement as the other safe strings in the AES$_t$ are their prefixes. This result further justifies why we build the reachability tree AES$_t$: since the AES$_t$ explicitly contains unsafe strings in some of its leaf states, we can evaluate those leaf states and determine how those unsafe strings are edited.

III.6.2 Synthesis Algorithm

We proceed to synthesize nondeterministic PP-enforcing edit functions based on the AES$_t$. We will give a condition for verifying the existence of nondeterministic PP-enforcing edit functions and show that the verification problem is closely related with the synthesis problem. Then we will solve these two problems together. To begin with, we derive the following result from Theorem III.5.1, which shows that ND-private safety is always ensured by the AES.

Lemma III.6.2. *If the AES is not empty, then there exists a privately safe nondeterministic edit function.*

The ND-public safety case is more challenging and we start by evaluating the unsafe strings in the leaf states of the AES$_t$. For each unsafe string $l_i \in L^u_{leaf}$, we define the set of *PP-enforcing candidate states* as $S_{pp}(l_i) = \{((x_d, x_f), (t, l_i)) \in Q^{AT1}_{Y-leaf}(l_i) : \exists y^2 = ((x'_d, x'_f), (t', l')) \in Q^{AT2}_{Y-l}, \text{ s.t. } t \leq t'\}$. That is, we search through AES$_t$ to find $((x'_d, x'_f), (t', l'))$ where some prefix of the edited string t' is just t while the unedited unsafe string is also l_i. So if the edit function reaches those states, it will be publicly safe by definition. On the other hand, if $S_{pp}(l_i) = \emptyset$ for some l_i, then we know we can not find a safe string that shares the same edited behavior with unsafe string l_i, in which case no nondeterministic PP-enforcing edit function exists.

Besides, we call states in $Q^{AT1}_{Y-leaf}(l_i) \setminus S_{pp}(l_i)$ *bad candidate states* since the edited behaviors of l_i indicated in those states can not be matched with edited behaviors of any other safe string. Thus, if those states are reached by the edit function, ND-public safety can not be achieved. Those states are expected to be avoided when synthesizing nondeterministic PP-enforcing edit functions.

Based on those concepts, we propose Algorithm III.5 for synthesis. First we group the leaf states by their unsafe string components $l_i \in L^u_{leaf}$ in line 2. Each state in $Q^{AT1}_{Y-leaf}(l_i)$ corresponds to a potentially different edited behavior of l_i. Then we search through the AES$_t$ to find bad candidate states and remove them from the AES$_t$. As the removal of those states may bring in

Algorithm III.5: Synthesize PP-enforcing edit functions

Input : AES_t

Output: Nondeterministic PP-enforcing edit function

1 **for** $l_i \in L^u_{leaf}$ **do**

2 collect $Q^{AT1}_{Y-leaf}(l_i)$, suppose $Q^{AT1}_{Y-leaf}(l_i)$ has m_i elements;

3 **for** $j = 1 : m_i$ **do**

4 consider $y^1_j(l_i) = ((x_d, x_f), (t, l_i)) \in Q^{AT1}_{Y-leaf}(l_i)$;

5 **if** $\nexists y^2(l') = ((x'_d, x'_f), (t', l')) \in Q^{AT2}_{Y-l}$, *s.t.* $t \leq t'$ **then**

6 remove $y^1_j(l_i)$ from the AES$_t$

7 $AES^r_t = Prune(AES_t)$;

8 **for** $l_i \in L^u_{leaf}$ **do**

9 denote by Q^{AT1}_{Y-re} (Q^{AT2}_{Y-re}) the Y-states in AES^r_t with (without) unsafe string components, then define $S^r_{pp}(l_i) = \{((x_d, x_f), (t, l_i)) \in Q^{AT1}_{Y-leaf}(l_i) \cap Q^{AT1}_{Y-re} : \exists y^2 = ((x'_d, x'_f), (t', l')) \in Q^{AT2}_{Y-l} \cap Q^{AT2}_{Y-re}$, s.t. $t \leq t'\}$;

10 **if** $S^r_{pp}(l_i) = \emptyset$ **then**

11 nondeterministic PP-enforcing edit functions do not exist, terminate the algorithm;

12 return the nondeterministic edit function embedded in AES^r_t;

deadlocking states, we apply Algorithm III.3 to resolve deadlocking states in line 7 and denote the remaining structure by AES^r_t. In this process, some states in $S_{pp}(l_i)$ may also be removed. We use Q^{AT1}_{Y-re} and Q^{AT2}_{Y-re} to denote the Y-states with and without unsafe string components in the AES^r_t, respectively. For unsafe string l_i, we define $S^r_{pp}(l_i)$ in line 9 as the set of PP-enforcing candidate states remaining in the AES^r_t after pruning. We claim that if $S^r_{pp}(l_i)$ is not empty for each l_i, then there exist nondeterministic PP-enforcing edit functions in the AES$_t$. Finally we may extract the edit function by following transitions in the AES^r_t

Theorem III.6.1. *Given the AES^r_t, nondeterministic PP-enforcing edit functions exist if and only if* $\forall l_i \in L^u_{leaf}$, $S^r_{pp}(l_i) \neq \emptyset$.

Proof. ($\Rightarrow$) By contradiction. Suppose $\exists f_{ne} \in AES^r_t$, $f_{ne} \vDash \varphi_{ndpp}$ and $\exists l_i \in L^u_{leaf}$, s.t. $S^r_{pp}(l_i) = \emptyset$. Then we can find $s \in f_{ne}(l_i)$, s.t. $\nexists t \in L_{safe}$ and $s \in f_{ne}(t)$, which contradicts $f_{ne} \vDash \varphi_{ndpp}$.

($\Leftarrow$) Given the AES$_t$ and the AES^r_t, it is sufficient to consider unsafe strings in L^u_{leaf} and safe strings in $L^s_{leaf} \cup L^s_{lp}$ for synthesis. Besides, we only need to check ND-public safety since the AES is not empty. If $\forall l_i \in L^u_{leaf}$, $S^r_{pp}(l_i) \neq \emptyset$, we know $\forall y^1(l_i) = ((x_d, x_f), (t, l_i)) \in S^r_{pp}(l_i)$, $\exists y^2 =$

$((x'_d, x'_f), (t', l')) \in Q^{AT2}_{Y-l} \cap Q^{AT2}_{Y-re}$, s.t. $t \leq t'$. Since $f_{ne} \in AES^r_t$, $f_{ne} \in AES$ also holds. We let all the players make the same decisions specified at states in the AES^r_t whenever a state is reached again in the AES. So we can design an edit function f_{ne} such that $f_{ne}(l_i) = \{t : \exists y^1(l_i) = ((x_d, x_f), (t, l_i)) \in S^r_{pp}(l_i)\}$ and $t' \in f_{ne}(l')$. Since $t \leq t'$, we know $f_{ne}(l_i) \subseteq L_{safe}$, $\forall l_i \in L^u_{leaf}$. Therefore, f_{ne} is both privately safe and publicly safe. □

Theorem III.6.1 gives a necessary and sufficient condition for verifying the existence of nondeterministic PP-enforcing edit functions. It also shows the completeness and soundness of Algorithm III.5, so the synthesis of nondeterministic PP-enforcing edit functions is reduced to finding $S^r_{pp}(l_i)$ for every $l_i \in L^u_{leaf}$ in the AES^r_t. When running Algorithm III.5, we collect all edited strings appearing in states from $S^r_{pp}(l_i)$ and include them as the potential edited behavior of $l_i \in L^u_{leaf}$. In that way, the synthesized nondeterministic edit function is "most permissive" in the sense that it preserves all feasible edit decisions to achieve ND-private safety and ND-public safety.

Remark III.6.2. *Compared with deterministic edit functions, nondeterministic edit functions perform better at enforcing public safety. The intuition is as follows. Consider the case when a safe string is edited to multiple (safe) strings which may be the edited behaviors of several unsafe strings. In the deterministic case, every string is mapped to a* unique *one so in the above case, we are only able to guarantee that one unsafe string shares the same edited behavior with a safe string, hence, public safety is violated. Thus, a deterministic PP-enforcing edit function may not always exist. However, if nondeterminism is allowed, as long as we find an edited string whose edited behaviors correspond to the edited behaviors of (potentially multiple) unsafe strings, then nondeterministic public safety is satisfied. The above argument further justifies why we explore nondeterministic edit functions, given that both deterministic and nondeterministic edit functions may enforce private safety.*

Example III.6.1. *Let the observer in Figure III.4 be with $E_o = \{a, b, c, d\}$, states 7 and 8 are composed of only secret states from the system. We omit the steps of building the AES and the AES$_t$, instead we directly show the AES$_t$ in Figure III.5. While we only label leaf states with strings here* *and Q^{AT1}_{Y-leaf} states are marked in red (those states contain an unsafe string label). Due to the edit constraints (not explicitly stated here), the edit function can only make decisions and reach states as indicated in the AES_t. We can see that $((6,8),(ab,b))$ shares the first string component with $((6,4),(ab,dabc))$, $((4,7),(dabc,abc))$ shares the first string component with $((4,4),(dabc,dabc))$. Also unsafe string b is edited to ab, unsafe string abc is edited to dabc, safe string dabc is edited to dabc or ab.*

It is interesting to notice that if we let the edit function be deterministic, i.e., every string is mapped to a unique one, then no PP-enforcing edit functions exist here since unsafe strings b and abc can not share the same modified behavior with safe string dabc simultaneously. However, a nondeterministic PP-enforcing edit function exists by Algorithm III.5. No states are removed from the AES_t and we have $S^r_{pp}(b) = \{((6,4),(ab,dabc))\}$, $S^r_{pp}(abc) = \{((4,4),(dabc,dabc))\}$. So the edit function inserts a before event b occurs from state 0; inserts d before event a occurs from state 0; inserts nothing before event d occurs from state 0 or just erases that d. This example reveals that introducing nondeterminism to edit functions may contribute to opacity enforcement by allowing more plausible denial for the intruder's inference, compared with the deterministic counterpart.

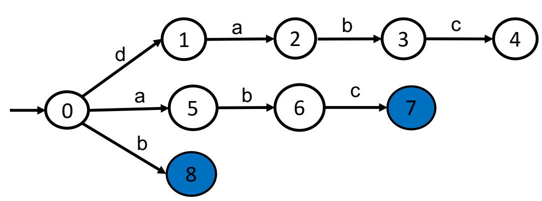

Figure III.4: The observer in Example III.6.1

III.7 Conclusion

We discussed opacity enforcement by edit functions in nondeterministic settings. Based on the knowledge of the adversary, we defined private safety and public safety of nondeterministic edit functions and then investigated their enforcement. This chapter is the first to apply nondeterministic edit functions to enforce opacity. The concept of edit constraint was introduced to restrict the

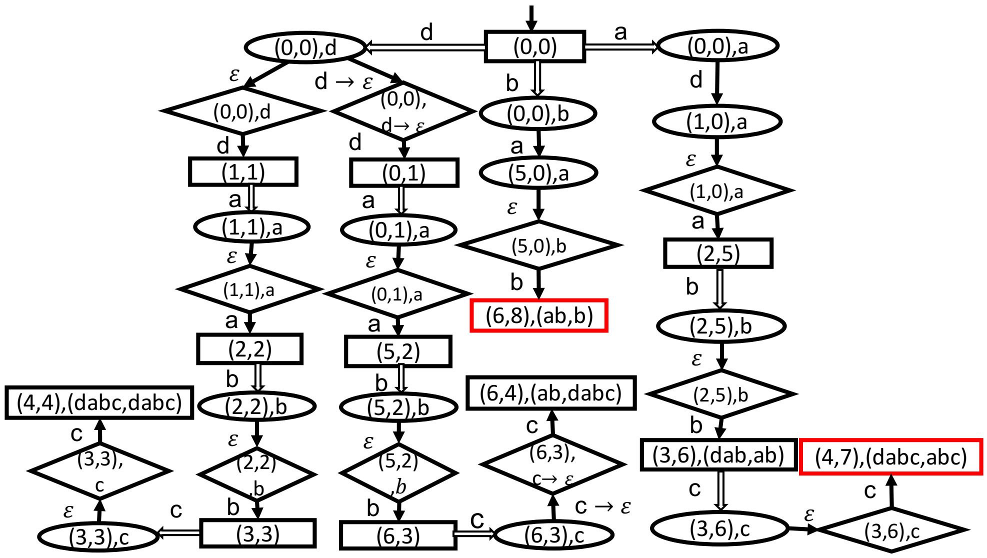

Figure III.5: The AES_t in Example III.6.1

choices of edit functions. Then we reformulated the problem as a three-player game and proposed the All Edit Structure (AES), which embedded all privately safe edit functions satisfying edit constraints. Finally, an algorithm was presented for synthesizing nondeterministic PP-enforcing edit functions based on the reachability tree of the AES.

CHAPTER IV

Enforcing Opacity by Insertion Functions under Multiple Energy Constraints

IV.1 Introduction

In this chapter, we formulate the problem of opacity enforcement by insertion functions under *multiple quantitative constraints*. Notice that here the insertion function only has partial observation of the system, i.e., it is only aware of the occurrence of observable events. The insertion functions are should enforce opacity while ensure that each type of resource of the system never drops below zero in the enforcement process, for all possible system behaviors (worst-case analysis). Then we transfer this problem to a two-player game between the insertion function and the environment, then solve it by constructing a discrete game structure called Energy Insertion Structure (EIS). The insertion function plays by inserting events, which consumes resources, while the system plays by executing events, which consumes or gains resources. Therefore,the system's resource levels dynamically change. EIS includes winning strategies of the insertion function under both qualitative and quantitative requirements.

Among the insertion strategies obtained from EIS, we are particularly interested in those that work in an "economical" way. In other words, there exists a upper bound for the rate of insertion cost so that only a reasonable amount of resource is consumed per step of insertion. Then we slightly modify EIS and formulate the bounded insertion cost rate problem as a multidimensional mean payoff game. This problem is solved by leveraging a novel approach called *hyperplane separation technique* proposed in [34].

Our work in this chapter is inspired by some recent works on quantitative two-player games in theoretical computer science, specially, energy game and mean payoff game. Those two games are closely related and thoroughly discussed in the literature; see, e.g., [4, 43]. In some cases, one player only has imperfect information about the game and thus is not informed of some moves of its opponent. Under imperfect information, energy games are decidable and known to be Ackermann-complete [87] with fixed amount of initial energy, while mean payoff games are in general undecidable [40]. Another generalization is multidimensional game [33], where both players have several quantitative objectives. The above works also inspired the work [90], which studies supervisory control for DES using energy games with partial observation. We adapt some methodology from [90] to the different problem of opacity enforcement by obfuscation. To the best of our knowledge, this chapter is the first to investigate opacity enforcement under multiple quantitative objectives.

This chapter is organized as follows. Section IV.2 describes our system model. Section IV.3 formulates the energy constrained opacity enforcement problem. Section IV.4 introduces the Energy Insertion Structure (EIS). Section IV.5 applies EIS to solve the energy constrained opacity enforcement problem. Section IV.6 formulates the bounded cost rate insertion strategy synthesis problem and solves it by the hyperplane separation technique. Finally, Section IV.7 concludes the chapter.

IV.2 System Model

We consider opacity and its enforcement in a quantitative DES modeled as a weighted finite-state automaton:

$$G = (X, E, f, x_0, \omega)$$

where X is the finite set of states, E is the finite set of events, $f : X \times E \to X$ is the partial state transition function, and $x_0 \in X$ is the unique initial state. We denote by $X_S \subset X$ the set of *secret* states that should remain opaque. The transition function is extended to domain $X \times E^*$ in the standard manner [23] and we still denote it by f. The language generated by G is defined as $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined". We write $s \leq u$ if string s is a prefix of string u; also $s < u$ if $s \leq u$ and $s \neq u$. We also denote by $t \in s$ if string t is a substring of s. The multidimensional function $\omega : E \to \mathbb{Z}^k$ assigns a k-dimensional weight vector to each event in E where k is a (fixed) positive integer and each entry reflects the gain or cost of a certain type of resource associated with the occurrence of an event. We denote by $\omega^{(i)}(e)$ the i-th component of $\omega(e)$ for $e \in E$. In this work, we let $\overrightarrow{0}$ be the k-dimensional vector of all 0s. The function ω is additive, whose domain is extended to E^* by letting $\omega(\epsilon) = \overrightarrow{0}$, $\omega(se) = \omega(s) + \omega(e)$ where $s \in E^*$, $e \in E$.

Given an automaton G, for $x_1, x_2 \in X$ and $e \in E$, we denote by $x_1 \xrightarrow{e} x_2$ if $f(x_1, e) = x_2$. A *run* in G is a sequence of alternating states and events: $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$ and it may be infinitely long. We denote the set of runs in G by $Run(G)$ and $x \in r$ if x is a state in r. We call a run *initial* if its initial state is the initial state of the system. Besides, a run forms a *cycle* if $x_1 = x_n$ and a cycle is *simple* if $\forall i, j \in \{1, 2, \cdots n-1\}$, $i \neq j \Rightarrow x_i \neq x_j$. If r is a cyclic run, there is a *corresponding loop* $e_1 e_2 \cdots e_{n-1}$ starting from and ending in x_1. We further call the loop *simple* if the cycle is simple.

We refer to the set of quantitative resources associated with the operation of the system as *energy*. The system is granted with initial energy vector $v_0 \in \mathbb{N}^k$ to support its operation. Given $s = e_0 e_1 \cdots e_{n-1} \in \mathcal{L}(G)$, the *energy level of the system after* s is $V(s) = v_0 + \sum_{i=0}^{n-1} \omega(e_i)$. We also denote by $V^{(i)}(s)$ the i-th component of the k-dimensional vector $V(s)$. Then we make the following important assumption that the energy level vector should always be nonnegative in every dimension and we will explain it in the next section.

Assumption IV.2.1. $\forall s \in \mathcal{L}(G)$, $V(s) \geq \overrightarrow{0}$.

System G is partially observable, i.e., $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events. Given $t = t'e \in E^*$, its natural projection under $P : E^* \to E_o^*$

is recursively defined as $P(t) = P(t')P(e)$ where $t' \in E^*$ and $e \in E$. The projection of an event is $P(e) = e$ if $e \in E_o$ and $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string.

Given a set of states $q \subseteq X$, the *unobservable reach*, denoted by $UR(q)$, is defined as: $UR(q) = \{x' \in X : \exists x \in q, \exists s \in E_{uo}^*, \text{ s.t. } f(x,s) = x'\}$. Besides, the *observable reach* under observable event e_o, denoted by $Next_{e_o}(q)$, is defined as: $Next_{e_o}(q) = \{x' \in X : \exists x \in q, e_o \in E_o, \text{ s.t. } f(x,e_o) = x'\}$. Then the *observer* of G is: $Obs(G) = (X_{obs}, E_o, \delta, x_{obs,0}, \omega_{obs})$ where $X_{obs} \subseteq 2^X$ is the state space; $\delta : X_{obs} \times E_o \to X_{obs}$ is the transition function and $\forall x_{obs} \in X_{obs}$, $\delta(x_{obs}, e_o) = UR(Next_{e_o}(x_{obs}))$; $x_{obs,0} = UR(x_0)$ is the initial state; $\omega_{obs} : E_o \to \mathbb{Z}^k$ is the same as ω over the restricted domain E_o. An observer state can be viewed as a *current state estimate (or state estimate in short)* of the system, which is a subset of X.

IV.3 Problem Formulation

In this section, we first review the notion of *current-state opacity* (Definition II.3.1) and the mechanism of insertion functions. Then we formulate the *energy constrained opacity enforcement problem*.

A system is current-state opaque if for every string reaching a secret state, there exists another string reaching a non-secret state which shares the same projection, thereby providing deniability of the secret. CSO can be verified by building the observer and checking whether an observer state contains solely secret states. Based on CSO, we define the *safe language*, which is the prefix-closure of the projected non-secret strings: $L_{safe} = P[\mathcal{L}(G)] \setminus \{[P[\mathcal{L}(G)] \setminus P(L_{NS})] E_o^*\}$. We also define the *unsafe language* $L_{unsafe} = P[\mathcal{L}(G)] \setminus L_{safe}$.

Given system G and its observer $Obs(G)$, the *desired observer* $Obs_d(G) = (X_d, E_o, \delta_d, x_{d,0})$ is obtained by removing all observer states composed of only secret states and then taking the accessible part, see [119]. Here $X_d \subseteq X_{obs}$ is the state space, E_o is the set observable events, $\delta_d : X_d \times E_o \to X_d$ is the same transition function as δ with restricted domain $X_d \times E_o$, $x_{d,0}$ is the initial state and we omit the weight function in $Obs_d(G)$. It is easy to see that $Obs_d(G)$ generates exactly L_{safe}.

Opacity may not always hold and an *insertion function* may be used to enforce opacity. The insertion function is an interface between the output of the system and the external environment including the intruder. It may insert fictitious events into the output stream of the system to obfuscate the intruder; see [54, 119] for more details of this concept.

Definition IV.3.1 (Insertion Function)**.** *An insertion function is defined as: $f_i : E_o^* \times E_o \to E_o^* E_o$ such that for $l \in E_o^*$ and $e_o \in E_o$, $f_i(l, e_o) = s_I e_o$ where $s_I \in E_o^*$.*

By definition, the insertion function inserts s_I before the next observable event e_o given that l has been observed, then it outputs $s_I e_o$ to the outside environment. Besides, s_I may be ϵ when nothing is inserted. We also define a string-based version of f_i and with a slight abuse of notation, denote it by f_i as well (it will be clear from the argument which form of f_i is being considered): $f_i(\epsilon, \epsilon) = \epsilon$ and $f_i(\epsilon, le_o) = f_i(\epsilon, l) f_i(l, e_o)$.

An insertion function inserts strings based on the observable behavior of the system. However, unobservable events do occur between two observable events. As a convention, when we need to discuss unprojected strings with insertion, we assume without loss of generality that the inserted string is placed right before the next observable event in an unprojected string.

Convention IV.3.1. *Given $s = \xi_0 e_0 \cdots \xi_{n-1} e_{n-1} \xi_n \in \mathcal{L}(G)$ where $\forall j \le n$, $\xi_j \in E_{uo}^*$ and $e_j \in E_o$, if $f_i(e_0 e_1 \cdots e_{j-1}, e_j) = \theta_j e_j$ where $\forall j \le n$, $\theta_j \in E_o^*$, then s is mapped to $s' = \xi_0 \theta_0 e_0 \cdots \xi_j \theta_i e_j \cdots \xi_n \theta_n e_n$ where $P(s') \in P[\mathcal{L}(G)]$.*

It is possible that $s' \notin \mathcal{L}(G)$, but what matters is that $P(s') \in P[\mathcal{L}(G)]$, since the intruder only observes strings in $P[\mathcal{L}(G)]$ for its inference of secrets.

An insertion function f_i may be encoded as an input/output (I/O) automaton $IA = (X_{ia}, E_o, E_o^+, \delta_{ia}, f_{ia}, x_{ia,0})$. Here X_{ia} is the state space; E_o is the set of input events; $E_o^+ = E_o^* E_o$ is the set of output strings; $\delta_{ia} : X_{ia} \times E_o \to X_{ia}$ is the transition function; $f_{ia} : X_{ia} \times E_o \to E_o^+$ is the output function such that $f_{ia}(x_{ia}, e_o) = s_I e_o$ where $\delta_{ia}(x_{ia}, e_o)!$ and $\delta_{ia}(x_{ia,0}, s) = x_{ia}$, if $f_I(s, e_o) = s_I e_o$; $x_{ia,0} \in X_{ia}$ is the initial state.

Next, we present the notion of *private safety* from [119], which indicates that every string in $P[\mathcal{L}(G)]$ is mapped to a safe string under certain insertion choices.

Definition IV.3.2 (Private Safety)**.** *Given system G with projection P and safe language L_{safe}, insertion function f_i is privately safe if $\forall s \in P[\mathcal{L}(G)]$, $f_i(s) \in L_{safe}$.*

We assume that event insertion always costs energy and define the *insertion weight function* $\omega_{in} : E_o \to (\mathbb{Z} \setminus \mathbb{N}^+)^k$, which assigns a k-dimensional weight vector to each inserted event, where all components are non positive. Function ω_{in} is additive and its domain is extended to E_o^* by letting $\omega_{in}(\epsilon) = \overrightarrow{0}$ and $\omega_{in}(se_o) = \omega_{in}(s) + \omega_{in}(e_o)$ for $s \in E_o^*$, $e_o \in E_o$. Equivalently, we may use $-\omega_{in}$ to stand for insertion costs. Without loss of generality, we assume that $\omega_{in}(e_o) \neq \overrightarrow{0}$ for all $e_o \in E_o$, i.e., insertion of an observable event always costs energy. The i-th component of $\omega_{in}(e_o)$ for $e_o \in E_o$ is denoted by $\omega_{in}^{(i)}(e_o)$.

Next, we define the *system's energy level after insertion* as $V_m : \mathcal{L}(G) \times E^* \to \mathbb{Z}^k$. Given $s = \xi_0 e_0 \xi_1 e_1 \cdots \xi_{n-1} e_{n-1} \xi_n \in \mathcal{L}(G)$ where $\forall j \leq n$, $\xi_j \in E_{uo}^*$ and $e_j \in E_o$, suppose s is mapped to $s' = \xi_0 \theta_0 e_0 \xi_1 \theta_1 e_1 \cdots \xi_{n-1} \theta_{n-1} e_{n-1} \xi_n$ by Convention IV.3.1 by some insertion function; then we let $V_m(s, s') = V(s) + \sum_{j=0}^{n-1} \omega_{in}(\theta_j)$. We will denote s' by s_{f_i} if s is mapped to s' by f_i. Hence, $V_m(s, s_{f_i})$ is the energy level of the system after string s is modified by insertion function f_i.

Given a non-opaque system G with initial energy vector v_0, we aim to design an insertion function f_i which enforces opacity but never forces the system's energy level to drop below zero in the component-wise sense. That is, the insertion function is constrained by the energy level of the system, i.e., $\forall s \in P[\mathcal{L}(G)]$, $V_m(s, s_{f_i}) \geq \overrightarrow{0}$. Since insertion always costs energy, we made Assumption IV.2.1 earlier to ensure some energy margins for the insertion function. We now formally formulate the energy constrained opacity enforcement problem.

Problem IV.3.1. *Given system G with initial energy vector v_0, the energy constrained opacity enforcement problem is to find an insertion function f_i such that:* (1) *f_i is privately safe;* (2) *$\forall s \in \mathcal{L}(G)$, $V_m(s, s_{f_i}) \geq \overrightarrow{0}$.*

Due to partial observation of the system, we need to estimate both current states and energy levels of the system so that insertion functions may make proper decisions to enforce opacity. This issue will be discussed in the following sections. Also notice that if there exists an insertion function solving Problem IV.3.1 with initial energy vector v_0, then the same insertion function also solves the problem with any initial energy vector $v'_0 \geq v_0$. We will see later that this simple monotonicity property allows us to define a finite structure to embed solutions to Problem IV.3.1.

IV.4 Energy Insertion Structure

In this section we define *energy information states* and a bipartite game structure *Energy Insertion Structure (EIS)*. By introducing these concepts, we transform Problem IV.3.1 into a reachability game with perfect information between the insertion functions and environment. Then we solve Problem IV.3.1 on EIS.

IV.4.1 Building the Verifier

We first review the concept of *verifier* proposed in [54]. It serves as an intermediate structure for constructing EIS here and encodes potentially feasible insertion choices for opacity enforcement without considering the energy constraints.

Given system G, in order to build the verifier, we first introduce the *feasible observer* [54]. The feasible observer is obtained by adding self-loops for all observable events at each state in observer $Obs(G)$. Formally, it is defined as $Obs_f(G) = (X_f, E_o, \delta, \delta_{sl}, x_0^f)$ where $X_f = X_{obs}$ is the state space; E_o is the set of observable events; δ is the same transition function as in the observer; $\delta_{sl} : X_f \times E_o \rightarrow X_f$ is the self-loop transition function such that $\forall x^f \in X_f$, $\forall e_o \in E_o$, $\delta_{sl}(x^f, e_o) = x^f$; $x_0^f = x_{obs,0}$ is the initial state. Thus at a state x^f, there may be two transitions labeled by e_o defined: (i) the normal transition δ representing the occurrence of an observable event and (ii) transition δ_{sl} representing potential event insertion.

Then we synchronize desired observer $Obs_d(G)$ and feasible observer $Obs_f(G)$ by the *verifier parallel composition* [54] to obtain the *verifier*, defined as $G_v = (X_v, E_o, \delta_{vd}, \delta_{vs}, x_{v0})$. Here

$X_v \subseteq X_d \times X_f$ is the state space, E_o is the set of observable events; $\delta_{vs} : X_v \times E_o \to X_v$ is the transition function corresponding to normal transitions in both $Obs_d(G)$ and $Obs_f(G)$; $\delta_{vd} : X_v \times E_o \to X_v$ is the transition function corresponding to normal transitions in $Obs_d(G)$ and added self-loop transitions in $Obs_f(G)$; $x_{v0} = (x_{obs,0}, x_{obs,0})$ is the initial state. A state $x_v = (x^d, x^f) \in X_v$ has two components: the left one is the intruder's estimate and the right one is the (true) system's estimate. They are usually different as insertion functions obfuscate the intruder by manipulating its observation.

Definition IV.4.1 (Verifier parallel composition)**.** *The verifier parallel composition $\|_v$ is a special parallel composition between $Obs_d(G)$ and $Obs_f(G)$: $G_v = Obs_d(G)\|_v Obs_f(G)$ where transition functions δ_{vs} and δ_{vd} are defined for synchronization: $\delta_{vs}((x^d, x^f), e) := (\delta_d(x^d, e), \delta(x^f, e))$ and $\delta_{vd}((x^d, x^f), e) := (\delta_d(x^d, e), \delta_{sl}(x^f, e)) = (\delta_d(x^d, e), x^f)$.*

The transition function δ_{vs} captures actual event occurrences, thus both the intruder's and the system's estimates change with such transitions; while δ_{vd} captures event insertions, thus only the intruder's estimate is updated. This is consistent with the mechanism of the insertion function, which is an interface between the output of the system and the outside environment. It only changes the intruder's observations but not the system's behavior. Here $x^d \in X_d$ and $x^d \notin 2^{X_S}$ by definition, so what the intruder observes does not reveal the system's secrets. For completeness, we define $\delta_{vd}(x_v, \epsilon) = x_v$ for all $x_v \in X_v$.

IV.4.2 Energy Information States

We aim to synthesize an insertion function which enforces opacity and maintains nonnegative energy level in all dimensions. To achieve these goals, we integrate the information of state estimates and energy levels into properly defined *Energy Information States*. Here we let $|\cdot|$ be the cardinality of a set.

Definition IV.4.2 (Energy Information State)**.** *Given system G, an energy information state is:*

$$q^e = ((x^d, x^f), [v(1), \cdots v(|x^f|)]) \in X_v \times \mathbb{Z}^{k \times |X|}$$

Let $Est_s(q^e)$ and $Lev_e(q^e)$ denote the state estimate and energy level components, respectively; hence, $q^e = (Est_s(q^e), Lev_e(q^e))$.

We denote by Q^E the set of energy information states, which track the system's estimate x^d , the intruder's estimate x^f and the energy levels of the system at each state in x^f. Besides, each $q^e \in Q^E$ induces a *belief function* $h_{q^e} : Est_s(q^e) \to \mathbb{Z}^k$. Specifically, for $q^e \in Q^E$ where $Est_s(q^e) = (x^d, x^f) \in X_v$, we have $Lev_e(q^e) = \{h_{q^e}(x) : x \in x^f\}$. We usually put $Lev_e(q^e)$ in a column vector's form: $[h_{q^e}(x_1), \cdots h_{q^e}(x_{|x^f|})]$. By convention, elements in $Lev_e(q^e)$ are placed in an increasing order w.r.t. state names in x^f. Our definition is inspired by the belief function in [40] and the observation function in [90]. In the following discussion, we use $h^{(i)}_{q^e}(x)$ to denote the i-th element in $h_{q^e}(x)$.

To compare energy level vectors, we extend the measure $\leq$ from scalars to vectors as follows: given two vectors $v_1 = [v_1(1), v_1(2), \cdots, v_1(k)]$, $v_2 = [v_2(1), v_2(2), \cdots, v_2(k)] \in \mathbb{Z}^k$, we denote by $v_1 \leq v_2$ (respectively $v_1 \geq v_2$) if $\forall 1 \leq i \leq k, v_1(i) \leq v_2(i)$ (respectively $v_1(i) \geq v_2(i)$). Then we further extend it to a measure on matrices: given two matrices $m_1 = [v_1, v_2, \cdots, v_n], m_2 = [v'_1, v'_2, \cdots, v'_n] \in \mathbb{Z}^{k \times n}$, we denote by $m_1 \leq m_2$ if $v_i \leq v'_i$ for all $1 \leq i \leq n$.

An energy information state $q^e \in Q^E$ is *energy safe* (or simply *safe*) if $\forall x \in x^f$ where $Est_s(q^e) = (x^d, x^f)$, $h_{q^e}(x) \geq \overrightarrow{0}$. We define an order $\preccurlyeq$ over the set of energy information states: for $q^e_1, q^e_2 \in Q^E$, $q^e_1 \preccurlyeq q^e_2$ if $Est_e s(q^e_1) = Est_s(q^e_2)$ and $Lev_e(q^e_1) \leq Lev_e(q^e_2)$. We also say that q^e_2 *subsumes* q^e_1 if $q^e_1 \preccurlyeq q^e_2$, i.e., q^e_1 and q^e_2 share the same verifier state component but the energy level vector of q^e_2 is no less than that of q^e_1 at every possible current state in $Est_s(q^e_2)$. By Dickson's lemma (see [69]), the order $\leq$ on $\mathbb{N}^m$ is a *well-quasi-ordering* for any $m \in \mathbb{N}$. Besides, the Cartesian product of two well-quasi-ordered sets $S \subseteq \mathbb{N}^m$ and $T \subseteq \mathbb{N}^m$ by using $\leq$ is also a well-quasi ordered set [80], i.e., $(s,t) \leq (s',t') \Leftrightarrow [s \leq s'] \wedge [t \leq t']$ for $s, s' \in S$, $t, t' \in T$. Thus we can further argue that $\preccurlyeq$ on safe energy information states is also a well-quasi ordering, i.e., for any infinite sequence of states $q^e_1, q^e_2 \cdots \in Q^E$, $\exists i, j \in \mathbb{N}$, s.t. $i < j$ and $q^e_i \preccurlyeq q^e_j$.

We call $q^{ae} \in Q^E \times E_o$ an *augmented energy information state*, i.e., q^{ae} is an energy information state augmented with an observable event. Let $I_E(q^{ae})$, $E(q^{ae})$ denote the energy information state and observable event components of q^{ae}, respectively. So we have $q^{ae} = (I_E(q^{ae}), E(q^{ae}))$. With a slight abuse of notation, we use $h_{q^{ae}}$ to stand for h_{q^e} where $q^e = I_E(q^{ae})$. Besides, q^{ae} is *(energy) safe* if $\forall x \in x^f$ where $Est_s(I_E(q^{ae})) = (x^d, x^f)$, $h_{q^{ae}}(x) \geq \overrightarrow{0}$. Then we define the following two concepts to characterize the update of energy and augmented energy information states with event insertion and execution.

For $e_o \in E_o$, we say that $q^{ae} \in Q^E \times E_o$ is an *e_o-execution successor* of $q^e \in Q^E$ if $I_E(q^{ae}) = q^e$ and $q^{ae} = (q^e, e_o)$. In other words, we simply combine an energy information state q^e with an observable event e_o to create an augmented energy information state q^{ae}.

For $\theta \in E_o^*$, $e_o \in E_o$, we say $q^e \in Q^E$ is a *(θ, e_o)-insertion successor* of $q^{ae} = (I_E(q^{ae}), e_o) \in Q^E \times E_o$ if: (i) $Est_s(q^e) = (x'^d, x'^f) = \delta_{vs}(\delta_{vd}((x^d, x^f), \theta), e_o)$ where $Est_s(I_E(q^{ae})) = (x^d, x^f)$; (ii) $\forall x' \in x'^f$, $\forall 1 \leq i \leq k$, $h^{(i)}_{q^e}(x') = \min_{\xi \in E^*_{uo}} \{h^{(i)}_{q^{ae}}(x) + \omega^{(i)}(e_o) + \omega^{(i)}(\xi) + \omega^{(i)}_{in}(\theta) : \exists x \in x^f, \text{ s.t. } f(x, e_o\xi) = x'\}$.

Intuitively, a (θ, e_o)-insertion successor indicates the update of state estimates and energy levels after string θ is inserted before observable event e_o. Since event insertion does not change the system's estimate, the system's estimate gets updated after e_o occurs. While the intruder's estimate is updated with both θ and e_o. For a current state x' in the system's estimate x'^f, it may be reached through strings starting from some state(s) x in x^f and those strings may have different unobservable strings as suffixes. In this case, $h_{q^e}(x')$ indicates the *minimum* energy level at every dimension at x' with the occurrence of e_o and unobservable string ξ from some $x \in x^f$ s.t. $x' = f(x, e_o\xi)$. We also take into account of the cost of inserted string θ (potentially ϵ). Intuitively, if the worst case energy level is nonnegative, then the system's energy level is always nonnegative.

An *insertion-execution sequence* is a sequence of alternating states, inserted strings and executed observable events of the form: $\rho = y^e_1 \xrightarrow{e_1} z^e_1 \xrightarrow{\theta_1} y^e_2 \xrightarrow{e_2} z^e_2 \cdots \xrightarrow{e_{n-1}} z^e_{n-1} \xrightarrow{\theta_{n-1}} y^e_n$ where $\forall i \leq n$, $\theta_i \in E^*_o$, $e_i \in E_o$, $y^e_i \in Q^E$, $z^e_i \in Q^E \times E_o$, z^e_i is an e_i-execution successor of y^e_i and y^e_{i+1} is a (θ_i, e_i)-insertion successor of z^e_i. Such a sequence may be finite or infinite.

Lemma IV.4.1. *Given an insertion-execution sequence $\rho = y^e_1 \xrightarrow{e_1} z^e_1 \xrightarrow{\theta_1} y^e_2 \xrightarrow{e_2} z^e_2 \cdots \xrightarrow{e_{n-1}} z^e_{n-1} \xrightarrow{\theta_{n-1}} y^e_n$, let $Est_s(y^e_i) = (x^d_i, x^f_i)$ for all $1 \leq i < n$ and let $l = e_1 e_2 \cdots e_{n-1}$ and $l' = \theta_1 e_1 \cdots \theta_{n-1} e_{n-1}$, then $\delta_d(x^d_1, l') = x^d_n$ in $Obs_d(G)$ and $\delta(x^f_1, l) = x^f_n$ in $Obs_f(G)$.*

Proof. By induction. First, consider $y^e_1 \xrightarrow{e_1} z^e_1 \xrightarrow{\theta_1} y^e_2$. Since z^e_1 is an e_1-execution successor of y^e_1 and

y_2^e is an (θ_1, e_1)-insertion successor of z_1^e, then $(x_2^d, x_2^f) = \delta_{vs}(\delta_{vd}((x_1^d, x_2^f), \theta_1), e_1)$. So $\delta_d(x_1^d, \theta_1 e_1) = x_2^d$ and $\delta(x_1^f, e_1) = x_2^f$ by definitions of δ_{vd} and δ_{vs} in the verifier parallel composition.

Then suppose the result holds for $y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \cdots \xrightarrow{e_{k-1}} z_{k-1}^e \xrightarrow{\theta_{k-1}} y_k^e$. When $n = k+1$, by a similar argument, we can show that $\delta_d(x_k^d, \theta_k e_k) = x_{k+1}^d$ and $\delta(x_k^f, e_k) = x_{k+1}^f$. Combining the inductive hypothesis, we know $\delta_d(x_1^d, \theta_1 e_1 \cdots \theta_k e_k) = x_{k+1}^d$ and $\delta(x_1^f, e_1 \cdots e_k) = x_{k+1}^f$, so the result also holds at $k+1$, which completes the whole proof. □

Lemma IV.4.1 illustrates that in an insertion-execution sequence, the "original string" $e_1 e_2 \cdots e_{n-1}$ before insertion is defined in the feasible observer and the string $\theta_1 e_1 \cdots \theta_{n-1} e_{n-1}$ after insertion is defined in the desired observer. This result further implies that the string after insertion is always a safe one, so private safety is not violated following the insertion choices in any insertion-execution sequence.

The following theorem shows that the belief function always returns the *minimum* energy level at every dimension by strings that have the same observation and reach some state in the estimate, under certain insertion choices. By convention, we denote by $\rho_j = y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \cdots \xrightarrow{e_{j-1}} z_{j-1}^e \xrightarrow{\theta_{j-1}} y_j^e$ for $1 \leq j \leq n$ the j-th prefix of ρ. Also we let $V_m^{(i)}(s, s')$ denote the i-th component of the k-dimensional vector $V_m(s, s')$.

Theorem IV.4.1. *Given an insertion-execution sequence $\rho = y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$, let $Est_s(y_i^e) = (x_i^d, x_i^f)$ for all $1 \leq i \leq n$ and let $l = e_1 \cdots e_{n-1}$, then $\forall x \in x_n^f$, $\forall 1 \leq i \leq k$, $h_{y_n^e}^{(i)}(x) = \min_s \{V_m^{(i)}(s, s') : \exists x' \in x_1^f, s \in P^{-1}(l),$ s.t. $f(x', s) = x, \delta_d(x_1^d, P(s')) = x_n^d\}$ where string s is mapped to s' following Convention IV.3.1 under insertions indicated by ρ.*

Proof. Proof by induction on the length of l. Suppose $s = \xi_1 e_1 \cdots \xi_{n-1} e_{n-1} \xi_n$, $P(s) = l = e_1 \cdots e_n$ and s is mapped to $s' = \xi_1 \theta_1 e_1 \cdots \xi_n \theta_n e_n \xi_{n+1}$ where $\theta_j \in E_o^*$ and $P(s') = \theta_1 e_1 \cdots \theta_n e_n = l'$. Let $l_j = e_1 \cdots e_j$ and $l'_j = \theta_1 e_1 \cdots \theta_j e_j$ be the j-th prefix of l and l', respectively. Let $l_0 = \epsilon$ and $s_j = \xi_1 e_1 \cdots \xi_{j-1} e_j \xi_{j+1}$, with $s_0 = \epsilon$. We also suppose $\delta_{vd}(\delta_{vs}(\cdots \delta_{vs}(\delta_{vd}((x_1^d, x_1^f), \theta_1), e_1) \cdots, e_{j-1}), \theta_j) = (x_j'^d, x_j'^f)$ and $\delta_{vs}((x_j'^d, x_j'^f), e_j) = (x_{j+1}^d, x_{j+1}^f)$ in G_v.

Induction Basis: When $n = 0$, nothing is inserted and the result holds immediately.

Inductive Hypothesis: Assume that the result holds when $n = j-1$, i.e., for ρ_j.

Induction Step: Consider $n = j$.

First, $\delta_{vd}((x_j^d, x_j^f), \theta_j) = (x'^d_{j+1}, x'^f_{j+1})$ and $\delta_{vs}(\delta_{vd}((x'^d_{j+1}, x'^f_{j+1}), \theta_j), e_j) = (x^d_{j+1}, x^f_{j+1})$ hold by the definition of the verifier.

Then in ρ_{j+1}, z_j^e is an e_j-execution successor of y_j^e and y_{j+1}^e is a (θ_j, e_j)-insertion successor of z_j^e. So by definition, $\forall x' \in x^f_{j+1}$, $\forall 1 \le i \le j$, $h^{(i)}_{y^e_{j+1}}(x') = \min_{\xi_{j+1} \in E^*_{uo}} \{h^{(i)}_{y^e_j}(x) + \omega^{(i)}(e_j) + \omega^{(i)}(\xi_{j+1}) + \omega^{(i)}_{in}(\theta_j) : \exists x \in x^f_j, \text{ s.t. } f(x, e_j\xi_{j+1}) = x'\}$. From the inductive hypothesis, we have $h^{(i)}_{y^e_{j+1}}(x') = \min_{s_{j-1}} \min_{\xi_{j+1} \in E^*_{uo}} \{V^{(i)}_m(s_{j-1}, s'_{j-1}) + \omega^{(i)}(e_j) + \omega^{(i)}(\xi_{j+1}) + \omega^{(i)}_{in}(\theta_j) : \exists x'' \in x^f_1, x \in x^f_j, \text{ s.t. } f(x'', s_{j-1}) = x,$ $\delta_d(x^d_1, P(s'_{j-1})) = x^d_j, f(x, e_j\xi_{j+1}) = x'\}$. That is, $h^{(i)}_{y^e_{j+1}}(x') = \min_{s_j}\{V^{(i)}_m(s_j, s'_j) : \exists x'' \in x^f_1, s_j \in P^{-1}(l_j)$, s.t. $f(x'', s_j) = x', \delta_d(x^d_1, P(s'_j)) = x^d_{j+1}\}$. Thus the result holds when $n = j$, completing the proof. □

Given an energy information state $y^e \in Q^E$, for every $x \in x^f$ where $Est_s(y^e) = (x^d, x^f)$, each component of $h_{y^e}(x)$ may be due to different strings with the same projection but different unobservable substrings. This can be interpreted as follows: since the insertion function does not know the occurrence of unobservable strings, it should be "conservative" and take into account the system's *worst case* energy level in every dimension.

IV.4.3 Building the Energy Insertion Structure

We now define the *Energy Insertion Structure (EIS)* by construction in Algorithm IV.1. *EIS* just reflects the update of energy and augmented energy information states with event insertion and execution. It is a bipartite structure of the form: $(Q^E_Y, Q^E_Z, E_o, f^E_{yz}, f^E_{zy}, y^e_0, v_0, Q^E_l)$ where $Q^E_Y \subseteq Q^E$ is the set of energy information states; $Q^E_Z \subseteq Q^E \times E_o$ is the set of augmented energy information states; $f^E_{yz} : Q^E_Y \times E_o \to Q^E_Z$ is the transition function from Q^E_Y states to Q^E_Z states; $f^E_{zy} : Q^E_Z \times E^*_o \to Q^E_Y$ is the transition function from Q^E_Z states to Q^E_Y states; E_o is the set of observable events; $y^e_0 \in Q^E_Y$ is the initial state; $v_0 \in \mathbb{N}^k$ is the initial energy vector; and Q^E_l is the set of leaf states. We call a Q^E_Y state as Y-state and a Q^E_Z state as Z-state. A Z-state z^e is *deadlocking* if $\nexists \theta \in E^*_o$, s.t. $f^E_{zy}(z^e, \theta)!$. Deadlocking Z-states are undesirable and will be pruned away in constructing EIS.

Algorithm IV.1: Construction of EIS

Input : $Obs(G)$, G_v, v_0

Output : $EIS = (Q_Y^E, Q_Z^E, E, f_{yz}^E, f_{zy}^E, E_o, y_0^e, v_0, Q_l^E)$

$Q_Y^E = \{y_0^e\}$ where $Est_s(y_0^e) = (x_{obs,0}, x_{obs,0})$, $\forall x \in x_{obs,0}$, $\forall i \leq k$, $h_{y_0^e}^{(i)}(x) = \min_{\xi \in E_{uo}^*} \{V^{(i)}(\xi) : f(x_0, \xi) = x\}$, and $Q_Z^F = \emptyset$, $Q_l^E = \emptyset$;

$EIS_{pre} = DoDFS(y_0^e, Obs(G), G_v)$;

$EIS = Prune(EIS_{pre})$;

Procedure: $DoDFS(y^e, Obs(G), G_v)$

for $e_o \in E_o$, *s.t.* $\delta(x^f, e_o)!$ *in* $Obs(G)$, *where* $Est_s(y^e) = x_v = (x^d, x^f)$ **do**
- let z^e be an e_o-execution successor of y^e;
- add transition $y^e \xrightarrow{e_o} z^e$ to f_{yz}^E;
- **if** $z^e \notin Z^E$ **then**
  - $Q_Z^E = Q_Z^E \cup \{z^e\}$;
  - **for** $\theta \in E_O^*$, *s.t.* $\exists \tilde{x}_v = \delta_{vd}(x_v, \theta)$, $\delta_{vs}(\tilde{x}_v, e_o)!$ **do**
    - let y'^e be an (θ, e_o)-insertion successor of z^e;
    - add transition $z^e \xrightarrow{\theta} y'^e$ to f_{zy}^E;
    - **if** $y'^e \notin Q_Y^E$ **then**
      - **if** y'^e *is energy safe* **then**
        - $Q_Y^E = Q_Y^E \cup \{y'^e\}$;
        - **if** *there exists a run from* y_0^e: $r_e = y_0^e \xrightarrow{e_0} z_0^e \xrightarrow{\theta_0} y_1^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y'^e$ *and* $\exists j \leq n$, *s.t.* $y_j^e \preccurlyeq y'^e$ **then**
          - let $Sub(y'^e) = y_j^e$, stop searching from y'^e, $Q_l^E = Q_l^E \cup \{y'^e\}$;
        - **else**
          - $DoDFS(y'^e, Obs(G), G_v)$;
      - **if** y'^e *is not energy safe* **then**
        - $Q_Y^E = Q_Y^E \cup \{y'^e\}$, $Q_l^E = Q_l^E \cup \{y'^e\}$, stop searching from y'^e, ignore all θ' s.t. $\theta < \theta'$;

Procedure: $Prune(EIS_{pre})$

for $z^e \in Q_Z^E$ *that is deadlocking* **do**
- remove z^e and all $y^e \in Q_Y^E$, s.t. $f_{yz}^E(y^e, e_o) = z^e$ for some $e_o \in E_o$;
- take the accessible part of the structure;

Algorithm IV.1 builds the state space of EIS recursively by adding (θ, e_o)-insertion successors and e_o-execution successors into the structure. In general, EIS represents a game with full observation between the insertion function and the environment. The environment plays at Y-states and the insertion function plays at Z-states. The procedure $DoDFS$ builds the state space of the EIS in a depth-first search like process. The game is initiated from y_0^e where the system plays first by executing observable events. The state estimate component of y_0^e contains the initial state of the observer and the initial state of the desired observer. For the energy level matrix $Lev_e(y_0^e)$, we track the minimum energy level of the system by unobservable strings. In Line 4, the environment plays by executing e_o if e_o is defined from the system's estimate x^f in observer $Obs(G)$. Then we create an e_o-execution successor z^e and define a f_{yz}^E transition out of y^e. Note that no string has been inserted yet and we create z^e simply to indicate that some string may be inserted before observable event e_o.

After that, the games goes on and it is the insertion function's turn to play by inserting stings. In Line 9, θ is a logically feasible insertion choice if a δ_{vd} transition labeled with θ is defined in the verifier and the δ_{vd} transition is followed by a δ_{vs} transition labeled by some observable event e_o. That means θ can be inserted before e_o in the logical sense, without considering the energy constraint. So we create a (θ, e_o)-insertion successor y'^e and define a f_{zy}^E transition out of z^e, indicating that θ has been inserted before e_o. Since the initial energy vector is fixed and insertion is costly, there may only be a finite set of finite-length inserted strings that lead to nonnegative energy levels. When y'^e is safe, i.e., θ is inserted before e_o without violating the energy constraint, we proceed to check the condition in Line 16. If there exists an initial run r_e ending in y'^e and $y_j^e \in r_e$ for some $j < n$, s.t. y'^e subsumes y_j^e, then we know the state estimate $Est_s(y_j^e)$ is reached again, i.e., $Est_s(y_j^e) = Est_s(y'^e)$. Let $Est_s(y_j^e) = (x_j^d, x_j^f)$, then we know there exists a simple cycle $x_j^f \xrightarrow{e_j} x_{j+1}^f \cdots \xrightarrow{e_{n-1}} x_j^f$ in the feasible observer $Obs_f(G)$ (also in the observer $Obs(G)$). There also exists a cycle starting from and ending in x_j^d in the desired observer, whose corresponding loop is $l = \theta_j e_j \cdots \theta_{n-1} e_{n-1}$. It is also the case that $\forall x \in x_j^d$, $\forall s \in P^{-1}(l)$, s.t. $f(x, s) = x$, we have $V(s) + \sum_{i=j}^{n-1} \theta_i \geq \overrightarrow{0}$. In words, even after considering the cost of inserting $\theta_j, \cdots, \theta_{n-1}$ into the original string, the system's energy level vector is still nondecreasing in every dimension.

Even though the structure may be further expanded, we terminate searching from y'^e and define $Sub(y'^e)$ to store the state subsumed by y'^e. Note that y'^e and y_j^e share the same state estimate while the energy level at y'^e is no less than that of y_j^e in component-wise sense. No matter what decision is made by the environment at y'^e, if the insertion function makes the same decision at the succeeding state of y'^e as it does at the succeeding state of y^e_j, then all the new succeeding states created in this manner are energy safe as well. This is consistent with the monotonicity property discussed at the end of Section V.3. Later on, we will see this observation ensures finiteness of EIS.

If no cycle is detected, we call $DoDFS$ again in Line 18 to continue searching until no more states are added to the structure. On the other hand, if y'^e is not energy safe, system's energy level is below 0 at some dimension. Then we stop searching from y'^e in Line 20 and discard longer string θ' where $\theta < \theta'$. Since $\omega_{in}(\theta') < \omega_{in}(\theta) \leq 0$, insertion of θ' would inevitably drop the energy level vector below 0 at certain dimension.

$DoDFS$ may result in some deadlocking Z-states where no insertion can be made. We denote by EIS_{pre} the intermediate structure obtained after $DoDFS$, then remove deadlocking Z-states and their preceding Y-states recursively in Procedure *Prune* since the observable events from Y-states can not be blocked from happening. More reasoning can be found in [119], where a similar pruning process is conducted. *Prune* works like calculating *supremal controllable sublanguage* [23] by viewing the environment's winning states as undesirable, f^E_{yz} transitions as uncontrollable, f^E_{zy} transitions as controllable, and Y-states as marked. Next, we show Algorithm IV.1 stops after a finite number of steps and returns a finite structure, namely, EIS.

Theorem IV.4.2. *The state space of EIS is finite.*

Proof. By contradiction. Suppose that EIS is infinite. The number of outgoing transitions at each state is finite since E_o is finite and there are only a finite number of insertion choices defined at a Z-state due to energy constraints. Then by König's lemma (see, e.g., [69]), there exists an infinite run $y^e_1 \xrightarrow{e_1} z^e_1 \xrightarrow{\theta_1} y^e_2 \xrightarrow{e_2} z^e_2 \xrightarrow{\theta_2} y^e_3 \cdots$ in EIS. From Algorithm IV.1, every state in the run is energy safe and it is never the case that $\exists i < j$, s.t. $y^e_i \preccurlyeq y^e_j$. However, this contradicts the well-quasi ordering $\preccurlyeq$ on safe energy information states. $\square$

The size of EIS is bounded by Ackermann function [92] following a similar augment as in [40], which also presented a procedure of "unfolding" the game graph until some simple cycles are formed or the energy level drops below 0. Since Ackermann functions are not primitive recursive, the complexity of EIS exceedsits counterpart without energy constraint, i.e., All Insertion Structure in [54].

In EIS, we call a leaf state $y^e \in Q_l^E$ as a *good leaf state* if y^e is energy safe, otherwise, we call it a *bad leaf state*. We denote the sets of good and bad leaf states by Q_{lg}^E and Q_{lb}^E, respectively. In order to win the game and solve Problem IV.3.1, the insertion function should make decisions such that only good leaf states are reached. The environment just does the opposite to prevent the insertion function from winning, thus the game on EIS is a *zero sum* reachability game. We elaborate the reasoning and discuss both players' strategies in the next section.

Example IV.4.1. *Let the automaton G in Figure IV.1 be with observable events $E_o = \{a,b,c,d\}$, unobservable events $E_{uo} = \{u_1,u_2,u_3,u_4,u_5,u_6,u_7\}$, and secret states $X_S = \{x_7,x_8,x_{10}\}$. The system is granted with initial energy $v_0 = [9,9]^T$ where T stands for the transpose of a matrix. The weight function in this example is 2-dimensional and the weight vector of each event is show in Figure IV.1. Besides, the insertion weight function ω_{in} is defined as follows: $\omega_{in}(a) = [-3,-6]^T$, $\omega_{in}(b) = [-1,-3]^T$, $\omega_{in}(c) = [-2,-2]^T$, $\omega_{in}(d) = [-3,-1]^T$.*

The observer is shown in Figure IV.2 with states: $A = \{x_0,x_3,x_4,x_9\}$, $B = \{x_1\}$, $C = \{x_2\}$, $D = \{x_5,x_6\}$, $E = \{x_7,x_8\}$ and $F = \{x_{10}\}$. The system is not current state opaque due to states E and F, thus we apply insertion functions to enforce opacity. The desired observer $Obs_d(G)$ is obtained by removing E and F from $Obs(G)$ and taking the accessible part, while the feasible observer $Obs_f(G)$ is obtained by adding self-loops for every event in E_o at every state in $Obs(G)$; their figures are omitted here due to space limitations. Next we build the verifier G_v in Figure IV.3 following Definition IV.4.1, where dashed lines indicate δ_{vd} transitions and solid lines indicate δ_{vs} transitions. G_v contains all potentially feasible insertion choices.

Then we follow Algorithm IV.1 to build EIS in Figure IV.4, where square states stand for Y-states while oval states stand for Z-states. In $DoDFS$, the game is initiated from y_0^e where the environment plays first: it can execute events a, b or c. For example, if b is executed, then b-execution successor $z_0^e = (y_0^e, b)$ is reached where it is the insertion function's turn to play; while if a is inserted, then a-insertion successor y_1^e is reached. We have $Est_s(y_1^e) = (C,D)$ as $\delta_{vd}((A,A),a) = (B,A)$

and $\delta_{vs}((B,A),b) = (C,D)$ *in* G_v. *We also have* $h^{(1)}_{y^e_1}(x_5) = \min\{h^{(1)}_{y^e_0}(x_3) + \omega^{(1)}(b) + \omega^{(1)}_{in}(a), h^{(1)}_{y^e_0}(x_4) + \omega^{(1)}(b) + \omega^{(1)}_{in}(a)\} = 5$, $h^{(2)}_{y^e_1}(x_5) = \min\{h^{(2)}_{y^e_0}(x_3) + \omega^{(2)}(b) + \omega^{(2)}_{in}(a), h^{(2)}_{y^e_0}(x_4) + \omega^{(2)}(b) + \omega^{(2)}_{in}(a)\} = 3$, $h^{(1)}_{y^e_1}(x_6) = \min\{h^{(1)}_{y^e_1}(x_5) + \omega^{(1)}(u_4), h^{(1)}_{y^e_1}(x_5) + \omega^{(1)}(u_5)\} = 0$ *and* $h^{(2)}_{y^e_1}(x_6) = \min\{h^{(2)}_{y^e_1}(x_5) + \omega^{(2)}(u_4), h^{(2)}_{y^e_1}(x_5) + \omega^{(2)}(u_5)\} = 0$. *Hence we have* $y^e_1 = \{(C,D), \begin{bmatrix} 5, & 0 \\ 3, & 0 \end{bmatrix}\}$. *The other states are calculated similarly.*

The first component of $h^{(2)}_{y^e_1}(x_5) = [5,3]^T$ *comes from string* u_2u_3b *and insertion of* a, *while the second component comes from string* u_1u_3b *and insertion of* a. *Since the insertion function does not know whether* u_2u_3b *or* u_1u_3b *occurs when it observes* b, *it has to estimate the worst case energy level, which is consistent with Theorem V.4.1. We list the energy and augmented energy information states obtained from* $DoDFS$ *in Table IV.1.*

After $DoDFS$, *we find* $y^e_2 \preccurlyeq y^e_4$, $y^e_{21} \preccurlyeq y^e_{19}$ *and* $y^e_{23} \preccurlyeq y^e_{19}$, *so we stop searching from* y^e_4, y^e_{21} *and* y^e_{23}. *Besides,* y^e_5, y^e_7, y^e_8, y^e_9, y^e_{10}, y^e_{11}, y^e_{12}, y^e_{16}, y^e_{17}, y^e_{18}, y^e_{24} *are not energy safe so they are the bad leaf states. Furthermore, Z-state* z^e_5 *is deadlocking since no transition is defined out of it. Then we prune away* z^e_5 *and its preceding Y-state* y^e_{13} *in process Prune of Algorithm IV.1. The final* EIS *is shown in Figure IV.4, where the dashed lines represent deleted states in the pruning process from* EIS_{pre} *to* EIS.

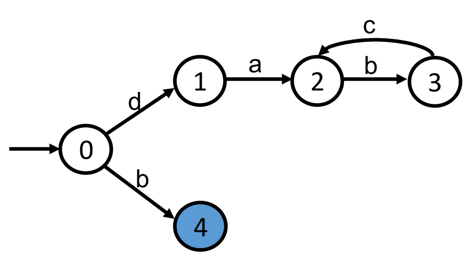

Figure IV.1: System G with secret states x_7, x_8, x_{10}

$y_0^e = \{\{A,A\}, \begin{bmatrix} 9, & 10, & 7, & 7 \\ 9, & 10, & 9, & 8 \end{bmatrix}\}$	$z_0^e = \{\{A,A\}, \begin{bmatrix} 9, & 10, & 7, & 7 \\ 9, & 10, & 9, & 8 \end{bmatrix}, b\}$
$y_1^e = \{(C,D), \begin{bmatrix} 5, & 0 \\ 3, & 0 \end{bmatrix}\}$	$z_1^e = \{(C,D), \begin{bmatrix} 5, & 0 \\ 3, & 0 \end{bmatrix}, c\}$
$y_2^e = \{(B,E), \begin{bmatrix} 2, & 1 \\ 2, & 1 \end{bmatrix}\}$	$z_2^e = \{(B,E), \begin{bmatrix} 2, & 1 \\ 2, & 1 \end{bmatrix}, c\}$
$y_3^e = \{(B,E), \begin{bmatrix} 3, & 2 \\ 1, & 0 \end{bmatrix}\}$	$z_3^e = \{(B,E), \begin{bmatrix} 1, & 0 \\ 3, & 2 \end{bmatrix}, c\}$
$y_4^e = \{(B,E), \begin{bmatrix} 2, & 1 \\ 2, & 1 \end{bmatrix}\}$	$y_5^e = \{(B,E), \begin{bmatrix} 4, & 3 \\ 0, & -1 \end{bmatrix}\}$
$y_6^e = \{(B,E), \begin{bmatrix} 1, & 0 \\ 3, & 2 \end{bmatrix}\}$	$z_4^e = \{(B,E), \begin{bmatrix} 3, & 2 \\ 1, & 0 \end{bmatrix}, c\}$
$y_7^e = \{(B,E), \begin{bmatrix} 0, & -1 \\ 4, & 3 \end{bmatrix}\}$	$y_8^e = \{(B,E), \begin{bmatrix} -4, & -5 \\ 0, & -1 \end{bmatrix}\}$
$y_9^e = \{(B,E), \begin{bmatrix} -2, & -3 \\ -2, & -3 \end{bmatrix}\}$	$y_{10}^e = \{(B,E), \begin{bmatrix} -3, & -4 \\ -1, & -2 \end{bmatrix}\}$
$y_{11}^e = \{(B,E), \begin{bmatrix} -1, & -2 \\ -3, & -4 \end{bmatrix}\}$	$y_{12}^e = \{(C,D), \begin{bmatrix} 2, & -3 \\ -2, & -5 \end{bmatrix}\}$
$y_{13}^e = \{(D,D), \begin{bmatrix} 8, & 9 \\ 3, & 6 \end{bmatrix}\}$	$z_5^e = \{(D,D), \begin{bmatrix} 8, & 9 \\ 3, & 6 \end{bmatrix}, c\}$
$z_6^e = \{(A,A), \begin{bmatrix} 9, & 10, & 7, & 7 \\ 9, & 10, & 9, & 8 \end{bmatrix}, c\}$	$y_{14}^e = \{(B,F), \begin{bmatrix} 5 \\ 1 \end{bmatrix}\}$
$y_{15}^e = \{(B,F), \begin{bmatrix} 3 \\ 3 \end{bmatrix}\}$	$y_{16}^e = \{(B,F), \begin{bmatrix} 2 \\ -4 \end{bmatrix}\}$
$y_{17}^e = \{(B,F), \begin{bmatrix} 0 \\ -2 \end{bmatrix}\}$	$y_{18}^e = \{(B,F), \begin{bmatrix} -2 \\ 0 \end{bmatrix}\}$
$z_7^e = \{(A,A), \begin{bmatrix} 9, & 10, & 7, & 7 \\ 9, & 10, & 9, & 8 \end{bmatrix}, a\}$	$y_{19}^e = \{(B,B), \begin{bmatrix} 1 \\ 1 \end{bmatrix}\}$
$z_8^e = \{(B,B), \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b\}$	$y_{20}^e = \{(C,C), \begin{bmatrix} 2 \\ 1 \end{bmatrix}\}$
$z_9^e = \{(C,C), \begin{bmatrix} 2 \\ 1 \end{bmatrix}, c\}$	$y_{21}^e = \{(B,B), \begin{bmatrix} 4 \\ 3 \end{bmatrix}\}$
$z_{10}^e = \{(B,B), \begin{bmatrix} 1 \\ 1 \end{bmatrix}, d\}$	$y_{22}^e = \{(C,C), \begin{bmatrix} 1 \\ 2 \end{bmatrix}\}$
$z_{11}^e = \{(C,C), \begin{bmatrix} 1 \\ 2 \end{bmatrix}, c\}$	$y_{23}^e = \{(B,B), \begin{bmatrix} 3 \\ 4 \end{bmatrix}\}$
$y_{24}^e = \{(B,E), \begin{bmatrix} 0, & -1 \\ -4, & -5 \end{bmatrix}\}$	

Table IV.1: Energy and augmented energy information states

IV.5 Solve the Constrained Opacity Enforcement Problem

In this section, we discuss the strategies for both players to win the game on the Energy Insertion Structure. We also show that the insertion function's winning strategies in EIS lead to sound

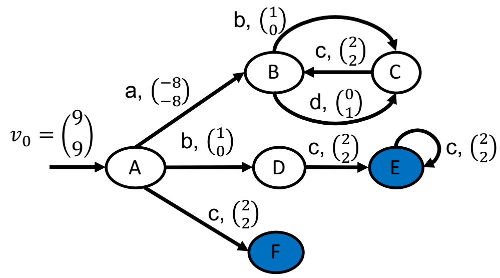

Figure IV.2: The observer $Obs(G)$

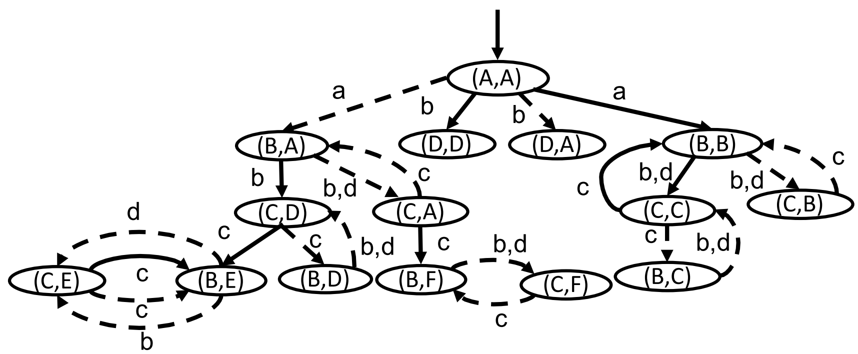

Figure IV.3: The verifier G_v where dashed transitions are δ_{vd} transitions and solid transitions are δ_{vs} transitions

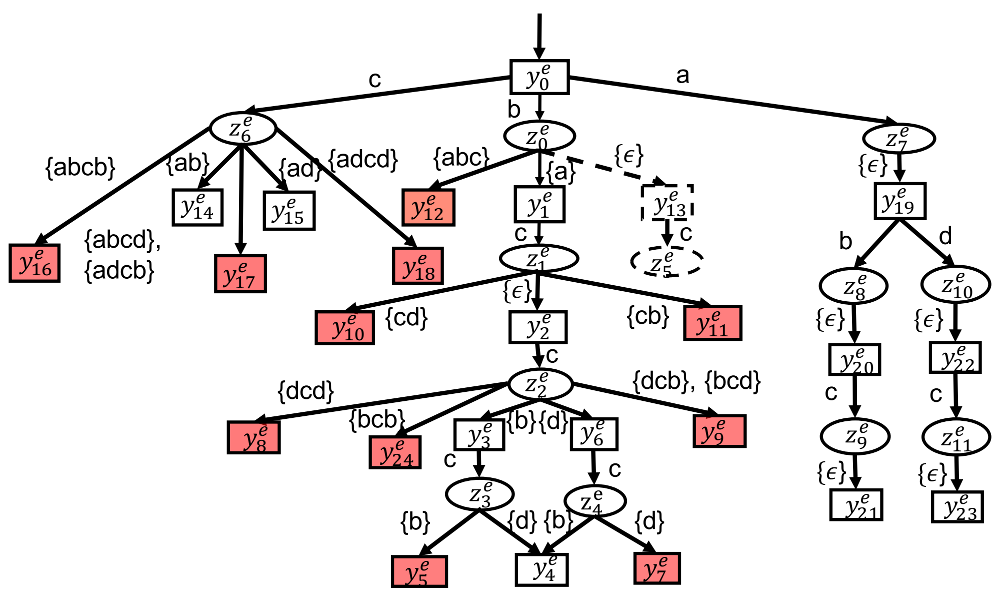

Figure IV.4: Energy Insertion Structure (without dashed states) solutions to Problem IV.3.1.

By definition and Theorem IV.4.2, the runs in EIS are finite insertion-execution sequences discussed in last section; we denote the set of runs in EIS by $Run(EIS)$. Given $r_e \in Run(EIS)$, we denote by $y^e \in r_e$ and $z^e \in r_e$ if y^e (respectively z^e) is a Y-state (respectively Z-state) in r_e. Let $Last_Y(r_e)$ and $Last_Z(r_e)$ be the last Y-state and Z-state of r_e, respectively, and denote by $Run_y(EIS)$ (respectively $Run_z(EIS)$) the set of runs whose last states are Y-states (respectively Z-states).

Given an initial run $r_e = y_0^e \xrightarrow{e_0} z_0^e \xrightarrow{\theta_0} y_1^e \xrightarrow{e_1} \cdots y_{n-1}^e \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$, the *edit projection* P_e : $Run(EIS) \to P[\mathcal{L}(G)]$ is defined such that $P_e(r_e) = e_0 e_1 \cdots e_{n-1}$. So P_e just returns the original string before any insertion takes place. For $r_e \in Run(EIS)$, we denote it by $r_e(l)$ if $P_e(r_e) = l$. Besides, we call $\theta_0 e_0 \theta_1 e_1 \cdots \theta_{n-1} e_{n-1}$ as the *generated string* of r_e and denote it by $l_g(r_e)$. In other words, $l_g(r_e)$ is the string after insertion. By Lemma IV.4.1, we know that $\delta_d(x_{obs,0}, l_g(r_e))$ is defined in $Obs_d(G)$, so $l_g(r_e) \in \mathcal{L}(Obs_d(G)) = L_{safe}$, i.e., l is mapped to a safe string by insertion decisions in EIS.

Then we define strategies for both players in EIS. The *insertion function's strategy (insertion strategy)* is defined as $\pi_{in} : Run_z(EIS) \to E_o^*$ and the *environment's strategy* as $\pi_{en} : Run_y(EIS) \to E_o$. When it is a player's turn to play, it selects a transition according to its strategies. Since the insertion function does not know the occurrence of unobservable events and makes decisions from its observations, its strategy is called *observation based*. Denote the set of all insertion strategies by Π_{in} and the set of all environment's strategies by Π_{en}. From an insertion strategy, we know exactly the decisions of an insertion function, so from now on, we use "insertion strategy" and "insertion function" interchangeably.

A strategy $\pi_i \in \Pi_i$ for player $i \in \{in, en\}$ in EIS is called *information state based* if the decisions only depend on the current energy (augmented energy) information state. In other words, $\pi_i \in \Pi_i$ is information state based if $\pi_i(r_f) = \pi_i(r'_f)$ for all $r_f, r'_f \in Run(EIS)$ such that $Last(r_f) = Last(r'_f)$. Therefore, information state based strategies for the insertion function and the environment can be represented as $\pi_{in} : Q_Z^E \to E_o^*$ and $\pi_{en} : Q_Y^E \to E_o$, respectively. We also call such strategies as or *positional*. From results in [4, 43], positional strategies are sufficient to win a reachability game so we assume both players' strategies are positional in the rest of this section.

If the insertion function plays π_{in} while the environment plays π_{en} from the initial state y_0^e, then a unique initial run, denoted by $r_e(\pi_{in},\pi_{en})$, is generated. We also define $Run(\pi_{in}, y^e) = \{y^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e : \forall i < n, \theta_i = \pi_{in}(y^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \cdots y_i^e \xrightarrow{e_i} z_i^e)\}$ as the set of runs starting from y^e and *consistent* with insertion strategy π_{in}, i.e., insertion decisions in the run are specified by π_{in}. The set of runs consistent with an environment's strategy π_{en} are defined analogously and we denote it by $Run(y^e, \pi_{en})$.

In EIS, we say that the insertion function wins the game if only good leaf states are reached while the environment wins if bad leaf states are reached. Thus they play a finite-duration *zero sum* reachability game. By defining the energy information states, we have constructed a game under *full observation* on EIS. Therefore, either the supervisor or the environment has a *winning strategy* [4]. Formally speaking, $\pi_{in} \in \Pi_{in}$ is *winning* from y^e if $\forall r_e \in Run(\pi_{in}, y^e)$, $Last_Y(r_e) \in Q_l^E \Rightarrow Last_Y(r_e) \in Q_{lg}^E$, i.e., π_{in} is a winning strategy for the insertion function if all runs consistent with it end in a good leaf state. In other words, the insertion function wins if private safety is satisfied and the energy level of the system is never below 0 in every dimension.

We define the insertion function's *winning region* Win_{in} in EIS as the set of states where it has a strategy to reach a good leaf state no matter what strategy the environment plays. This is a commonly used concept in graph game theory, see., e.g. [4]. Then we present Algorithm IV.2 to compute Win_{in}.

Algorithm IV.2: Compute the insertion function's winning region

---

Input : EIS
Output: Win_{in}

Remove all bad leaf states from EIS;
while $\exists z^e \in Q_Z^F$, *s.t.* z^e *is deadlocking* **do**
  Remove z^e and all $y^e \in Q_Y^E$, s.t. $f_{yz}^E(y^e, e_o) = z^e$ for some $e_o \in E_o$;
  Take the accessible part of the structure;
Denote the remaining structure by EIS_w;
if EIS_w *is not empty* **then**
  Return all states in EIS_w;
else
  Return $\emptyset$;

In Algorithm IV.2, we prune away bad leaf states and calculate the winning region for the insertion function in an iterative manner. We first remove all bad leaf states from EIS. If the removal of bad leaf states results in some deadlocking Z-states, then we know all transitions from such Z-states lead to bad leaf states, where the insertion function loses the game *for sure*. Thus we further remove those Z-states and their preceding Y-states where the environment has a way to reach the deadlocking Z-states. This process continues until no more states are removed and we denote the resulting structure by EIS_w. The pruning process works in a *fixed-point iteration* manner.

By definition, a privately safe insertion function (strategy) maps every string in $P[\mathcal{L}(G)]$ to a safe one. However, state pruning may remove all potentially feasible insertion choices for a particular string if they all violate energy constraints. Thus we need to guarantee that all strings in $P[\mathcal{L}(G)]$ are still preserved in the EIS_w after the pruning. Before proving that assertion, we present the following result from Algorithm 13.

Lemma IV.5.1. *If* $Win_{in} \neq \emptyset$*, then* $\nexists l \in P[\mathcal{L}(G)]$*, s.t.* $\forall \pi_{in} \in \Pi_{in}$*,* $\forall r_e \in Run(\pi_{in}, y_0^e)$ *with* $P_e(r_e) = l$*,* $Last_Y(r_e) \in Q_{lb}^E$ *in* EIS*.*

Proof. By contradiction. We assume $\exists l \in P[\mathcal{L}(G)]$, s.t. $\forall \pi_{in} \in \Pi_{in}, \forall r_e \in Run(\pi_{in}, y_0^e)$ with $P_e(r_e) = l$ in EIS, $Last_Y(r_e) \in Q_{lb}^E$. Suppose $l = e_0 \cdots e_{n-1}$ and $r_e = y_0^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e \in Run(\pi_{in}, y_0^e)$. Since $Last_Y(r_e) \in Q_{lb}^E$ for all $r_e \in Run(\pi_{in}, y_0^e)$ with $P_e(r_e) = l$ and for all $\pi_{in} \in \Pi_{in}$, the last Y-state of every run in $Run(\pi_{in}, y_0^e)$ with $P_e(r_e) = l$ is pruned in Algorithm 13. Then we know the last Z-state of each run in $Run(\pi_{in}, y_0^e)$ with $P_e(r_e) = l$ becomes deadlocking so those z_{n-1}^e are pruned away as well. Furthermore, we also prune away all preceding Y-states y_{n-1}^e such that $f_{yz}^E(y_{n-1}^e, e_{n-1}) = z_{n-1}^e$ by Algorithm 13. This process continues until the initial state y_0^e is pruned, so EIS_w is empty. □

Next we slightly modify EIS_w: merge y^e with $Sub(y^e)$ by letting all transitions going to y^e reach $Sub(y^e)$ instead, if $Sub(y^e)$ is defined in Algorithm IV.1. Intuitively, we assume that the game continues at the leaf states of EIS_w, which share the same state estimate with the state subsumed by them. We denote the resulting structure by EIS_m and extend concepts of runs and both players' strategies to EIS_m. Besides, the energy level vector at each leaf state is no less than that at the state subsumed by the same leaf state. Thus if every leaf state is energy safe, the system's energy level vector never contains a negative element when their state estimates are reached again. In this way the game is extend to be infinite-duration without loss of generality since we assume that the insertion functions in EIS_w always make the same decisions at each leaf state and the state subsumed by it. Therefore, if the insertion function plays according to strategies in EIS_m, *it will always maintain the system's energy level above 0 in each dimension*. This is an implication of the monotonicity of energy game discussed at the end of Section V.3 : if the insertion function wins the game from some state with energy level vector $v \in \mathbb{N}^k$, it also wins the game from the same state with any energy level vector $v' \geq v$.

In EIS_m, we define the *unmodified language* $\mathcal{L}_u(EIS_m) = \{l \in P[\mathcal{L}(G)] : \exists r_e \in Run(EIS_m), \text{ s.t. } P_e(r_e) = l\}$, where $Run(EIS_m)$ denotes the set of runs in EIS_m. $\mathcal{L}_u(EIS_m)$ just "retrieves" the original language before any insertion takes place. Then we prove a property of $\mathcal{L}_u(EIS_m)$ in Lemma IV.5.2.

Lemma IV.5.2. *If $Win_{in} \neq \emptyset$, then $\mathcal{L}_u(EIS_m) = P[\mathcal{L}(G)]$.*

Proof. By the definition of $\mathcal{L}_u(EIS_m)$, $\mathcal{L}_u(EIS_m) \subseteq P[\mathcal{L}(G)]$ holds immediately. Thus we only need to show $P[\mathcal{L}(G)] \subseteq \mathcal{L}_u(EIS_m)$ and we proceed by contradiction. Assume that $\mathcal{L}_u(EIS_m) \nsubseteq P[\mathcal{L}(G)]$ and $\exists l \in P[\mathcal{L}(G)]$ but $l \notin \mathcal{L}_u(EIS_m)$. Then by construction of EIS and EIS_m, there exists a finite prefix $l' < l$, s.t. $\forall \pi_{in} \in \Pi_{in}$, $\forall r_e \in Run(\pi_{in}, y_0^e)$ with $P_e(r_e) = l'$, $Last_Y(r_e) \in Q_{lb}^E$. That is, there exists a finite string in $P[\mathcal{L}(G)]$ such that no insertion strategy in EIS_m can map it to a safe string without reaching a bad leaf state. However, that means $Win_{in} = \emptyset$ by Lemma IV.5.1, which contradicts the assumption. □

We are now ready to state one of the main results in this chapter. Given a winning insertion strategy in EIS, we can always construct an insertion function solving Problem IV.3.1. Conversely, if there exists an insertion function solving Problem IV.3.1, we can always find a winning insertion strategy in EIS.

Theorem IV.5.1. *There exists an insertion function solving Problem IV.3.1 if and only if there exists a winning strategy for the insertion function in EIS.*

Proof. The "only if" part. We show by contrapositive, i.e., if no winning insertion strategy exists in EIS, then there does not exist an insertion function solving Problem IV.3.1. If no strategy exists for the insertion function to reach good leaf states in EIS, then we know the winning set Win_{in} is empty, i.e., Algorithm 13 returns an empty set. So by Lemma IV.5.1, $\exists s \in \mathcal{L}(G)$ with $P(s) = l = e_0 \cdots e_{n-1}$, s.t. for all initial $r_e(l) \in Run(EIS)$, $Last_Y(r_e(l)) \in Q_l^E \Rightarrow Last_Y(r_e(l)) \in Q_{lb}^E$, i.e., all runs with original string l end in bad leaf states. Then by the pruning process in Algorithm 13, every initial run $r_e(l)$ would be removed, thus the initial state of EIS is also removed and EIS_w becomes empty. From the construction in Algorithm IV.1, for all feasible insertion choices $\theta_0, \cdots, \theta_{n-1}$ s.t. s is mapped to s' by Convention IV.3.1 and $\theta_0 e_0 \cdots \theta_{n-1} e_{n-1} \in L_{safe}$, we have that $V_m(s, s') < \overrightarrow{0}$. In other words, no matter what string is inserted into l, the system's energy level would drop below 0 at some dimension. Thus no insertion function solves Problem IV.3.1.

The "if" part. Suppose that π_{in} is a winning insertion strategy in EIS. Since we follow Algorithm 13 to obtain Win_{in} and EIS_w, then π_{in} is also in EIS_w. Then we extend EIS_w to EIS_m by merging states. By definition of EIS, the state estimate component of each state is in $X_v \subseteq X_{obsd} \times X_{obs}$ so the intruder's estimate is always in X_{obsd}. Since by the definition of the desired observer, $\forall x_{obsd} \in X_{obsd}$, $x_{obsd} \notin 2^{X_S}$, we know π_{in} maps every string in $P[\mathcal{L}(G)]$ into a safe string.

Besides, $\forall s \in \mathcal{L}(G)$ with $P(s) = l = e_0 e_1 \cdots e_{n-1}$, suppose that there exists a run $r_e(l) = y_0^e \xrightarrow{e_0} z_0^e \xrightarrow{\theta_0} y_1^e \xrightarrow{e_1} \cdots y_{n-1}^e \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$ consistent with π_{in} in EIS_m, denoted by $r_{\pi_{in}}(l)$. Every $y^e \in r_{\pi_{in}}(l)$ is energy safe and the belief function in each energy information state returns the minimum energy level of the system at every dimension under certain insertion choices. Then from Theorem V.4.1, we know that $\forall s \in P^{-1}(l) \cap \mathcal{L}(G)$, $V_m(s, s_{\pi_{in}}) \geq \overrightarrow{0}$, therefore π_{in} solves Problem IV.3.1. □

The above theorem shows the completeness and soundness of Algorithms IV.1 and 13. Therefore, Problem IV.3.1 can be solved by first building EIS and then finding the insertion function's winning strategies if they exist. As was shown in last section, the state space of EIS is bounded by

Ackermann function [87]. Besides, both the winning set and strategies for a reachability game can be computed in linear time with respect to the size of EIS [4]. Therefore we have the complexity bound for solving Problem IV.3.1. We end this section by revisiting our running example.

Example IV.5.1. *We revisit Example IV.4.1 and synthesize insertion functions to solve Problem IV.3.1. We follow Algorithm 13 and build EIS_w in Figure V.4. In Algorithm 13, all bad leaf states are removed and the winning region Win_{in} is the set of states in EIS_w. Here we use dashed lines to connect each good leaf state with the state subsumed by it. Observe that condition $\mathcal{L}_u(EIS_m) = P[\mathcal{L}(G)]$ holds for EIS_m in Figure V.4 so that every string in $P[\mathcal{L}(G)]$ may be mapped to some safe strings. From EIS_w, we find one winning insertion strategy, which solves Problem IV.3.1 and is indicated by blue lines in Figure V.4. Finally, we encode this selected insertion function as an I/O automaton in Figure IV.6, where the insertion decisions are explicitly shown.*

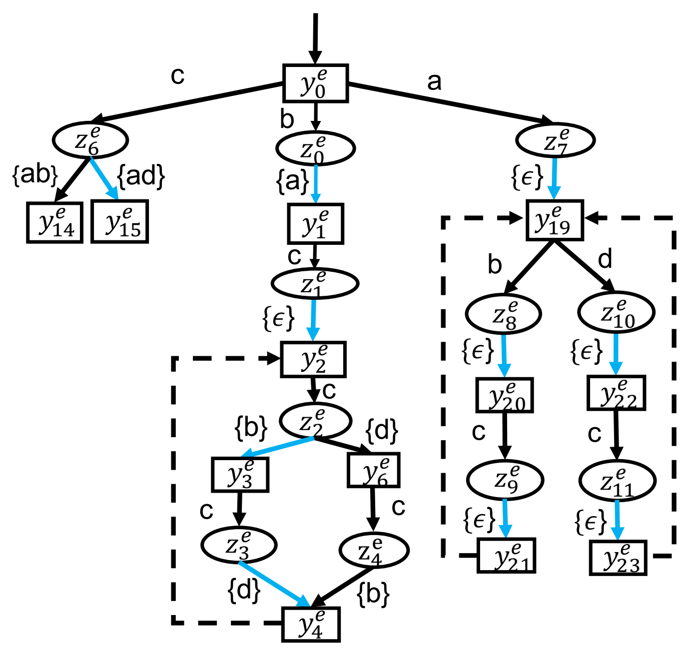

Figure IV.5: EIS_w with a winning insertion strategy indicated by blue lines

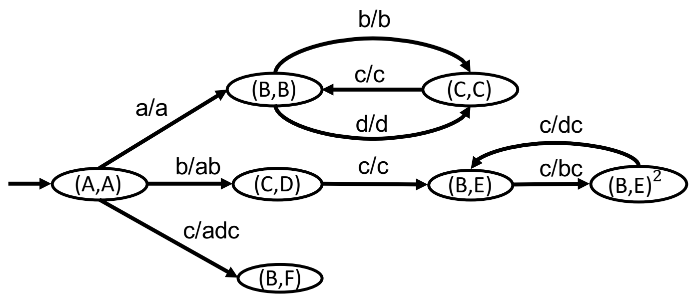

Figure IV.6: An insertion function that solves Problem IV.3.1

IV.6 Bounded Cost Rate Insertion Strategies

In the last section, we have solved the opacity enforcement problem so that the system's energy level at every dimension never drops below 0. Since event insertion always costs energy, it is beneficial to explore an economical way of insertion for practical purposes. Motivated by this requirement, we propose the concept of *bounded cost rate insertion strategies* and investigate their synthesis in this section.

IV.6.1 Motivation and Problem Formulation

The structure EIS_w obtained in the last section usually contains more that one insertion strategies that solve Problem IV.3.1. Generally, there exist cycles in the original system thus insertion functions may need to insert fictitious events infinitely often to enforce opacity, in which case event insertion consumes an infinite amount of energy. From a practical point of view, it is desirable to require that the insertion function's long run rate of energy consumption be bounded so that the designer may control the energy consumed per insertion step.

To facilitate our discussion, we proceed as before and merge each leaf state of EIS_w with the state subsumed by it, resulting in EIS_m. As was discussed earlier, the same decision is made at the leaf state and at the state subsumed by it; also, the same game starts from the leaf states as from the subsumed states. Thus we are able to discuss infinite-duration games on EIS_m.

To explore the rate of insertion cost, we first define $V_c : Run(EIS_m) \to (\mathbb{Z} \setminus \mathbb{N})^k$ as the accumulative insertion cost function for runs in EIS_m. Given $r_m = y_0^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$, $V_c(r_m) = \sum_{i=1}^{n} \omega_{in}(\theta_i)$. We also define $V_{mc} : Run_{inf}(EIS_m) \to \mathbb{R}^k$ as the limit mean insertion weight function for infinite runs in EIS_m. Given $r_m = y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \xrightarrow{\theta_2} \cdots$, $V_{mc}(r_m) = \liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \omega_{in}(\theta_i)$. Then we propose the *bounded cost rate insertion strategy synthesis problem.*

Problem IV.6.1. *Synthesize a bounded cost rate insertion strategy* π_{in} *such that for any infinite initial run* $r_m \in Run_{inf}(\pi_{in}, y_0^e)$, $-V_{mc}(r_m) \le v_b$ *for some threshold vector* $v_b \in \mathbb{N}^k$.

Intuitively, we require the long run average of insertion cost be below a threshold under bounded rate cost insertion strategies, so that the rate of insertion cost does not blow up. This problem is discussed on EIS_m and is meaningful when the original system G is cyclic, i.e., there are infinite runs in G and the EIS_m. Problem IV.6.1 can be viewed as a *multidimensional mean payoff game* [33] between the insertion function and the environment. Specifically, the insertion function tries to maintain multidimensional mean payoff vectors bounded by a given threshold v_b while the antagonistic environment tires to spoil the goal. Furthermore, this game is with complete information as inserted events and insertion cost are known to both players. Due to this fact, we may ignore the state information but only focus on weights associated with f_{zy} transitions in EIS_m.

We add a minus sign on both sides of the inequality in Problem IV.6.1 and obtain $\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \omega_{in}(\theta_i) \ge -v_b$. Equivalently, we may show whether $\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (\omega_{in}(\theta_i) + v_b) \ge \overrightarrow{0}$ holds. Hence, we can add v_b to each insertion weight vector in EIS_m and discuss the equivalent mean payoff objective. For simplicity, we still denote the structure by EIS_m and will determine whether the limit mean insertion cost is above 0 in the game graph. We further let $W = \max\{-\omega_{in}^{(i)}(\theta) : \exists z^e \in Q_Z^E, \theta \in E_o^*, \text{ s.t. } f_{zy}^E(z^e, \theta)!, 1 \le i \le k\}$ be the maximal absolute value of elements in insertion weight functions defined in EIS_m. Obviously, W is a positive integer.

IV.6.2 Hyperplane Separation Technique

A multidimensional mean payoff game is more challenging to solve than a one-dimensional game since the objectives in different dimensions may be in conflict. In this section, we apply a recently-proposed method called *hyperplane separation technique* from [34] to solve Problem IV.6.1. Originally, this technique was developed for general multidimensional mean payoff games. The main idea is to reduce the multidimensional mean payoff game in Problem IV.6.1 to a one-dimensional mean payoff game on the same graph and then solve it. It can be further shown that there is close relation between winning regions of both players in the original game and the induced game.

Since the algebraic mean of a set of vectors can always be expressed as a *convex combination* of those vectors, we have the following observation: if there exists a convex combination of the cost vectors such that some dimensions remain negative, then there exists a strategy for the environment to spoil the goal of the insertion function in Problem IV.6.1. Intuitively, we are going to "separate" the convex combinations leading to each player to win the game. From results in geometry, a hyperplane may also be used to separate vectors in a linear space.

In a linear space, a vector v lies *above* a hyperplane $\mathcal{H}$ with normal vector λ if $v^T \cdot \lambda \geq 0$; otherwise, it lies *below* $\mathcal{H}$; see, e.g., [13]. Furthermore, if the mean payoff vector resulted from a game lies below a hyperplane containing the origin, then it has at least one negative element. Therefore, if such a hyperplane exists, then the insertion function fails to enforce its multidimensional mean payoff objective and loses the game. On the other hand, if the insertion function is able to achieve mean payoff vectors that lie above all possible hyperplanes, then it can ensure its objective and win the game.

Given a k-dimensional insertion weight vector $\omega_{in}(\theta)$ for some insertion decision θ and a vector $\lambda \in \mathbb{R}^k$, we denote by $\omega_{in}(\theta)^T \cdot \lambda$ the inner product between $\omega_{in}(\theta)$ and λ. With a slight abuse of notation, we also use $\omega_{in}^T \cdot \lambda$ when there is no need to specify the insertion decision θ.

Then we assign $\omega_{in}^T \cdot \lambda$ to the edge labeled with insertion weight function ω_{in} in EIS_m and transfer a game with multidimensional objective to one with one-dimensional objective. From the above discussion, the insertion function achieves a mean payoff vector that lies above $\mathcal{H}$ or a mean payoff vector with all nonnegative elements if and only if it ensures that the one-dimensional mean payoff objective remains nonnegative, with weight function $\omega_{in}^T \cdot \lambda$ in EIS_m. Therefore, our goal is to search for such hyperplanes, which transfers the problem of solving a multidimensional mean payoff game to one of finding a proper normal vector in the k-dimensional integer space.

IV.6.3 Synthesize Bounded Cost Rate Insertion Strategies

In this section, we present several results to establish the relation between the original multidimensional mean payoff game and the induced one-dimensional mean payoff game after applying the hyperplane separation technique. Based on them, we then derive solutions to Problem IV.6.1.

Denote by Win_{em} (respectively Win_{im}) the winning region of the environment (respectively the insertion function) in the multidimensional mean payoff game with weight function ω_{in}; further denote by Win_{em}^{λ} (respectively Win_{im}^{λ}) the winning region of the environment (respectively the insertion function) in the one-dimensional mean payoff game with weight function $\omega_{in}^T \cdot \lambda$. From now on, we focus on the environment's winning strategies. Since a mean payoff game under complete information is *determined* [43], i.e., from any vertex in the game graph, exactly one player has a winning strategy, we may directly obtain the insertion function's winning strategies afterwards.

Given a vector $\lambda \in \mathbb{R}^k$, we do the inner product between λ and each insertion weight vector in EIS_m to obtain a game with scalar insertion weights, while we do not consider the weights associated with event occurrence anymore. In the new game, we hope to achieve a nonnegative mean payoff objective. We repeat Lemma 1 and Lemma 2 in work [34] here, which establish the relation between the winning regions for both players in the original game and the new game.

- For every $\lambda \in \mathbb{R}^k$, we have that $Win_{em}^{\lambda} \subseteq Win_{em}$; also if $Win_{em}^{\lambda} \neq \emptyset$, then $Win_{em} \neq \emptyset$.
- If for all $\lambda \in \mathbb{R}^k$ we have that $Win_{em}^{\lambda} = \emptyset$, then $Win_{em} = \emptyset$

These results illustrate a potential way to determine whether the environment player has a nonempty winning region in the multidimensional mean payoff game: we just need to check all $\lambda \in \mathbb{R}^k$

to determine whether the environment wins the one-dimensional mean payoff game with weight function $\omega_{in}^T \cdot \lambda$. The readers are referred to [34] for detailed proofs.

Therefore, the key point is to search for a hyperplane and then determine the winner of the induced one-dimensional mean payoff game. However, it seems that we need to check infinitely many vectors in $\mathbb{R}^k$, which is not feasible in practice. Fortunately, by Lemma 3 in [34], we only need to check a finite number of vectors in a k-dimensional space. Let $M = (k \cdot n \cdot W)^{k+1}$, where W is the maximal absolute value in insertion weight functions defined in EIS_m, n is the number of states in EIS_m, and k is the number of dimensions. For a positive integer i, we denote by $Z_i^{\pm} = \{j \in \mathbb{Z} : -i \leq j \leq i\}$ (resp. $Z_i^{+} = \{j \in \mathbb{N} : 1 \leq j \leq i\}$) the set of integers (positive integers) from $-i$ to i (resp. from 1 to i). The lemma is stated here while its proof is omitted, which can be found in [34].

- There exists $\lambda \in \mathbb{R}^k$ such that $Win_{em}^{\lambda} \neq \emptyset$ if and only if there exists $\lambda' \in (Z_M^{\pm})^k$ such that $Win_{em}^{\lambda'} \neq \emptyset$.

To summarize and strengthen the above results, we repeat Lemma 4 in [34] as a theorem here to show the key argument for solving Problem IV.6.1.

Theorem IV.6.1. *Given the multidimensional mean-payoff game on EIS_m, we have that:* (1) $\bigcup_{\lambda \in (Z_M^+)^k} Win_{em}^{\lambda} \subseteq Win_{em}$; (2) *if* $\bigcup_{\lambda \in (Z_M^+)^k} Win_{em}^{\lambda} = \emptyset$, *then* $Win_{em} = \emptyset$.

This theorem illustrates that if the environment wins the one-dimensional mean payoff game with weight vector $\omega_{in}^T \cdot \lambda$ at a certain state in EIS_m for some $\lambda \in (Z_M^+)^k$, then it also has a way to beat the insertion function and win the multidimensional mean payoff game from the same state; conversely, if the insertion function wins any one-dimensional mean payoff game with weight vector $\omega_{in}^T \cdot \lambda$ where $\lambda \in (Z_M^+)^k$ at a state in EIS_m, then the insertion function also wins the original multidimensional game from that state. This theorem suggests that we can restrict attention to vectors in $(Z_M^+)^k$ and determine which player wins the transformed one-dimensional game. More details concerning the proof of the theorem can be found in [34].

Based on the above results, we present Algorithm IV.3 to solve Problem IV.6.1. In the algorithm, we assume that each state in EIS_m is numbered from 1 to n. At each state in EIS_m, we sequentially iterate over vector $\lambda \in (Z_M^+)^k$ to see if there exists a winning strategy for the environment with weight function $\omega_{in}^T \cdot \lambda$ by the pseudo-polynomial algorithm proposed in [17] for mean payoff games. Then we define the *attractor* for each player in EIS_m. Let Q be a set of states in EIS_m, then for the environment ("em" for short), $Attr_{em}(Q)$ is defined recursively as follows: $Q_0 = Q$, $Q_{j+1} = Q_j \cup \{y^e \in Q_Y^E : \exists z^e \in Q_j, e_o \in E_o \ s.t.\ f_{yz}^E(y^e, e_o) = z^e\} \cup \{z^e \in Q_Z^E : \forall y^e \in Q_Y^E : [\exists \theta \in E_o^*, \text{ s.t. } f_{zy}^E(z^e, \theta) = y^e] \Rightarrow [y^e \in Q_j]\}$ and $Attr_{em}(Q) = \bigcup_{j \geq 0} Q_j$. Similarly, we define the attractor for the insertion function. Intuitively, the environment ensures to reach Q_i from Q_{i+1} within one transition regardless of the insertion function's strategies. Therefore, the environment may reach states in Q from states in $Attr_{em}(Q)$ within a finite number of transitions regardless of the insertion function's strategies. On the other hand, the environment may avoid reaching Q if it is at states outside of $Attr_{em}(Q)$.

Algorithm IV.3: Find solutions to Problem IV.6.1

Input : EIS_m
Output: Insertion strategies solving Problem IV.6.1

for $j = 1 : n$ **do**
  if *q_j is still in the remaining structure* **then**
    Consider $q_j \in Q_Y^E \cup Q_Z^E$ in EIS_m;
    for $\lambda \in (Z_M^+)^k$ **do**
      if *there exists an environment's winning strategy from q_j to achieve a negative mean payoff in the transformed one-dimensional game with weight function $\omega_{in}^T \cdot \lambda$ by the method in Section 5 of [17]* **then**
        Remove $Attr_{em}(\{q_j\})$ from EIS_m;
if *the remaining structure is not empty* **then**
  Return insertion strategies in the structure;
else
  No solution exists for Problem IV.6.1.

In Algorithm IV.3, we apply the method in [17] to solve the induced one-dimensional mean payoff game and this method outperforms any other known method in terms of complexity. If at the current state in EIS_m, there exists a winning strategy for the environment for the one-dimensional mean-payoff objective with weight function $\omega_{in}^T \cdot \lambda$, then we remove the attractor of the current state and proceed to the next iteration. The reason is that if the environment wins the mean payoff game from a vertex in the game graph, it also wins the game from the attractor of the current vertex.[1] Thus the game graph may be shrinking when the algorithm is running. However, if the environment is unable to win the one-dimensional game for any $\lambda \in (Z_M^+)^k$ at the current state, i.e., the insertion function has a winning strategy to enforce a nonnegative mean payoff from the current state for all $\lambda \in (Z_M^+)^k$, then the insertion function may enforce a mean payoff vector with all nonnegative elements. Thus this state should be included in the winning region of the insertion function for the multidimensional mean payoff game. Therefore, after all states in EIS_m are checked, the insertion function has winning strategies for Problem IV.6.1 against all environment's strategies if the remaining structure is not empty. Otherwise, no solution exists for Problem IV.6.1 if all states of EIS_m are removed. Besides, as positional strategies suffice to win a mean payoff game with perfect information [43], we simply let strategies returned by Algorithm IV.3 be positional so that a finite number of strategies are returned. The correctness of Algorithm IV.3 is from Theorem IV.6.1 and more details concerning solving a one-dimensional mean payoff game are available in [17].

Finally, we briefly discuss the complexity of Algorithm IV.3 following a similar argument as in [34]. When running the algorithm, we need n iterations under the worst case and in each iteration we solve at most M^k one-dimensional mean payoff games. Thus the iterative algorithm needs to solve $O(n \cdot M^k)$ one-dimensional mean payoff games with m edges, n vertexes, and the maximal weight being at most $k \cdot W \cdot M$ (as the maximum element in all $\lambda \in (Z_M^+)^k$ is M, the maximum weight in every dimension of ω_{in} is W, and we sum k dimensions). Since one-dimensional mean payoff games with n vertexes, m edges and maximal weight W can be solved in time $O(n \cdot m \cdot W)$ by the method proposed in [17], the overall complexity of the algorithm is $O(n^2 \cdot m \cdot k \cdot W \cdot (k \cdot n \cdot W)^{k^2+2k+1})$, which is polynomial in terms of the number of vertexes when k is fixed.

Example IV.6.1. *We revisit Example IV.5.1 and further discuss Problem IV.6.1 based on the solutions of Problem IV.3.1. We show EIS_m in Figure IV.7 after merging the leaf states with states subsumed by them in EIS_w. Then we investigate the bound of insertion cost rate by starting with*

[1] The pruning here is similar to calculating the supremal controllable sublanguage [23] by viewing the environment's winning states as undesirable, f_{yz}^E transitions as uncontrollable, f_{zy}^E transitions as controllable, and Y-states as marked.

threshold $v_b = [3,3]^T$ and see if Problem IV.6.1 has a solution. It is seen that EIS_m contains cyclic runs and this problem is discussed on them. We add v_b to each each insertion cost vector in EIS_m to obtain the new weight vectors $\omega_{in}(b) + v_b = [2,0]^T$, $\omega_{in}(d) + v_b = [0,2]^T$, $\omega_{in}(\epsilon) + v_b = [3,3]^T$ and those events are inserted in cyclic runs. After running Algorithm IV.3, we find that there exist insertion strategies solving Problem IV.6.1. The detailed process is tedious and is omitted here. For example, one feasible insertion strategy is to choose to insert b at Z-state z_2^e. Then it is easy to see that this strategy achieves a positive mean payoff value.

However, if we change the threshold vector to $v_b' = [1,1]^T$, then Problem IV.6.1 has no solution. From Figure IV.7, we see that two simple cycles $y_2^e \xrightarrow{c} z_2^e \xrightarrow{\{b\}} y_3^e \xrightarrow{c} z_3^e \xrightarrow{\{d\}} y_2^e$ and $y_2^e \xrightarrow{c} z_2^e \xrightarrow{\{d\}} y_6^e \xrightarrow{c} z_4^e \xrightarrow{\{b\}} y_2^e$ both have weight vector $\omega_{in}(b) + \omega_{in}(d) = [-4,-4]^T$. Since $\frac{-\omega_{in}(b) - \omega_{in}(d)}{2} = [2,2]^T > v_b$, no insertion strategy is able to enforce the mean payoff threshold $[1,1]^T$.

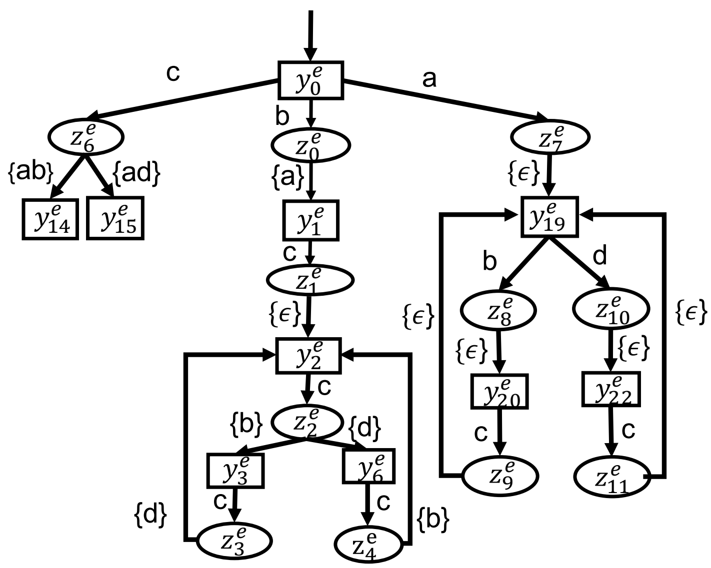

Figure IV.7: EIS_m after merging states

IV.7 Conclusion

This chapter investigated opacity enforcement by insertion functions under multiple energy constraints. To the best of our knowledge, it is the first to investigate opacity enforcement under such quantitative constraints. The system is initialized with certain types of energy and the energy levels change dynamically with event insertion and execution. Our goal is to synthesize an insertion function that enforces opacity as well as ensures that the system's energy level in every dimension is never below zero. A bipartite information structure called Energy Insertion Structure (EIS) was defined to characterize the game between the insertion function and the environment. The insertion function's winning strategies in EIS provably solve the opacity enforcement problem while if no winning insertion strategy exists in EIS, no solution to the problem exists. Thus EIS provides a sound and complete characterization of the solution space. Based on these solutions, we subsequently consider the rate of insertion cost and proposed the bounded cost rate insertion strategy synthesis problem, which is formulated as a multidimensional mean payoff game. A method called hyperplane separation technique was applied to reduce the multidimensional game to a one-dimensional game on the same graph. Additional analysis showed that by solving the induced games, we obtain valid solutions for the original problem.

CHAPTER V

Optimal Mean Payoff Supervisory Control under Partial Observation

V.1 Introduction

In this chapter, we formulate two types of optimal mean payoff supervisory control problems under partial observation and solve them in sequence. The first goal for the supervisor under both scenarios is to ensure that the energy level of the system is never below 0 or that the limit rate of energy level change is above a certain threshold. Then the concept of *energy information states* is proposed, which incorporate necessary information about the current states and the energy level of the system. After that we transfer each of the above problems into a two-player safety game [4] between the supervisor and the "environment" (aka system) on a finite information structure. The structure is called First Cycle Energy Inclusive Controller (FCEIC). By construction, we show that the winning strategies of the supervisor in the FCEIC correspond to potential solutions of the proposed problems: they ensure that the nonnegative energy level or the mean payoff threshold condition is satisfied. It turns out that the corresponding FCEICs only bear slight differences under the two problems, which is why we treat them concurrently. In the second phase, starting from the preceding respective solutions, we find optimal control strategies for the long run operation of the system, by searching over the winning regions in the FCEICs.

Our solution methodology is also inspired by graph games in quantitative reactive synthesis, especially mean payoff games [43]. A mean payoff game is an infinite-duration turn based two-player game on a weighted graph. The two players take turns to play by selecting an outgoing edge at their positions, resulting in an infinite path. The goal of the first player is to maximize the average payoffs (weights) of traversed edges while the goal of the second player is to minimize them, thus the game is *zero sum*. Well structured solutions were proposed for mean payoff games with *perfect information* [43,135], where both players know the complete history of the game up to their current positions. What is more challenging is the case of mean payoff game with *imperfect information* where one player is absent from the complete history of its opponent. Such games are in general undecidable [40] while some decidable classes were presented in [50], which motivated our assumptions on the system in this chapter. From the results in [2], the winning strategies for both players in mean payoff games may be derived by focusing on the first simple cycle that appears infinitely often. This inspired us to propose the concept of FCEIC.

In contrast with reactive synthesis, there is usually a *plant*, i.e., a system to be controlled, in supervisory control theory. Besides, the supervised system is *closed-loop* in the sense that the "input" to the supervisor is the set of strings generated by the system so far and the "output" of the supervisor is a control decision to inform the system what events are allowed to occur. Furthermore, the supervisor may allow multiple events to occur simultaneously, in which case it is the system that decides what event to execute next. This mechanism is similar to the so-called *multi-strategy* in game theory [4], under which one player may choose more than one outgoing edges at a position. In general, the supervisor may only have limited control and observation capabilities, i.e., some events of the system can never be disabled and some events are not observed by the supervisor. Those limitations are usually not characterized in games for reactive synthesis. Besides, the designed supervisor in this chapter should satisfy logical and quantitative requirements simultaneously. The above mentioned differences impose additional difficulties on directly applying existing results of quantitative graph games to solve a supervisory control problem. Thus special analysis is necessary to "bridge the gap" [42] and the established methods in two-player games may also need to be adjusted for our specific problem.

The structure FCEIC is proposed so that the analysis of two-player quantitative games may be performed in the presence of a plant to fit in the framework of supervisory control. The FCEIC is similar to the concept of *Kripke structure* in [5]. Notice that our work is not the first one to solve supervisory control problems by leveraging results from graph games in reactive synthesis, see, e.g., [90, 91, 125, 126]. However, both [125] and [126] focused on qualitative synthesis problems by control while [91] solved a mean payoff optimization problem with full observation supervisors. In contrast to the problem studied in this chapter as well, [90] discussed supervisory control under another game framework, namely fixed-initial-credit energy games under partial observation. Our work is also inspired by the work in Chapter IV which solved opacity enforcement by insertion functions under quantitative constraints and transformed the problem to a game with some different quantitative objective.

The following sections are organized as follows. Section V.2 describes the system model. In Section V.3, we formulate two types of optimal mean payoff supervisory control problems under partial observation. Section V.4 introduces energy information states and the First Cycle Energy Inclusive Controller (FCEIC) for each problem. Section V.5 analyzes relevant properties of the FCEIC, then partially solves the two proposed problems. Winning control strategies in the FCEIC ensure that the energy level of the system is always nonnegative or that the mean payoff is always above some threshold, corresponding to the two formulated problems. Section V.6 completely solves the two problems by finding the optimal solution from the partial solutions obtained in Section V.5. Finally, Section V.7 concludes the chapter and raises directions for future work.

V.2 System model

We consider supervisory control in the same system model by weighted finite-state automata as in Chapter IV:

$$G = (X, E, f, x_0, \omega)$$

where X is the finite state space, E is the finite set of events, $f : X \times E \to X$ is the partial transition function, $x_0 \in X$ is the initial state, $\omega : E \to \mathbb{Z}$ is the weight function that assigns an integer to each event. We view the event's weight as its *energy payoff* in this chapter. A positive number stands for energy gain while a negative number stands for energy cost. The transition function is extended to $X \times E^*$ in the standard manner and we still denote the extended function by f. The language generated by G is defined as $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined". We denote by $s \leq u$ if string s is a prefix of u, and $s < u$ if $s \leq u, s \neq u$. The function ω is additive and its domain can be extended to E^* by letting $\omega(\epsilon) = 0$, $\omega(se_o) = \omega(s) + \omega(e_o)$ for $s \in E^*$ and $e \in E$. Given $s \in \mathcal{L}(G)$, the *(accumulative) payoff* of s is the sum of each event's weight in s, i.e. $\omega(s)$. The system also has $v_0 \in \mathbb{N}$ as its *initial energy*.

In this chapter, we assume that the safety property is already satisfied and we do not consider the non-blockingness property either, thus no marked states are included in the system model. Instead, we discuss the (weak) *liveness* property: a system G is live if its generated language $\mathcal{L}(G)$ is live, i.e., $\forall s \in \mathcal{L}(G)$, $\exists u \in E$, s.t. $su \in \mathcal{L}(G)$. That is, there is a transition defined at each state in G, which thus never terminates. The liveness requirement on G is without loss of generality since it can be relaxed by adding observable self-loops at terminal states where no active events are defined, as was done in [103]. Overall, we can think of the given G as a controlled system that satisfies the original safety and non-blockingness requirements.

Given an automaton G, for $x_1, x_2 \in X$ and $e \in E$, we denote by $x_1 \xrightarrow{e} x_2$ if $f(x_1, e) = x_2$. A *run* in G is a sequence of states and events: $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$ and it may be infinitely long. We denote the set of runs in G by $Run(G)$. A run is *initial* if its initial state is the initial state of the system. We say that a run forms a *cycle* if $x_1 = x_n$ and a cycle is *simple* if $\forall i, j \in \{1, 2, \cdots n-1\}$, $i \neq j \Rightarrow x_i \neq x_j$. If r is a cycle, there is a corresponding (string) loop $e_1 e_2 \cdots e_{n-1}$ starting from and ending in x_1. The loop is called *simple* if the cycle is simple.

For a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_{n+1}$, its (accumulative) payoff is $\sum_{i=1}^{n} \omega(e_i)$ and its mean payoff is $\frac{1}{n} \sum_{i=1}^{n} \omega(e_i)$. We also define the system's *energy level* for a run as $V : Run(G) \to \mathbb{Z}$ where

$V(r) = v_0 + \sum_{i=1}^{n} \omega(e_i)$. So the energy level changes dynamically with the occurrence of events.

Furthermore, we let $Run_{inf}(G)$ be the set of infinite runs in automaton G. Then we define $V_{lim} : Run_{inf}(G) \to \mathbb{R}$ as the *limit mean payoff* of an infinite run. For a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots$,

$$V_{lim}(r) = \liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \omega(e_i)$$

Here we take the infimum of the sequence $\{\frac{1}{n} \sum_{i=1}^{n} \omega(e_i)\}$ so that its limit always exists. Notice that the limit mean payoff of a run only depends on the mean payoff of cycles that are traversed infinitely often in the run. For example, if $x_i \xrightarrow{e_i} x_{i+1} \xrightarrow{e_{i+1}} \cdots \xrightarrow{e_j} x_{j+1}$ is the only cycle that appears infinitely often in the run r, then

$$V_{lim}(r) = \frac{1}{j-i+1} \sum_{l=i}^{j} \omega(e_l)$$

In other words, the limit mean payoff is *independent* of finite prefixes of a run.

The system is controlled by a *supervisor* [23] that dynamically enables/disables events of the system so that some specification is achieved. The event set E is partitioned as $E = E_c \cup E_{uc}$, where E_c is the set of controllable events and E_{uc} is the set of uncontrollable events. A control decision $\gamma \in 2^E$ by the supervisor is *admissible* if $E_{uc} \subseteq \gamma$, i.e., the supervisor never disables uncontrollable events. We define $\Gamma = \{\gamma \in 2^E : E_{uc} \subseteq \gamma\}$ as the set of admissible control decisions. The system is also partially observable and E is partitioned as $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events. Given a string $t = t'e \in E^*$, its natural projection $P : E^* \to E_o^*$ is recursively defined as $P(t) = P(t'e) = P(t')P(e)$ where $t' \in E^*$ and $e \in E$. The projection of an event is $P(e) = e$ if $e \in E_o$ and $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string.

A supervisor is a function $S : P[\mathcal{L}(G)] \to \Gamma$ and we denote by $\mathbb{S}$ the set of supervisors. A partial observation supervisor makes decisions only based on the projected behavior of the system. We use S/G to represent the controlled system under S. Accordingly, we denote by $\mathcal{L}(S/G)$ the language generated in S/G and $Run(S/G)$ the set of runs in S/G, respectively.

Next, we define some operators in G. Given a set of states $q \subseteq X$, the *unobservable reach*, denoted by $UR(q)$, is defined as: $UR(q) = \{x' \in X : \exists x \in q, s \in E_{uo}^*, \text{ s.t. } f(x,s) = x'\}$. Specifically, the unobservable reach under a set of events $\gamma \subseteq E$, denoted by $UR_\gamma(q)$, is defined as: $UR_\gamma(q) = \{x' \in X : \exists x \in q, s \in (E_{uo} \cap \gamma)^*, \text{ s.t. } f(x,s) = x'\}$. Besides, the *observable reach* under event $e_o \in E_o$, denoted by $Next_{e_o}(q)$, is defined as: $Next_{e_o}(q) = \{x' \in X : \exists x \in q \text{ s.t. } f(x,e_o) = x'\}$.

The *observer* of G is defined as: $Obs(G) = (X_{obs}, E_o, \delta, x_{obs,0})$ where $X_{obs} \subseteq 2^X$ is the state space; $x_{obs,0} = UR(\{x_0\})$ is the initial state and δ is the transition function where $\forall x_{obs} \in X_{obs}$, $\forall e_o \in E_o$: $\delta(x_{obs}, e_o) = UR(Next_{e_o}(x_{obs}))$. The weight function is omitted here in the definition. An observer state is termed a *(current) state estimate* of the system.

V.3 Problem Formulations

In this section, we formulate two optimal mean payoff supervisory control problems, i.e., with and without the constraint of nonnegative energy level. Before stating them, we first assume that there are no unobservable loops in $\mathcal{L}(G)$ and we keep this assumption in the following discussion.

Assumption V.3.1. *Given an automaton G, $\forall x \in X$, $\forall s \in E^* \setminus \{\epsilon\}$, $[f(x,s) = x] \Rightarrow [P(s) \neq \epsilon]$.*

We first formulate the *constrained optimal mean payoff supervisory control problem* by considering both qualitative and quantitative objectives. The supervised system should be live, thus never terminates. Besides, the limit rate of energy generation is optimized even if the system operates in the most adversarial condition, provided that the energy level of the system never drops below 0.

Problem V.3.1 (Constrained Optimal Mean Payoff Supervisory Control Problem)**.** *Given system G with initial energy $v_0 \in \mathbb{N}$, design a supervisor $S^* \in \mathbb{S}$ such that: (i) $\mathcal{L}(S^*/G)$ is live; (ii) $\forall r \in Run(S^*/G)$: $V(r) \geq 0$; (iii) $\inf_{r \in Run_{inf}(S^*/G)} V_{lim}(r) = \sup_{S \in \mathbb{S}} \inf_{r \in Run_{inf}(S/G)} V_{lim}(r)$.*

The problem statement says that the supervised system satisfies the following conditions: (i) it is live; (ii) its energy level for any run is nonnegative; (iii) its worst case limit mean payoff is maximized.

As a slight variant of the above problem, we also formulate the *unconstrained optimal mean payoff supervisory control problem*, which ignores the nonnegative energy level constraint in Problem V.3.1. We make an extra assumption to restrict the system in the unconstrained optimal control problem.

In the observer of the system, given a state $x_{obs} \in X_{obs}$, let $Loop(x_{obs}) = \{l \in E_o^* \setminus \{\epsilon\} : \delta(x_{obs}, l) = x_{obs}$ and $\forall l' < l$ s.t. $l' \neq \epsilon, \delta(x_{obs}, l') \neq x_{obs}\}$ be the set of simple loops starting from x_{obs}. Also, given string $l \in Loop(x_{obs})$, we let $SimLp(x_{obs}, l) = \{t \in E^* \setminus \{\epsilon\} : \exists x \in X_{obs}$ s.t. $f(x,t) = x, P(t) = l$ and $\forall t' < t, f(x,t') \neq x\}$ be the set of non-ϵ simple loops with the same projection l and starting from some state in x_{obs}.

Assumption V.3.2. *Given automaton G and its observer $Obs(G)$, $\forall x_{obs} \in X_{obs}$, $\forall l \in Loop(x_{obs})$, and $\forall s, s' \in SimLp(x_{obs}, l)$, we have either $\omega(s) < 0 \Rightarrow \omega(s') < 0$ or $\omega(s) \geq 0 \Rightarrow \omega(s') \geq 0$.*

That is to say, for two simple loops with the same projection, their payoffs should have the same sign. This assumption is inspired by the decidable classes of mean payoff games with partial observation in [50]. Later on, we will see how this assumption helps us solve the unconstrained optimal mean payoff supervisory control problem. We say that a system is *with unambiguous cycle payoffs* if it satisfies Assumption V.3.2.

Example V.3.1. *Let the system G in Figure V.1 be with $E_{uo} = \{u_1, u_2\}$ and $E_o = \{o_1, o_2, o_3\}$. The weight of each event is shown in the figure. There are 4 simple cycles: $x_0 \xrightarrow{u_1} x_1 \xrightarrow{o_1} x_3 \xrightarrow{o_2} x_0$ with payoff 2, $x_0 \xrightarrow{u_2} x_2 \xrightarrow{o_1} x_4 \xrightarrow{o_2} x_0$ with payoff 1, $x_0 \xrightarrow{u_1} x_1 \xrightarrow{o_1} x_3 \xrightarrow{o_3} x_0$ with payoff -1 and $x_0 \xrightarrow{u_2} x_2 \xrightarrow{o_1} x_4 \xrightarrow{o_3} x_0$ with payoff -2. So G is with unambiguous cycle payoffs.*

Problem V.3.2 (Unconstrained Optimal Mean Payoff Supervisory Control Problem)**.** *Given system G with unambiguous cycle payoffs, initial energy $v_0 \in \mathbb{N}$ and threshold $v \in \mathbb{N}$, design a supervisor $S^* \in \mathbb{S}$ such that: (i) $\mathcal{L}(S^*/G)$ is live; (ii) $\forall r \in Run_{inf}(S^*/G)$: $V_{lim}(r) \geq v$; (iii) $\inf_{r \in Run_{inf}(S^*/G)} V_{lim}(r) = \sup_{S \in \mathbb{S}} \inf_{r \in Run_{inf}(S/G)} V_{lim}(r)$.*

Compared with Problem V.3.1, we also require that the supervised system be live and the worst case limit mean payoff be optimized. However, we omit the requirement of nonnegative energy

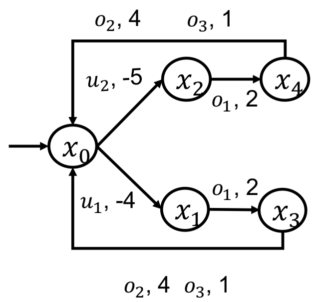

Figure V.1: An automaton with unambiguous cycle payoffs

level, instead, we are to achieve that the limit mean payoff (rate of energy gain) of any infinite run is above a given threshold v. Actually, given v, we may subtract v from the weight of each event and equivalently evaluate whether the limit mean payoff is above 0. Hence we will assume $v = 0$ in the following discussion without loss of generality.

Specifically, we call the first two conditions in Problem V.3.1 (respectively Problem V.3.2) as its *mean payoff decision problem*. In both Problem V.3.1 and Problem V.3.2, the optimal supervisor should maximize the worst case limit mean payoff. We may imagine that the supervisor is "playing a game" against an antagonistic opponent, where the supervisor is to maximize its mean payoff while its opponent is to prevent the supervisor. However, the two sides may have asymmetric information since the supervisor only has partial observation of the system. Thus it is essential to construct proper estimates for current states and the energy level of the system so that the supervisor may make decisions. In the following discussion, we solve Problem V.3.1 and Problem V.3.2 sequentially: we first find solutions to their corresponding mean payoff decision problems, then completely solve them by resolving the optimization issues.

V.4 First Cycle Energy Inclusive Controller

In this section, we define *Energy Information States* and then transfer both Problem V.3.1 and Problem V.3.2 to two-player games between the supervisor and the environment. We further propose the *First Cycle Energy Inclusive Controller* (FCEIC) as the game structure, which records the update of both current state estimates and the energy level of the system under control. The FCEIC is inspired by the Bipartite Transition System and All Enforcement Structure in [126] and [125], which include supervisors enforcing several logical properties.

V.4.1 Energy Information States

We define some orders for vectors. Given two vectors $v_1 = [v_1(1), v_1(2), \cdots, v_1(n)]$, $v_2 = [v_2(1), v_2(2), \cdots, v_2(n)] \in \mathbb{Z}^n$, we denote by $v_1 \le v_2$ (respectively $v_1 \ge v_2$) if $\forall 1 \le i \le n, v_1(i) \le v_2(i)$ (respectively $v_1(i) \ge v_2(i)$). We also denote by $v_1 < v_2$ if $\forall 1 \le i \le n, v_1(i) \le v_2(i)$ and $\exists 1 \le j \le n$, $v_1(j) < v_2(j)$ (respectively $\forall 1 \le i \le n$, $v_1(i) \ge v_2(i)$ and $\exists 1 \le j \le n$, $v_1(j) > v_2(j)$), i.e., at least one element in v_1 is strictly smaller (larger) than the element at the same position in v_2.

The partial observation of supervisors adds special difficulty to Problem V.3.1 and Problem V.3.2. We hope to transfer each problem into another problem, which is under full observation. Then our goal is to solve the transformed problems and show that by solving the new problems, we obtain solutions to the original problems. In order to track the unobservable reaches between states and the their payoffs, we define *energy information states* as follows. Here we let $|\cdot|$ be the cardinality of a set.

Definition V.4.1 (Energy Information States)**.** *Given system G, an energy information state is:* $q^e = (q, [v(1), \cdots v(|q|)]) \in 2^X \times (\cup_{k=1}^{|X|} \mathbb{Z}^k)$*. Let* $Est(q^e)$ *and* $Lev(q^e)$ *denote the state estimate and energy level components of* q^e*, respectively, hence,* $q^e = (Est(q^e), Lev(q^e))$*.*

Denote by Q^E the set of energy information states. Each $q^e \in Q^E$ induces a *belief function* $h_{q^e} : Est(q^e) \to \mathbb{Z}$. Specifically, for $q^e \in Q^E$ where $Est(q^e) = q \in 2^X$, $Lev(q^e) = \{h_{q^e}(x) : x \in q\}$. We usually put $Lev(q^e)$ in a vector form: $[h_{q^e}(x_1), \cdots h_{q^e}(x_{|q|})]$ and by convention in this work, elements in $Lev(q^e)$ are placed in an increasing order w.r.t. state names in $Est(q^e)$. An energy information state q^e is *energy safe* if $\forall x \in Est(q^e)$, $h_{q^e}(x) \ge 0$.

We define an order $\preccurlyeq$ over Q^E: for $q_1^e, q_2^e \in Q^E$, $q_1^e \preccurlyeq q_2^e$ if $Est(q_1^e) = Est(q_2^e)$ and $Lev(q_1^e) \le Lev(q_2^e)$. We also say that q_2^e *subsumes* q_1^e if $q_1^e \preccurlyeq q_2^e$. In other words, q_2^e shares the same state estimate with q_1^e and the energy level vector of q_2^e is no less than that of q_1^e in a point-wise sense. We define another order $\prec$ over Q^E: for $q_1^e, q_2^e \in Q^E$, $q_1^e \prec q_2^e$ if $Est(q_1^e) = Est(q_2^e)$, $Lev(q_1^e) < Lev(q_2^e)$. That is to say, q_1^e and q_2^e have the same state estimate and there exists $Lev(q_1^e)(i) < Lev(q_2^e)(i)$ at some state $Est(q_1^e)(i)$ for some $i \geq 1$.

By Dickson's lemma (see, e.g., [69]), "$\leq$" on k-dimensional nonnegative integer space $\mathbb{N}^k$ is a *well-quasi ordering* for any $k \in \mathbb{N}^+$. We further argue that $\preccurlyeq$ on energy safe energy information states is also a well-quasi ordering, i.e., for any infinite sequence of energy safe energy information states $q_1^e, q_2^e \cdots \in Q^E$, there exist two indexes $i < j$, such that $q_i^e \preccurlyeq q_j^e$.

We call $q^{ae} \in Q^E \times \Gamma$ an *augmented energy information state*, which augments an energy information state with a control decision. Let $I_E(q^{ae})$, $\Gamma(q^{ae})$ denote the energy information state component and control decision component of q^{ae}, respectively, so $q^{ae} = (I_E(q^{ae}), \Gamma(q^{ae}))$. With a slight abuse of notation, we also use $h_{q^{ae}}$ to stand for h_{q^e} where $q^e = I_E(q^{ae})$. An augmented energy information state q^{ae} is also called energy safe if $\forall x \in Est(I_E(q^{ae}))$, $h_{q^{ae}}(x) \geq 0$. Then we give the following two concepts.

For $\gamma \in \Gamma$, $q^{ae} \in Q^E \times \Gamma$ is a *γ-successor* of $q^e \in Q^E$ if: (i) $Est(I_E(q^{ae})) = UR_\gamma(Est(q^e))$; (ii) $\forall x' \in Est(q^{ae})$, $h_{q^{ae}}(x') = \min\limits_{\xi}\{h_{q^e}(x) + \omega(\xi) : \exists x \in Est(q^e), \xi \in (E_{uo} \cap \gamma)^* \text{ s.t. } f(x, \xi) = x'\}$. Overall, $q^{ae} = (I_E(q^{ae}), \gamma)$. Its state estimate component is the unobservable reach of $Est(q^e)$ under γ. We also use the belief function to track the *minimum* energy level by some unobservable string ξ reaching a possible state in $Est(I_E(q^{ae}))$.

For $e_o \in E_o$, $q^e \in Q^E$ is an *e_o-successor* of $q^{ae} \in Q^E \times \Gamma$ if: (i) $e_o \in \Gamma(q^{ae}) = \gamma$ and $Est(q^e) = Next_{e_o}(Est(I_E(q^{ae})))$; (ii) $\forall x \in Est(q^e)$, $h_{q^e}(x) = \min\limits_{x'}\{h_{q^{ae}}(x') + \omega(e_o) : \exists x' \in Est(I_E(q^{ae})) \text{ s.t. } f(x', e_o) = x\}$. So the state estimate component of q^e is the observable reach of $Est(I_E(q^{ae}))$ under e_o. Meanwhile, we use the belief function to track the *minimum* energy level by observable event e_o reaching a possible state in $Est(q^{ae})$.

A *control-observation sequence* is a sequence of states, events and control decisions in the form of $\rho = y_1^e \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} z_2^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y_n^e$ or $\rho' = y_1^e \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} z_2^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y_n^e \xrightarrow{\gamma_n} z_n^e$ where $\forall i \leq n$, $\gamma_i \in \Gamma$, $e_i \in E_o$, $y_i^e \in Q^E$, $z_i^e \in Q^E \times \Gamma$, z_i^e is a γ_i-successor of y_i^e and y_{i+1}^e is an e_i-successor of z_i^e. Such a sequence characterizes the update of state estimate and energy level under control decisions. By convention, we let $\rho_k = y_1^e \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} z_2^e \cdots \xrightarrow{\gamma_{k-1}} z_{k-1}^e \xrightarrow{e_{k-1}} y_k^e$ and $\rho'_k = y_1^e \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} z_2^e \cdots \xrightarrow{\gamma_{k-1}} z_{k-1}^e \xrightarrow{e_{k-1}} y_k^e \xrightarrow{\gamma_k} z_k^e$, for $1 \leq k \leq n$. With the supervisor making decisions, strings are generated in the supervised system.

Definition V.4.2 (Strings generated by Control-Observation Sequence)**.** *Given a control-observation sequence ρ or ρ', the set of strings generated by ρ is defined recursively as: $\forall 1 \leq k \leq n$, let $Str(\rho_1) = \{\epsilon\}$, $Str(\rho'_1) = \{\xi_1 \in E_{uo}^* : \exists x \in Est(y_1^e), x' \in Est(I_E(z_1^e)), \xi_1 \in (\gamma_1 \cap E_{uo})^* \text{ s.t. } f(x,\xi_1) = x'\}$, then $Str(\rho_{k+1}) = \{s'_k e_k : \exists x \in Est(y_1^e), x' \in Est(I_E(z_k^e)), x'' \in Est(y_{k+1}^e), s'_k \in Str(\rho'_k), \text{ s.t. } f(x, s'_k) = x', f(x', e_k) = x''\}$ and $Str(\rho'_{k+1}) = \{s_{k+1}\xi_{k+1} : \exists x \in Est(y_1^e), x' \in Est(y_{k+1}^e), x'' \in Est(I_E(z_{k+1}^e)), s_{k+1} \in Str(\rho_{k+1}), \xi_{k+1} \in (\gamma_{k+1} \cap E_{uo})^*, \text{ s.t. } f(x, s_{k+1}) = x', f(x', \xi_{k+1}) = x''\}$.*

Then we show that in an energy or augmented energy information state, belief functions always return the *minimum* payoff of strings reaching a state in the state estimate.

Theorem V.4.1. *For a control-observation sequence ρ or ρ', we have that $\forall x \in Est(y_n^e)$, $h_{y_n^e}(x) = \min_{s \in Str(\rho)} \{\omega(s) : \exists \tilde{x} \in Est(y_1^e), \text{ s.t. } f(\tilde{x}, s) = x\}$ and $\forall x' \in Est(I_E(z_n^e))$, $h_{z_n^e}(x') = \min_{s \in Str(\rho')} \{\omega(s) : \exists \tilde{x} \in Est(y_1^e), \text{ s.t. } f(\tilde{x}, s) = x'\}$.*

Proof. Prove by induction on the length of observable string $t = e_1 \cdots e_{n-1}$ $(n \in \mathbb{N}^+)$ where $|t| = n-1$. The length of t reflects the length of the sequence. We also use the notations ρ_k and ρ'_k in the following discussion.

Induction Basis: $n = 1$ and consider y_1^e or $y_1^e \xrightarrow{\gamma_1} z_1^e$. The result obviously holds for single state y_0^e and also holds for $y_1^e \xrightarrow{\gamma_1} z_1^e$ by Definition V.4.2 and the definition of γ-successor.

Inductive Hypothesis: we assume the lemma holds when $n = k$, i.e., for ρ_k and ρ'_k.

Induction Step: when $n = k+1$, consider ρ_{k+1} and ρ'_{k+1}. First, y_{k+1}^e is an e_k-successor or z_k^e. Let $Est(I_E(z_k^e)) = q'_k$ and $Est(y_{k+1}^e) = q_{k+1}$, then $\forall x \in q_{k+1}$, $h_{y_{k+1}^e}(x) = \min_{x'}\{h_{z_k^e}(x') + \omega(e_k) : \exists x' \in q'_k, \text{ s.t. } f(x', e_k) = x\}$. By the inductive hypothesis and Definition V.4.2, $h_{y_{k+1}^e}(x) = \min_{x'} \min_{s'_k}\{\omega(s'_k) + \omega(e_k) : \exists \tilde{x} \in Est(y_1^e), s'_k \in Str(\rho'_k) \text{ s.t. } f(\tilde{x}, s'_k) = x'\} = \min_{s_{k+1}}\{\omega(s_{k+1}) : \exists \tilde{x} \in Est(y_1^e), s_{k+1} \in Str(\rho_{k+1}) \text{ s.t. } s_{k+1} = s'_k e_k, f(\tilde{x}, s_{k+1}) = x\}$.

Then z^e_{k+1} is a γ_{k+1}-successor of y^e_{k+1}. Let $Est(y^e_{k+1}) = q_{k+1}$ and $Est(I_E(z^e_{k+1})) = q'_{k+1}$, so $\forall x' \in q'_{k+1}$, $h_{z^e_{k+1}}(x') = \min_{\xi_{k+1}}\{h_{y^e_{k+1}}(x) + \omega(\xi_{k+1}) : \exists x \in q_{k+1}, \xi_{k+1} \in (E_{uo} \cap \gamma_{k+1})^* \text{ s.t. } f(x, \xi_{k+1}) = x'\}$. From what we just proved, $h_{z^e_{k+1}}(x') = \min_{\xi_{k+1}} \min_{s_{k+1}}\{\omega(s_{k+1}) + \omega(\xi_{k+1}) : \exists \tilde{x} \in Est(y^e_1), s_{k+1} \in Str(\rho_{k+1}) \text{ s.t. } f(\tilde{x}, s_{k+1}) = x, f(x, \xi_{k+1}) = x'\} = \min_{s'_{k+1}}\{\omega(s'_{k+1}) : \exists \tilde{x} \in Est(y^e_1), s'_{k+1} \in Str(\rho'_{k+1}) \text{ s.t. } s'_{k+1} = s_{k+1}\xi_{k+1}, f(\tilde{x}, s'_{k+1}) = x'\}$. Thus the result holds at $k+1$, completing the proof. $\square$

Since we always count the minimum string payoff when creating a new e_o-successor or γ-successor, Theorem V.4.1 establishes that the belief function returns the minimum payoff among strings reaching the current state. Since all strings generated by a control-observation sequence have the same observation with the same payoffs, the minimum payoff is due to the unobservable substrings.

V.4.2 Build the First Cycle Energy Inclusive Controller

We consider both energy flow and information flow under control and define a discrete structure called the *first cycle energy inclusive controller (FCEIC)* for Problem V.3.1 and Problem V.3.2. The two variants of FCEICs are formally defined by construction, i.e., by adding feasible e_o-successors and γ-successors to the state space recursively in Algorithm 11 and Algorithm 12, respectively. The FCEICs with respect to system G for both problems are constructed in a similar way and of the same generic form $(Q^F_Y, Q^F_Z, E, f^F_{yz}, f^F_{zy}, \Gamma, y^e_0, Q^F_l, v_0)$ where:

- $Q^F_Y \subseteq Q^E$ is the set of energy information states;
- $Q^F_Z \subseteq Q^E \times \Gamma$ is the set of augmented energy information states and for $z^e \in Q^F_Z$, $z^e = (I_E(z^e), \Gamma(z^e))$;
- $f^F_{yz} : Q^F_Y \times \Gamma \to Q^F_Z$ is the transition function from Q^F_Y states to Q^F_Z states, where for all $y^e \in Q^F_Y$, $\gamma \in \Gamma$ and $z^e \in Q^F_Z$, $[f^F_{yz}(y^e, \gamma) = z^e] \Rightarrow [z^e$ is a γ-successor of $y^e]$;
- $f^F_{zy} : Q^F_Z \times E_o \to Q^F_Y$ is the transition function from Q^F_Z states to Q^F_Y states, where for all $z^e \in Q^F_Z$, $e_o \in E_o$ and $y^e \in Q^F_Y$, $[f^F_{zy}(z^e, e_o) = y^e] \Leftrightarrow [y^e$ is an e_o-successor of $z^e]$;
- Γ is the set of admissible control decisions;

- $y_0^e \in Q_Y^F$ is the initial energy information state where $Est(y_0^e) = x_0$ and $Lev(y_0^e) = v_0$;
- Q_l^F is the set of leaf Q_Y^F states;
- $v_0 \in \mathbb{N}$ is the initial energy of the system.

Algorithm V.1: Construction of the FCEIC for Problem V.3.1

Input : G, v_0
Output : $FCPEC = (Q_Y^F, Q_Z^F, E, f_{yz}^F, f_{zy}^F, \Gamma, y_0^e, Q_l^F, v_0)$

$Q_Y^F = \{y_0^e\}$, $Q_Z^F = \emptyset$, $Q_l^F = \emptyset$;
$FirstCycle_1(y_0^e, FCPEC)$;
Return $FCEIC$;

Procedure: $FirstCycle_1(y^e, FCPEC)$
for $\gamma \in \Gamma$ **do**
  Let z^e be a γ-successor of y^e;
  if *z^e is deadlock free and energy safe* **then**
    Add transition $y^e \xrightarrow{\gamma} z^e$ to f_{yz}^F;
    if $z^e \notin Q_Z^F$ **then**
      $Q_Z^F = Q_Z^F \cup \{z^e\}$;
      for $e_o \in \gamma \cap E_o$ **do**
        Let $\tilde{y}^e$ be an e_o-successor of z^e;
        Add transition $z^e \xrightarrow{e_o} \tilde{y}^e$ to f_{zy}^F;
        if $\tilde{y}^e \notin Q_Y^A$ **then**
          $Q_Y^F = Q_Y^F \cup \{\tilde{y}^e\}$;
          if *$\tilde{y}^e$ is energy safe* **then**
            if *there exists a run from y_0^e: $y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_1^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e} \tilde{y}^e$ and $\exists j < n$, s.t. $y_j^e \preccurlyeq \tilde{y}^e$* **then**
              Stop searching from $\tilde{y}^e$, $Sub(\tilde{y}^e) = y_j^e$, $Q_l^F = Q_l^F \cup \{\tilde{y}^e\}$, $Q_{lg}^F = Q_{lg}^F \cup \{\tilde{y}^e\}$;
            else
              $FirstCycle_1(\tilde{y}^e, FCPEC)$;
          else
            Stop searching from $\tilde{y}^e$, $Q_l^F = Q_l^F \cup \{\tilde{y}^e\}$, $Q_{lb}^F = Q_{lb}^F \cup \{\tilde{y}^e\}$;

We call a Q_Y^F state as Y-state and a Q_Z^F state as Z-state. A Z-state z^e is *deadlock free* if $\forall x \in Est(I_E(z^e))$, $\exists e \in \Gamma(z^e)$, s.t. $f(x,e)!$, i.e., there is an enabled event at every state in the state estimate of z^e. Otherwise, z^e is a *deadlocking* state. Since there are no unobservable loops in G by Assumption V.3.1, a deadlock free Z-state always has f_{zy}^F transitions defined out of it.

Algorithm V.2: Construction of the FCEIC for Problem V.3.2

Input : G, v_0
Output : $FCEIC = (Q_Y^F, Q_Z^F, E, f_{yz}^F, f_{zy}^F, \Gamma, y_0^e, Q_l^F, v_0)$

```
Q_Y^F = {y_0^e}, Q_Z^F = ∅, Q_l^F = ∅;
FirstCycle_2(y_0^e, FCEIC);
Return FCEIC;
Procedure: FirstCycle_2(y^e, FCEIC)
for γ ∈ Γ do
    Let z^e be a γ-successor of y^e;
    if z^e is deadlock free then
        Add transition y^e --γ--> z^e to f_yz^F;
        if z^e ∉ Q_Z^F then
            Q_Z^F = Q_Z^F ∪ {z^e};
            for e_o ∈ γ ∩ E_o do
                Let ỹ^e be an e_o-successor of z^e;
                Add transition z^e --e_o--> ỹ^e to f_zy^F;
                if ỹ^e ∉ Q_Y^A then
                    Q_Y^F = Q_Y^F ∪ {ỹ^e};
                    if there exists a run from y_0^e: y_0^e --γ_0--> z_0^e --e_0--> y_1^e ··· --γ_{n−1}--> z_{n−1}^e --e--> ỹ^e and
                    ∃j < n, s.t. y_j^e ≼ ỹ^e then
                        Stop searching from ỹ^e, Sub(ỹ^e) = y_j^e, Q_l^F = Q_l^F ∪ {ỹ^e},
                        Q_lg^F = Q_lg^F ∪ {ỹ^e};
                    if There exists a run from y_0^e: y_0^e --γ_0--> z_0^e --e_0--> y_1^e --γ_1--> z_1^e ··· --γ_{n−1}--> z_{n−1}^e --e--> ỹ^e
                    and ∃j < n, s.t. ỹ^e ≺ y_j^e then
                        Stop searching from ỹ^e, Q_l^F = Q_l^F ∪ {ỹ^e}, Q_lb^F = Q_lb^F ∪ {ỹ^e};
                    else
                        FirstCycle_2(ỹ^e, FCEIC);
```

The FCEIC in general describes a game between the supervisor and the environment. A Y-state is an energy information state where the supervisor issues control decisions. If the supervisor issues an admissible control decision γ, a f_{yz}^F transition is defined out of a Y-state, which follows the definition of γ-successor. While a Z-state is an augmented energy information state, where the environment plays by selecting observable events to occur from the events enabled by the supervisor. When a particular observable event e_o is selected to occur by the environment, a f_{zy}^F transition is defined out of a Z-state, which follows the definition of e_o-successor. Then it is again the supervisor's turn to make the next control decision. This is in consistent with the mechanism of supervisory control under partial observation where the supervisor's decision gets updated with occurrence of observable events. In this manner, the two players take turns to play and a game is formed.

The procedure $FirstCycle_i$ where $i \in \{1,2\}$ in either algorithm builds the state space of the FCEIC by a depth-first search like process. We first discuss $FirstCycle_1$ in Algorithm 11. In this process, we only add deadlock free Z-states to the structure and ensure that there are events enabled at every state in the state estimate of any Z-state. In lines 15, 16 and 17 of Algorithm 11, if the newly added energy safe state $\tilde{y}^e$ subsumes a non-leaf state y_j^e on the run starting from the initial state, then we know that the two energy information states share the same state estimate but the new state $\tilde{y}^e$ has a nondecreasing energy level vector compared with y_j^e. We also know that some simple cycles with nonnegative payoffs are formed in the system for the first time. Then we terminate searching and add the new state as a leaf state of the FCEIC. That is why we call this structure first cycle energy inclusive controller. In the following sections, we will explain in more detail why it is sufficient to consider simple cycles to solve Problem V.3.1. On the other hand, if a new Z-state or Y-state is not energy safe, we stop searching since the system's energy level drops below 0 at some state, thus the second requirement in Problem V.3.1 is violated.

Similarly for $FirstCycle_2$ of Algorithm 12, in lines 15 and 17, if the newly added state $\tilde{y}^e$ subsumes or is subsumed by an existing state on the run from initial state y_0^e, we know that the two energy information states share the same state estimate but $\tilde{y}^e$ may have a nondecreasing or decreasing energy level vector compared with the existing state. We also know that some simple cycles with nonnegative or negative payoffs are formed in the system for the first time. Then we terminate searching and add the new state as a leaf state. Since Problem V.3.2 does not require nonnegative energy level, the states created by $FirstCycle_2$ are not necessarily energy safe.

Next, we partition leaf Y-states as: $Q_l^F = Q_{lg}^F \cup Q_{lb}^F$ where Q_{lg}^F represents *good leaf states* and Q_{lb}^F represents *bad leaf states*. In the FCEIC for Problem V.3.1, a good leaf state is energy safe and subsumes a non-leaf state, while a bad leaf state is energy unsafe. If a good leaf state is reached, there are simple cycles with nonnegative payoffs in the system and the system's energy level would be nonnegative forever if those cycles are traversed indefinitely. However, if a bad leaf state is reached, the energy level of the system drops below 0 by some strings. Similarly, in the FCEIC for Problem V.3.2, a good leaf state subsumes a non-leaf state while a bad leaf state is subsumed by a non-leaf state. If a good leaf state is reached, we know there exist simple cycles with nonnegative payoffs in the system; while if a bad leaf state is reached, there exist simple cycles with negative payoffs. In both algorithms, we use $Sub(y^e)$ to store the preceding state subsumed by good leaf state y^e. Actually, the goal of the supervisors in both Problem V.3.1 and Problem V.3.2 is to reach good leaf states but to avoid bad ones, which is explained in more detail later on. Finally, if no state subsumes another, we call $FirstCycle$ recursively in both algorithms until no more new states are added to the structure.

We now show that Algorithm 11 and Algorithm 12 converge in finite steps and return a finite and acyclic structure.

Theorem V.4.2. *Algorithm 11 returns a finite structure.*

Proof. By contradiction, assume that the FCEIC is infinite. Since E, $\Gamma \subseteq 2^E$ and E_o are finite, the number of transitions defined at each state in the structure is finite. Then by König's lemma (see, e.g., [69]) and Algorithm 11, there exists an infinite run $y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_1^e \xrightarrow{\gamma_1} z_1^e \cdots$ in the FCEIC such that it is never the case that $\exists y_i^e, y_j^e$, $i < j$, s.t. $y_i^e \preccurlyeq y_j^e$. However, this contradicts with the fact that $\preccurlyeq$ is a well-quasi ordering on energy safe energy information sates. $\square$

Theorem V.4.3. *Algorithm 12 returns a finite structure.*

Proof. Prove by contradiction. Assume that the FCEIC is infinite. Since E, $\Gamma \subseteq 2^E$ and E_o are finite, the number of transitions defined at each state in the structure is finite. Then by König's lemma (see, e.g., [69]), there exists an infinite run $y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_1^e \xrightarrow{\gamma_1} z_1^e \cdots$ in the FCEIC such that it is neither the case that $\exists y_i^e, y_j^e$, $i < j$, s.t. $y_i^e \preccurlyeq y_j^e$ nor the case that $y_j^e \prec y_i^e$. That means there exist y_i^e, y_j^e $(i < j)$ and integers $k \neq l$ s.t. $Est(y_i^e) = Est(y_j^e)$, $Lev(y_i^e)(k) \leq Lev(y_j^e)(k)$ and $Lev(y_i^e)(l) > Lev(y_j^e)(l)$ for elements in $Lev(y_i^e)$ and $Lev(y_j^e)$. Hence there exist two simple cycles in G: $x_1 \xrightarrow{e_1} x_2 \cdots \xrightarrow{e_n} x_1$ and $x'_1 \xrightarrow{e'_1} x'_2 \cdots \xrightarrow{e'_n} x'_1$ s.t. $x_1, x'_1 \in Est(y_i^e)$, $P(e_1 \cdots e_n) = P(e'_1 \cdots e'_n)$, $\omega(e_1 \cdots e_n) \geq 0$ and $\omega(e'_1 \cdots e'_n) < 0$. However, this contradicts with Assumption V.3.2 that G is with unambiguous cycle payoffs. □

As for the space complexity of the FCEIC, the size of its state space is bounded by Ackermann function [92] following a similar argument as in [87], which solved energy games by "unfolding" the game graph until a simple cycle is formed.

Example V.4.1. *In this example, we construct a first cycle energy inclusive controller following Algorithm 11. Let the system G in Figure V.2 be with* $E_o = \{o_1, o_2, o_3, o_4\}$, $E_{uo} = \{a_1, a_2, a_3, a_4, b_1, b_2, c_1, c_2, c_3, c_4.c_5\}$, $E_c = \{c_1, c_2, c_3, c_4, c_5\}$, $E_{uc} = \{a_1, a_2, a_3, a_4, b_1, b_2, o_1, o_2, o_3, o_4\}$. *The weight of each event is shown in the figure and the system has initial energy* $v_0 = 3$. *Then all admissible control decisions are:* $\gamma_0 = E_{uc}$, $\gamma_1 = \{c_1, c_2\} \cup E_{uc}$, $\gamma_2 = \{c_3\} \cup E_{uc}$, $\gamma'_2 = \{c_3, c_5\} \cup E_{uc}$, $\gamma_3 = \{c_4\} \cup E_{uc}$, $\gamma_4 = \{c_1\} \cup E_{uc}$, $\gamma_5 = \{c_2\} \cup E_{uc}$.

Then we follow Algorithm 11 to build the FCEIC in Figure V.3. For simplicity of the graph, we do not put the energy level vectors in the figure but show them in Table V.4.1. The elements in each energy level vector are placed in the same order as the order of states in the state estimate.

In the FCEIC, the game is initiated from y_0^e *where the only feasible control decision is* γ_0. *If the supervisor plays* γ_0, *a Z-state* z_0^e *is reached where the environment selects observable event* o_1 *to occur. Then the supervisor takes the turn to play at* y_1^e *and the rest of the structure is interpreted in a similar way. Notice that at* y_2^e, *the supervisor should not issue control decision* γ_2 *to enable*

state name	state components
y_0^e	$\{\{x_0\}, 3\}$
z_0^e	$\{\{x_0, x_1, x_2\}, [3, 1, 0], \gamma_0\}$
y_1^e	$\{\{x_3, x_4\}, [2, 1]\}$
z_1^e	$\{\{x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}, [2, 1, 5, 2, 7, 2, 6, 5], \gamma_1\}$
y_2^e	$\{\{x_{12}\}, 4\}$
z_2^e	$\{\{x_{12}\}, 4, \gamma_0\}$
y_{2-2}^e	$\{\{x_{12}\}, 6\}$
z_8^e	$\{\{x_9, x_{12}, x_{14}\}, [0, 4, 3], \gamma_2'\}$
y_{2-3}^e	$\{\{x_{12}\}, -2\}$
y_{2-4}^e	$\{\{x_{12}\}, 6\}$
y_3^e	$\{\{x_{13}\}, 2\}$
z_3^e	$\{\{x_{13}\}, 2, \gamma_0\}$
y_{3-2}^e	$\{\{x_{13}\}, 4\}$
z_9^e	$\{\{x_{10}, x_{13}, x_{15}\}, [1, 2, 1], \gamma_3\}$
y_{3-3}^e	$\{\{x_{13}\}, -2\}$
y_{3-4}^e	$\{\{x_{13}\}, 4\}$
z_4^e	$\{\{x_3, x_4, x_5, x_7\}, [2, 1, 5, 7], \gamma_4\}$
y_{1-2}^e	$\{\{x_3, x_4\}, [3, 2]\}$
z_5^e	$\{\{x_3, x_4, x_6, x_8\}, [2, 1, 2, 2], \gamma_5\}$
y_{1-3}^e	$\{\{x_3, x_4\}, [3, 2]\}$
z_6^e	$\{\{x_3, x_4\}, [2, 1], \gamma_0\}$
y_{1-4}^e	$y_{1-4}^e = \{\{x_3, x_4\}, [3, 2]\}$
y_{1-5}^e	$y_{1-5}^e = \{\{x_3, x_4\}, [3, 2]\}$

Table V.1: Energy and augmented energy information states in Figure V.3

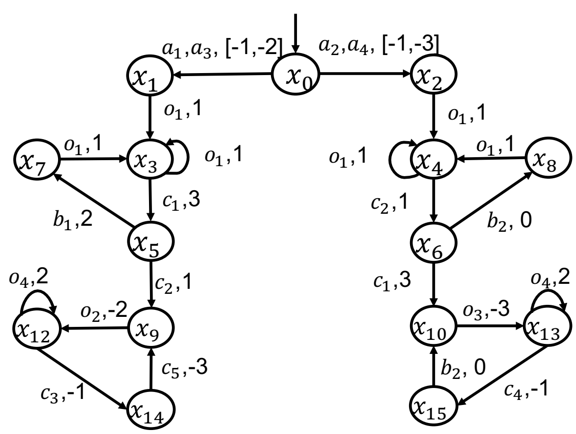

Figure V.2: The automaton G in Example V.4.1

c_3 but to disable c_5. Otherwise, a deadlocking Z-state z_7^e is reached since no event can occur at x_{14} if c_5 is disabled. Here z_7^e is not included in the FCEIC by Algorithm 11. Meanwhile, we

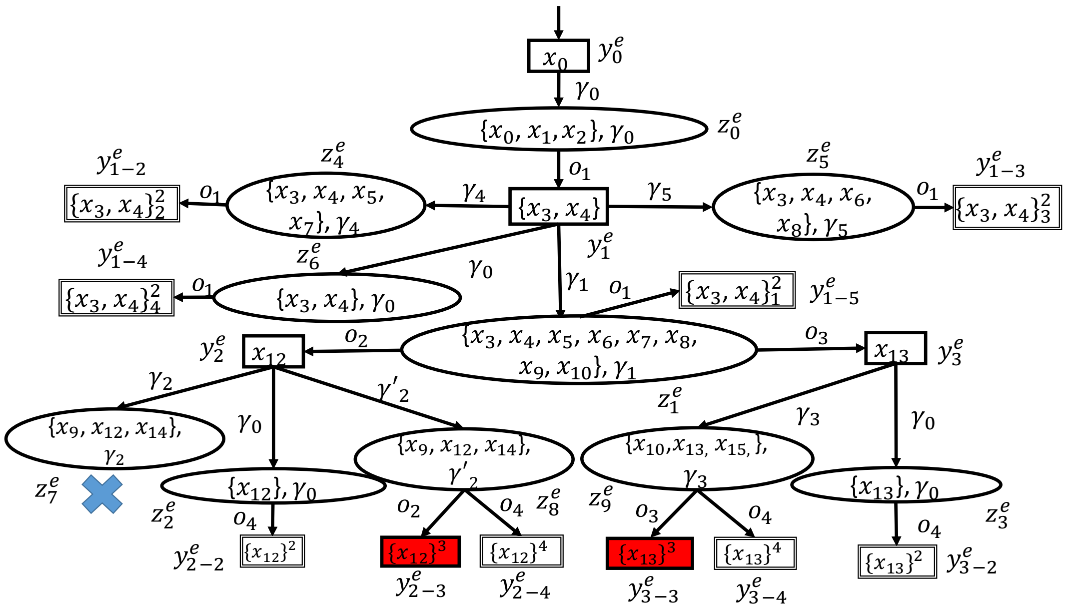

Figure V.3: The First Cycle Positive Energy Controller in Example V.4.1(without z_7^e)

calculate the energy level vector of each state. For example, $Est(y_0^e) = \{x_0\}$*,* $Lev(y_0^e) = v_0 = 3$*; since* z_0^e *is the* γ_0*-successor of* y_0^e*, we have that* $Est(I_E(z_0^e)) = UR_{\gamma_0}(Est(y_0^e)) = \{x_0, x_1, x_2\}$*,* $h_{z_0^e}(x_1) = \min\{\omega(a_1), \omega(a_3)\} = 1$*,* $h_{z_0^e}(x_2) = \min\{\omega(a_2), \omega(a_4)\} = 0$ *and* $z_0^e = \{\{x_0, x_1, x_2\}, [3, 1, 0], \gamma_0\}$*; since* y_1^e *is the* o_1*-successor of* z_1^e*, we have that* $Est(y_1^e) = Next_{o_1}(\{\{x_0, x_1, x_2\}) = \{x_3, x_4\}$*,* $h_{y_1^e}(x_3) = h_{z_0^e}(x_1) + \omega(o_1) = 2$*,* $h_{y_1^e}(x_4) = h_{z_0^e}(x_2) + \omega(o_1) = 1$ *and* $y_1^e = \{\{x_3, x_4\}, [2, 1]\}$*.*

From the table, we find that $y_1^e \preccurlyeq y_{1-2}^e$*,* $y_1^e \preccurlyeq y_{1-3}^e$*,* $y_1^e \preccurlyeq y_{1-4}^e$*,* $y_1^e \preccurlyeq y_{1-5}^e$*,* $y_2^e \preccurlyeq y_{2-2}^e$*,* $y_2^e \preccurlyeq y_{2-4}^e$*,* $y_3^e \preccurlyeq y_{3-2}^e$ *and* $y_3^e \preccurlyeq y_{3-4}^e$ *by evaluating their energy level vectors. We also find two energy unsafe states* y_{2-3}^e *and* y_{3-3}^e *since* $Lev(y_{2-3}^e) = -2$ *and* $Lev(y_{3-3}^e) = -2$*. We stop searching from the leaf states in Figure V.3, then have good leaf states* $Q_{lg}^F = \{y_{1-2}^e, y_{1-3}^e, y_{1-4}^e, y_{1-5}^e, y_{2-2}^e, y_{2-4}^e, y_{3-2}^e, y_{3-4}^e\}$ *and bad leaf states* $Q_{lb}^F = \{y_{2-3}^e, y_{3-3}^e\}$*. For example, when* y_{1-2}^e *is reached, we locate three simple cycles with nonnegative payoffs in automaton G:* $x_3 \xrightarrow{c_1} x_5 \xrightarrow{b_1} x_7 \xrightarrow{o_1} x_3$ *with payoff* 6*,* $x_3 \xrightarrow{o_1} x_3$ *with payoff* 1 *and* $x_4 \xrightarrow{o_1} x_4$ *with payoff* 1*. The bad leaf states actually come from the two simple cycles with negative payoffs in G:* $x_9 \xrightarrow{o_2} x_{12} \xrightarrow{c_3} x_{14} \xrightarrow{c_5} x_9$ *with payoff* -6 *and* $x_{10} \xrightarrow{o_3} x_{13} \xrightarrow{c_4} x_{15} \xrightarrow{b_2} x_{10}$ *with payoff* -4*. Those two cycles should be avoided if we want to solve Problem V.3.1.*

Example V.4.2. *The system G is the same as the one in Example V.4.1 and we construct a first* *cycle energy inclusive controller following Algorithm 12. It happens that the FCEIC is the same as the one in Figure V.3. Specifically, $y^e_{2-3} \prec y^e_2$ and $y^e_{3-3} \prec y^e_3$, so y^e_{2-3} and y^e_{3-3} are also bad leaf states in this example. They are due to the two simple cycles with negative payoffs mentioned at the end of Example V.4.1. Again, those two cycles should be avoided if we want to solve Problem V.3.2.*

V.5 Mean Payoff Decision Problems

In this section, we first discuss some properties of the first cycle energy inclusive controller (FCEIC), then partially solve Problem V.3.1 and Problem V.3.2 by synthesizing a supervisor that satisfies the first two requirements in each problem. As was mentioned earlier, the first two conditions in both problems constitute the so-called *mean payoff decision problems*. The last requirement in both problems, i.e., the optimization issue, will be discussed and addressed in the next section. Since the following analysis apply to both FCEICs returned by Algorithm 11 and Algorithm 12, we will not distinguish them but just use the term "FCEIC" when there is no confusion.

By definition, the runs in the FCEIC (defined by both Algorithms 11 and 12 are the finite control-observation sequences discussed in the last section. We denote by $Run(F)$ the set of runs in the FCEIC. Given $r_f \in Run(F)$, we denote by $y^e \in r_f$ and $z^e \in r_f$ if y^e (respectively z^e) is a Y-state (respectively Z-state) in r_f. We also let $Last_Y(r_f)$ and $Last_Z(r_f)$ be the last Y-state and Z-state of r_f, respectively. Besides, we denote by $Run_y(F)$ (respectively $Run_z(F)$) the set of runs whose last states are Y-states (respectively Z-states).

Then we discuss strategies of both players in the FCEIC. Define the *supervisor's strategy (control strategy)* as $\pi_s : Run_y(F) \to \Gamma$ and *environment's strategy* as $\pi_e : Run_z(F) \to E_o$. Both players select a transition according to their strategies when it is their turn to play. Since the supervisor only has partial observation of the system and makes decisions from state estimates, we call its strategy *observation based*. Denote the set of all supervisor's strategies by Π_s and the set of all environment's strategies by Π_e. If the supervisor plays π_s while the environment plays π_e from the initial state y^e_0, then a unique initial run, denoted by $r_f(\pi_s,\pi_e)$, is generated. We also let $Run(y^e,\pi_s) =$

$\{y^e \xrightarrow{\gamma_1} z^e_1 \xrightarrow{e_1} y^e_2 \cdots \xrightarrow{\gamma_{n-1}} z^e_{n-1} \xrightarrow{e_{n-1}} y^e_n : \forall i < n, \gamma_i = \pi_s(y^e \xrightarrow{\gamma_1} z^e_1 \xrightarrow{e_1} y^e_2 \cdots \xrightarrow{\gamma_{i-1}} z^e_{i-1} \xrightarrow{e_{i-1}} y^e_i)\}$ be the set of runs starting from y^e and *consistent* with control strategy π_s, i.e., the control decisions in the run are specified by π_s.

In the FCEIC, we say the supervisor wins the game if only good leaf states are reached, otherwise, the environment wins the game if bad leaf states are reached. So the game on the FCEIC is a *zero sum* safety game. The game on the FCEIC is of full observation after introducing the energy information states and either the supervisor or the environment has a *winning strategy* from any state in the FCEIC, since safety games are *determined* [4].

A strategy $\pi_i \in \Pi_i$ for player $i \in \{s, e\}$ in the FCEIC is *information state based* if the decisions only depend on the current energy or augmented energy information state. In other words, $\pi_i \in \Pi_i$ is information state based if $\pi_i(r_f) = \pi_i(r'_f)$ for all $r_f, r'_f \in Run(F)$ such that $Last(r_f) = Last(r'_f)$. Therefore, information state based strategies for the supervisor and the environment can be represented by $\pi_s : Q^F_Y \to \Gamma$ and $\pi_e : Q^F_Z \to E_o$, respectively. We also call an information state based strategy *positional*. From existing results, see, e.g. [4, 43], positional strategies are sufficient to win a finite safety game so in the following discussion, we assume that both players' strategies are positional.

Following the transitions in the FCEIC, we can specify control decisions from Y-states and the control decisions are updated after observable events occur from Z-states. Thus the control strategies in the FCEIC work in the same way as standard partial observation supervisors. In the following discussion, we will use the words "supervisor" and "supervisor's strategy (control strategy)" interchangeably.

We define *the supervisor's winning region* Win_s as the set of states from which the supervisor has a strategy to reach good leaf states *for sure* regardless of the environment's strategies. To solve Problem V.3.1 or Problem V.3.2, the supervisor should only reach good leaf states. Actually, the procedures to obtain Win_s for both problems are the same after the FCEIC is given. Hence we present one unified algorithm, i.e., Algorithm 13, to compute Win_s for Problem V.3.1 or Problem V.3.2.

Algorithm V.3: Compute the winning region of the FCEIC

Input : $FCEIC$ returned by Algorithm 11 or Algorithm 12

Output : Win_s for Problem V.3.1 or Problem V.3.2

while $\exists y^e \in Q_Y^F \setminus Q_{lg}^F$, *s.t.* y^e *has no successor* **do**

  Remove y^e and all $z^e \in Q_Z^F$, s.t. $f_{zy}^F(z^e, e_o) = y^e$ for some $e_o \in E_o$;

  Take the accessible part of the structure;

Denote the remaining structure by $FCEIC_w$ and return the states in it;

In Algorithm 13, all bad leaf states are removed first as well as their preceding Z-states. Then we further prune away Y-states that have no successor states and their preceding Z-states in an iterative manner until no more states are removed. Notice that when we prune away a Y-state, we also need to remove all its preceding Z-states, otherwise the already enabled observable events are blocked from happening. However, when a Z-state is removed, we only remove its preceding Y-state if the Y-state has no successors, since the supervisor is still able to avoid the removed Z-state when it has other successors.

Algorithm 13 is similar to the standard procedure of calculating *attractors* and *winning regions* of graph games in a *fixed point calculation* manner [4]. Besides, it is also similar to calculating the *supremal controllable sublanguage* in nonblocking supervisory control problem under full observation [23]: the bad leaf states are viewed as undesirable marked states while the good leaf states are viewed as desirable ones; besides, f_{yz}^F transitions are viewed as controllable while f_{zy}^F transitions are viewed as uncontrollable. In this way, we make sure that only good leaf states are reached under certain control strategies. In other words, any control strategy in the FCEIC$_w$ is a winning control strategy in the FCEIC, and vice versa. It is possible that Algorithm 13 returns an empty set thus the environment always wins the game regardless of the supervisor's strategies.

Then we argue that if there exists a winning control strategy in the FCEIC, i.e., Win_s is not empty, then there always exists a supervisor solving the mean payoff decision problem of Problem V.3.1 or Problem V.3.2. The idea is straightforward. If only good leaf states are reached under a wining control strategy π_s in the FCEIC, then only simple cycles with nonnegative payoff are formed in the supervised system. Since a belief function in an energy information state returns the minimum string payoff by Theorem V.4.1, the payoffs of all strings with the same observation and reaching the same state are nonnegative if the minimum string payoff is nonnegative.

We let the supervisor make the same decision whenever the state estimate of a good leaf state is reached again. Intuitively speaking, the supervisor "ignores" the actual energy level of the system and just views the game starting from a good leaf state y^e as the same game that starts from the state subsumed by y^e. We can imagine that y^e is "merged" with $Sub(y^e)$ by letting all transitions going to y^e lead to $Sub(y^e)$ instead. In this way, the supervisor perpetually completes cycles with nonnegative payoffs since every simple cycle has a nonnegative payoff. So the limit mean payoff of every infinite run in the supervised system is also nonnegative.

Since there are no deadlocking Z-states and every Y-state has successors in the FCEIC$_w$, we may show that the supervised system by any control strategy in the FCEIC$_w$ is live, following a similar argument as in Section V of [126]. Overall, any control strategy in the FCEIC$_w$ solves the mean payoff decision problem of Problem V.3.1 or Problem V.3.2. Conversely, we claim that if the mean payoff decision problem has solutions, then we can find winning control strategies in the FCEIC returned by either Algorithm 11 or 12. Formally speaking, the following two theorems hold.

Theorem V.5.1. *There exists a supervisor solving the mean payoff decision problem of Problem V.3.1 if and only if the supervisor has a winning strategy in the FCEIC defined by Algorithm 11.*

Proof. The "only if" part. We show by contrapositive, i.e., if there does not exist a winning control strategy in the FCEIC, then there does not exist a supervisor solving the mean payoff decision problem. If no winning control strategy exists, then Win_s is empty by Algorithm 11. So $\forall \pi_s \in \Pi_s$, $\exists \pi_e \in \Pi_e$, s.t. $Last_Y(r_f(\pi_s, \pi_e)) \in Q_l^F \Rightarrow Last_Y(r_f(\pi_s, \pi_e)) \in Q_{lb}^F$, i.e, no matter what decisions made by the supervisor, there always exist runs ending in bad leaf states. Therefore for π_s, there always exists a run r consistent with π_s in the supervised system such that $V(r) < 0$, i.e., the supervised system's energy level becomes negative under π_s for some string. That is to say, no supervisor solves the mean payoff decision problem.

The "if" part. Suppose that π_s is a winning control strategy in the FCEIC. We follow Algorithm 13 and obtain Win_s and FCEIC$_w$, so π_s is also in the FCEIC$_w$. In the following discussion, we imagine that all transitions leading to a leaf state y^e in the FCEIC$_w$ lead to $Sub(y^e)$ so that the game on the FCEIC$_w$ becomes infinite-duration. That is, $\forall r_f = y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_i^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y_n^e \in Run(y_0^e, \pi_s)$ where y_0^e is the initial state of the FCEIC, if $y_n^e \in Q_{lg}^F$, then we extend the domain of π_s by letting $\pi_s(r_f) = \pi_s(y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_i^e \cdots \xrightarrow{e_{m-1}} y_m^e)$ for some $m < n$ and $y_m^e \preceq y_n^e$. Whenever $Est(y_n^e)$ is reached again, the control strategy (supervisor) makes the same decision as if $Est(y_n^e)$ is reached for the first time. By perpetually making the same decision whenever a state estimate is reached, the supervisor guarantees that the energy level in the supervised system never becomes negative since all states in the FCEIC$_w$ are energy safe and only cycles with nonnegative payoffs are formed and traversed infinitely often.

Finally, the system under the constructed supervisor is live following a similar argument as in Section V of [126]. Thus π_s solves the decision problem of Problem V.3.1. □

Theorem V.5.2. *There exists a supervisor solving the mean payoff decision problem of Problem V.3.2 if and only if the supervisor has a winning strategy in the FCEIC defined by Algorithm 12.*

Proof. The proof is similar to that of Theorem V.5.1. We just substitute the argument of limit mean payoff for the argument of the total payoff to show this result. □

Therefore, we have shown that we can transform the mean payoff decision problem for Problem V.3.1 (Problem V.3.2) into a safety game under perfect information on the FCEIC and solve it by finding winning control strategies. We have also shown the soundness and completeness of Algorithms 11 and 12.

Example V.5.1. *We revisit Example V.4.1 (Example V.4.2) to find the winning regions of the FCEIC following Algorithm 13. Since the good (bad) leaf states in both examples coincide, the winning regions for both examples remain the same. The FCEIC$_w$ is shown in Figure V.4, where green dashed lines connect each good leaf state with the state subsumed by it, indicating that the supervisor always makes the same decision from the two connected states. So the game is extended to be infinite-duration. In building the FCEIC$_w$, red states y_{2-3}^e and y_{3-3}^e in Figure V.3 are bad leaf* *states, thus are pruned by Algorithm 13. Meanwhile, good leaf states y^e_{2-4} and y^e_{3-4} are also removed as they become no longer accessible from the initial state y^e_0 after their preceding Z-states z^e_8 and z^e_9 are removed. That means that the supervisor should not choose γ'_2 at y^e_2 or γ_3 at y^e_3, otherwise, the environment can choose o_2 at z^e_8 or o_3 at z^e_9 to reach some bad leaf states and win the game.*

Then we locate a winning control strategy, which is indicated by blue lines in Figure V.4. As is seen, the supervisor S issues γ_0 at y^e_0, γ_1 at y^e_1, γ_0 at y^e_2 and γ_0 at y^e_3. If the supervisor makes those decisions infinitely often, then only cycles with nonnegative payoffs are formed in the supervised system. Finally we show the supervised system under this strategy in Figure V.5. Compared with the original system in Figure V.2, the cycles with a negative payoff have been broken. Then it is easy to verify that the supervised system is live and all infinite runs have a positive limit mean payoff. So S solves the mean payoff decision problem of Problem V.3.1 (Problem V.3.2).

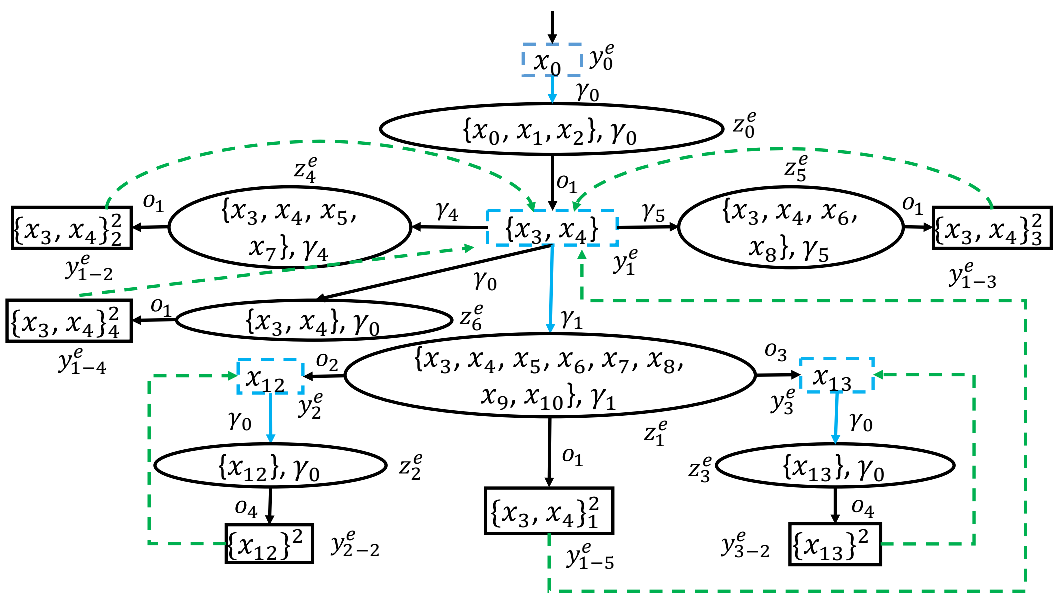

Figure V.4: The FCEIC$_w$ with dashed green lines connecting good leaf states with their subsumed states; Win_s is the set of all states

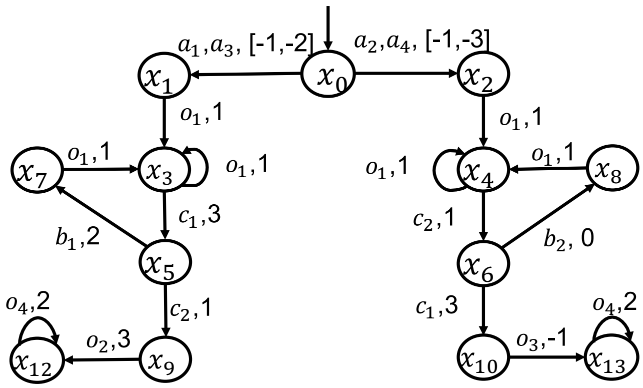

Figure V.5: A supervisor solving the mean payoff decision problem

V.6 Mean Payoff Optimization Problems

In the preceding section, we investigated the mean payoff decision problems of both Problem V.3.1 and Problem 12. Among the potentially multiple control strategies in the FCEIC$_w$, we find an optimal one and completely solve both problems in this section. As there is no difference between the procedures of obtaining the optimal control strategies for the two problems, we present a uniform optimization approach in this section.

In the FCEIC$_w$, we denote by $Run(F_w)$ the set of runs and $Run_{leaf}(F_w)$ the set of runs ending in a good leaf state, respectively. Given a run $r_f = y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_1^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y_n^e \in Run(F_w)$ with $y_j^e \preccurlyeq y_n^e$ for some $j < n$ where y_n^e is a leaf state, we know that simple loops with nonnegative payoffs are generated from each state in state estimate $I(y_j^e)$.

In order to determine the mean payoffs of strings generated by runs in the FCEIC$_w$, we need to know exactly what observable and unobservable events are in the string. However, we only know the occurrence of observable events from transitions in the FCEIC$_w$ since the unobservable transitions are within each state. In order to explicitly show the inner connections between states by unobservable strings *inside* each Y-state or Z-state in the FCEIC$_w$, we introduce a new automaton called the *Energy Inter Connected System* (EICS), which is inspired by the Inter Connected System proposed in [125].

Definition V.6.1 (Energy Inter Connected System (EICS))**.** *Given the FCEIC$_w$ w.r.t. system G, its* *corresponding Energy Inter Connected System (EICS) is defined as:*

$$EICS = (Q^{EICS}, E^{EICS}, f^{EICS}, q_0^{EICS}, Q_l^{EICS})$$

where:

- $Q^{EICS} \subseteq (Q_Y^F \times X) \cup (Q_Z^F \times X)$ *is the state space such that:*
  - $(y^e, x) \in Q^{EICS}$ *if* $y^e \in Q_Y^F$ *and* $x \in I(y^e)$*;*
  - $(z^e, x) \in Q^{EICS}$ *if* $z^e \in Q_Z^F$ *and* $x \in I(I_E((z^e))$*;*
- $E^{EICS} = E \cup \Gamma$ *is the set of events in the EICS;*
- $f^{EICS} : Q^{EICS} \times E^{EICS} \to Q^{EICS}$ *is the partial transition function defined as:* $\forall \gamma \in \Gamma$, $\forall e \in E$*:*
  - $f^{EICS}((y^e, x_1), \gamma) = (z^e, x_2)$ *if* $x_1 = x_2$ *in* G *and* $f_{yz}^F(y^e, \gamma) = z^e$ *in the* $FCEIC_w$*;*
  - $f^{EICS}((z^e, x_1), e) = (z^e, x_2)$ *if* $f(x_1, e) = x_2$ *in* G *and* $e \in \Gamma(z^e) \cap E_{uo}$*;*
  - $f^{EICS}((z^e, x_1), e) = (y^e, x_2)$ *if* $f(x_1, e) = x_2$ *in* G*,* $e \in \Gamma(z^e) \cap E_o$ *and* $f_{zy}^F(z^e, e) = y^e$ *in the* $FCEIC_w$*;*
- $q_0^{EICS} = \{y_0^e, x_0\}$ *is the initial state;*
- $Q_l^{EICS} = \{(y^e, x) \in Q^{EICS} : y^e \in Q_{lg}^F$ *in the* $FCEIC_w\}$ *is the set of leaf states.*

Intuitively, the EICS is similar to the structure obtained from parallel composition between the FCEIC$_w$ and the system G. It explicitly shows both observable and unobservable reaches between and within states of the FCEIC$_w$. The state components in the EICS are from the FCEIC$_w$ and G. There are three types of f^{EICS} transitions defined in the EICS. The first type indicates the supervisor's decisions from certain states of the system, so the first component of an EICS state changes from a Y-state to its succeeding Z-state in the FCEIC$_w$ while the second component stays the same. The second type indicates the unobservable reaches within Z-states in the FCEIC$_w$, so the first state component of (z^e, x_1) stays the same while the second component becomes $x_2 = f(x_1, e)$

under $e \in \Gamma(z^e) \cap E_{uo}$. The third type indicates observable reaches between Y-states and Z-states in the FCEIC$_w$, so the first component gets updated from a Z-state to its succeeding Y-state in the FCEIC$_w$ and the second component also gets updated by the enabled observable event. With the EICS built, we are able to explicitly see how simple cycles are formed under control decisions in the FCEIC$_w$.

By definition, the EICS is an acyclic structure whose leaf states contain leaf states of the FCEIC$_w$. Those states also indicate simple cycles in the FCEIC$_w$. For a leaf state $(y^e, x) \in Q_l^{EICS}$, we are able to track simple loops starting from $x \in Est(y^e)$ by following transitions between $(\tilde{y}^e, x)$ and (y^e, x), where $\tilde{y}^e \leq y^e$. We define $Lp_{sim}(y^e, x) = \{t \in E^* : \exists r_f = y_0^e \xrightarrow{\gamma_0} z_0^e \xrightarrow{e_0} y_1^e \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y^e \in Run(F_w), \text{s.t. } \exists j < n, y_j^e \leq y^e, t \in Str(y_j^e \xrightarrow{\gamma_j} z_j^e \xrightarrow{e_j} \cdots \xrightarrow{\gamma_{n-1}} z_{n-1}^e \xrightarrow{e_{n-1}} y^e), f(x, t) = x\}$ as the set of such simple loops. For a simple loop $t \in Lp_{sim}(y^e, x)$, we denote by $V_{sl}(t) = \frac{\omega(t)}{|t|}$ its mean payoff.

Furthermore, we define $V_{leaf} : Run_{leaf}(F_w) \to \mathbb{R}$ to characterize the (limit) mean payoff of runs ending in a leaf state of the FCEIC$_w$. For a run r_f ending in a leaf state y^e, we have $V_{leaf}(r_f) = \min_{x \in Est(y^e)} \min_{t \in Lp_{sim}(y^e, x)} V_{sl}(t)$, i.e., the minimum possible mean payoff of all simple loops formed from states in $Est(y^e)$. We take the minimum mean payoff among simple loops to characterize the (limit) mean payoff of the run, since only the cyclic part of a run contributes to the limit mean payoff and the supervisor needs to maximize the worst case limit mean payoff. With a slight abuse of notation, we also use $V_{leaf}(Last(r_f))$ to stand for $V_{leaf}(r_f)$.

Given a pair of strategies $\pi_s \in \Pi_s$ and $\pi_e \in \Pi_e$ in the FCEIC$_w$, we let $r_f(\pi_s, \pi_e)$ be the unique initial run generated under (π_s, π_e) and its last state $Last(r_f(\pi_s, \pi_e)) \in Q_{lg}^F$. Then we define the optimal control strategy in the FCEIC$_w$.

Definition V.6.2 (Optimal Control Strategy in the FCEIC$_w$)**.** *Suppose that π_s^* is a winning control strategy in the FCEIC$_w$, it is optimal if* $\min_{\pi_e \in \Pi_e} V_{leaf}(r_f(\pi_s^*, \pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(\pi_s, \pi_e))$.

Since both Π_s and Π_e are finite sets in the FCEIC$_w$, an optimal control strategy always exists by enumeration. We may compute the mean payoffs of strings from the leaf states in the EICS and those strings are generated under certain control strategies. Since we assume that the supervisor always plays positional strategies, whenever a simple cycle with positive payoff is formed, the supervisor would perpetually form the same cycle in the rest of the game. Furthermore, since the limit mean payoff of a run only depends on the mean payoff of the simple cycle traversed infinitely often, it is possible to calculate $V_{leaf}(r_f(\pi_s,\pi_e))$ from the FCEIC$_w$.

From Definition V.6.2, the optimal supervisor is to maximize its mean payoff against the environment's strategies, which are to minimize the supervisor's mean payoff. So the optimization problem can be viewed as a *min-max game* [83] on the FCEIC$_w$. Next, we leverage the standard technique of *backward induction* [83] to determine the optimal control strategy in the FCEIC$_w$. In this iterative procedure, the supervisor and the environment make decisions by maximization or minimization. Here we present Algorithm 14 to find the information state based optimal control strategy from the FCEIC$_w$ to completely solve Problem V.3.1 or Problem V.3.2.

As was mentioned before, the supervisor and the environment are playing a min-max game on the FCEIC$_w$. Algorithm 14 returns an optimal control strategy that maximizes the minimum mean payoff against the antagonistic environment's strategies. In Algorithm 14, the EICS is used to determine the mean payoffs of simple loops from the leaf states of the FCEIC$_w$ in line 5. For a leaf state $(y^e,x) \in Q_l^{EICS}$, we can always find another state $(\tilde{y}^e,x) \in Q^{EICS}$ such that $\tilde{y}^e \preceq y^e$ in the FCIEC$_w$. Then we track f^{EICS} transitions to find both observable and unobservable events between $(\tilde{y}^e,x)$ and $(y^e,x) \in Q_l^{EICS}$. Afterwards, we determine $Lp_{sim}(y^e,x)$ and calculate $V_{sl}(t)$ for each $t \in Lp_{sim}(y^e,x)$. There may be multiple simple loops formed from $x \in Est(y^e)$, with different mean payoffs. Then we calculate $V_{leaf}(y^e)$, the minimum mean payoff of all possible simple loops formed from all states in $Est(y^e)$. $V_{leaf}(y^e)$ is also the minimum possible mean payoff the supervisor may achieve when state estimate $Est(y^e)$ is reached. Since the FCEIC$_w$ is finite, Algorithm 14 always terminates.

Then we run Procedure *Optimal* to assign a value $V_F(q^e)$ to each state q^e in the FCEIC$_w$. In this procedure, we first assign values to each leaf state, then propagate the values backwards to determine the values of other states until the root state is assigned a value. Specifically, if the current state is a leaf state, we just assign V_{leaf} to it in line 13. If the current state is a Z-state, we assign the minimum value of its successor states to it in line 17. This corresponds to the fact

Algorithm V.4: Find an optimal control strategy in FCEIC$_w$

Input : $FCEIC_w$ and $EICS$ for Problem V.3.1 or V.3.2
Output : An optimal strategy π_s for Problem V.3.1 or V.3.2

for *leaf state* y^e *in* $FCEIC_w$ **do**
  for *leaf state* (y^e, x) *in* $EICS$ **do**
    Get $Lp_{sim}(y^e, x)$ following transitions in $EICS$;
    for $t \in Lp_{sim}(y^e, x)$ **do**
      Calculate $V_{sl}(t)$;
  Get $V_{leaf}(y^e) = \min\limits_{x \in Est(y^e)} \min\limits_{t \in Lp_{sim}(y^e,x)} V_{sl}(t)$;
for $q^e \in Q_Y^F \cup Q_Z^F$ **do**
  $V_F(q^e) = Optimal(q^e)$;
for $y^e \in Q_Y^F \setminus Q_l^F$ **do**
  Find one $\gamma \in \Gamma$, s.t. $\exists z^e \in Q_Z^F$, $f_{yz}^F(y^e, \gamma) = z^e$ and $V_F(y^e) = V_F(z^e)$;
  Return $\pi_s^*(y^e) = \gamma$;

Procedure: $Optimal(q^e)$
for $q^e \in Q_l^F$ **do**
  $V_F(q^e) = V_{leaf}(q_e)$;
  Return $V_F(q^e)$;
for $q_e \in (Q_Y^F \cup Q_Z^F) \setminus Q_l^F$ **do**
  if $q^e \in Q_Z^F$ **then**
    $V_F(q^e) = \min\limits_{\tilde{q}^e \in Q_Y^F} \{Optimal(\tilde{q}^e) : \exists e_o \in E_o, \text{ s.t. } f_{zy}^F(q_e, e_o) = \tilde{q}^e\}$;
    Return $V_F(q^e)$;
  if $q^e \in Q_Y^F$ **then**
    $V_F(q_e) = \max\limits_{\tilde{q}^e \in Q_Z^F} \{Optimal(\tilde{q}^e) : \exists \gamma \in \Gamma, \text{ s.t. } f_{yz}^F(q^e, \gamma) = \tilde{q}^e\}$;
    Return $V_F(q^e)$;

that the environment is to minimize the mean payoff of the supervisor. If the current state is a Y-state (not a leaf state), we assign the maximum value of its successor states to it in line 20. This comes from the fact that the supervisor is to maximize its mean payoff. This procedure goes on until a value is assigned to the initial state y_0^e of the FCEIC$_w$. When *Optimal* is implemented, we can assign orders to states in the FCEIC$_w$ so that a state is evaluated after all its successors are evaluated. This is similar to the standard process of *backward induction* in solving min-max games [83]. After obtaining V_F values, we specify the optimal control decisions at Y-states of the FCEIC$_w$, which constitute the optimal control strategy. When there are multiple optimal control decisions at the current Y-state, which occurs if some of its successors have the same V_F value, we randomly choose a control decision.

After obtaining an optimal energy information state based control strategy in the FCEIC$_w$, we can follow a similar procedure as in the last section by letting the supervisor make the same decision at the current Y-state as from the state subsumed by it. In this way, the game is extended to infinite-duration and we obtain a supervisor that issues control decisions perpetually and generates a live system. Besides, an energy information state based strategy is sufficient to be an optimal solution to Problem V.3.1 (Problem V.3.2). Intuitively, the supervisor should always traverse a simple cycle with highest mean payoff, while alternating between cycles with different mean payoffs does not contribute to a higher mean payoff. We formally present this result as follows.

Theorem V.6.1. *If π_s^* is an energy information state based control strategy returned by Algorithm 14, then we can extend π_s^* to a supervisor S^* that solves Problem V.3.1 (Problem V.3.2).*

Proof. By Algorithm 14, for every leaf state $y^e \in Q_{lg}^F$, $V_{leaf}(y^e) = \min_{x \in Est(y^e)} \min_{t \in Lp_{sim}(y^e,x)} V_{sl}(t)$. Let string $t^*(y^e)$ be such that $V_{sl}(t^*(y^e)) = \min_{x \in Est(y^e)} \min_{t \in Lp_{sim}(y^e,x)} V_{sl}(t) = V_{leaf}(y^e)$. Suppose that a Z-state z^e can reach k leaf states $y_1^e, y_2^e, \cdots, y_k^e \in Q_{lg}^F$, i.e., $\forall i \le k$, $\exists e_i \in E_o$, s.t. $f_{zy}^F(z^e, e_i) = y_i^e$. Thus we know that $V_F(z^e) = \min\{V_F(y_1^e), \cdots V_F(y_k^e)\} = \min\{V_{sl}(t(y_1^e)), \cdots, V_{sl}(t(y_k^e))\}$. Let string $t^*(z^e)$ be such that $V_{sl}(t^*(z^e)) = \min\{V_{sl}(t(y_1^e)), \cdots, V_{sl}(t(y_k^e))\}$ thus $t^*(z^e)$ is the string with the minimum loop mean payoff. Therefore, the environment still locates the string minimum mean whose simple loop has the minimum mean payoff, by evaluating $V_{leaf}(y^e)$. Also with the EICS built, we can explicitly see which cyclic string has the minimum loop mean payoff.

Suppose that one preceding Y-state of z^e is $\tilde{y}^e$ and $\tilde{y}^e$ has succeeding Z-states $z_1^e, \cdots, z_m^e$ (z^e is one of them). Then the supervisor chooses to maximize, i.e., $V_F(\tilde{y}^e) = \max_{z_i^e} V_F(z_i^e)$ where $i \le m$. Since $V_F(z_i^e)$ is the minimum mean payoff of some simple loop, then $V_F(\tilde{y}^e)$ still maximizes the minimum mean payoffs of simple loops obtained from some leaf states in the FCEIC$_w$. Thus the supervisor loses no information when making decisions by evaluating $V_F(z^e)$. By Algorithm 14, the supervisor just chooses the control decision that maximizes $V_F(z_i^e)$. Then we can repeat the same argument and work backwards to the root state to show that by evaluating the V_F values for Y-states or Z-states, the supervisor correctly performs maximization among V_F values from its successors while the environment correctly performs minimization.

Finally, we are able to conclude that $V_F(y_0^e) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(\pi_s, \pi_e))$. Then we can transfer π_s to a supervisor S^* by the same argument as in the proof of Theorem V.5.1, i.e., imagine that each leaf state in the FCEIC$_w$ is "merged" with the state subsumed by it and let the supervisor make the same decision whenever a state estimate is reached. By checking the transitions in the EICS, we are also able to find a run in the supervised system S^*/G leading to $V_F(y_0^e) = \inf_{r \in Run_{inf}(S^*/G)} V_{lim}(r) = \sup_{S \in \mathbb{S}} \inf_{r \in Run_{inf}(S/G)} V_{lim}(r)$ Therefore, S^* solves Problem V.3.1 (Problem V.3.2). □

From results in [98], the time complexity of the minimax search is $O(b^n)$ and the space complexity is $O(bn)$, where b is the maximum number of choices at each point in the search tree and n is the depth of the tree. For Algorithm 14, $b = \max\{2^{|E_c|}, |E_o|\}$ and $n = 2 \cdot 2^{|X|} + 1$ in the worst case, where $2^{|E_c|}$ is the maximum number of control decisions at a state and $2 \cdot 2^{|X|} + 1$ is the maximum number of states on a brunch in the FCEIC$_w$. Thus we get the complexity bound for Algorithm 14.

Given a pair of strategies $(\pi_s, \pi_e) \in \Pi_s \times \Pi_e$ and an initial run $r'_f \in Run(F_w)$, let $r_f(r'_f; \pi_s, \pi_e)$ be the run whose "prefix" is r'_f and continues under π_s and π_e, until it ends in a leaf state of the FCEIC$_w$. Formally, $r_f(r'_f; \pi_s, \pi_e) = r'_f \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} \cdots \xrightarrow{e_n} y_n^e$ where $y_n^e \in Q_{lg}^F$ $\gamma_1 = \pi_s(r'_f)$, $e_1 = \pi_e(r'_f \xrightarrow{\gamma_1} z_1^e)$ and $\gamma_i = \pi_s(r'_f \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} \cdots \xrightarrow{e_i} y_i^e)$, $e_i = \pi_e(r'_f \xrightarrow{\gamma_1} z_1^e \xrightarrow{e_1} y_2^e \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_i} z_i^e)$ for all $2 \le i \le n$. We also write $r_f(r'_f; \pi_s, \pi_e)$ as $r_f(Last(r'_f); \pi_s, \pi_e)$ since both players' decisions only depend on their current positions. Since the FCEIC$_w$ is finite, $r_f(r'_f; \pi_s, \pi_e)$ is also finite. The following proposition shows that the optimal control strategy returned by Algorithm 14 also enjoys a structural property similar to *subgame perfect equilibrium* in game theory [83] and *Bellman's optimality principle* in dynamic programming [9].

Proposition V.6.1. *Let π_s^* be an energy information state based control strategy returned by Algorithm 14, then for any initial run $r'_f \in Run(F_w)$, we have that* $\min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s,\pi_e))$.

Proof. By definition, the FCEIC$_w$ is an acyclic structure and the depth of its runs is thus bounded. So there exists a positive integer m such that from its initial state, every leaf state can be reached within m steps. Then we prove this proposition by induction on the number of steps for an initial run to reach a leaf state of the FCPEC$_w$, i.e., we show that $V_F(Last(r'_f)) = \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s,\pi_e))$.

Induction Basis: Consider the case when the last state of r'_f is a leaf states in the FCPEC$_w$. Then this proposition becomes Theorem V.6.1, thus it holds naturally.

Inductive Hypothesis: Suppose that the result holds for any r'_f that reaches leaf states within at most k steps, where $k \leq m-2$ for some integer $m > 2$. In addition, the function *Optimal* in the algorithm assigns $V_F(Last(r'_f)) = \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(r'_f;\pi_s,\pi_e))$ to the last state of r'_f.

Induction Step: Consider r'_f that reaches leaf states within at most $k+2$ steps. Suppose that $Last(r'_f) = Last_Y(r'_f) = y'^e$. We know that there exists $z^e = f_{yz}^F(y'^e,\gamma)$ for some $\gamma \in \Gamma$ and specifically, $\tilde{z}^e = f_{yz}^F(y'^e,\gamma^*)$ for $\gamma^* = \pi_s^*(y'^e,\gamma^*)$. Thus succeeding Z-state $z^e = f_{yz}^F(y'^e,\gamma)$ of y'^e reaches a leaf state within at most $k+1$ steps. By Algorithm 14, $V_F(y'^e) = V_F(\tilde{z}^e) = \max_{z^e} V_F(z^e)$. Also some f_{zy}^F transitions are defined from z^e and lead to succeeding Y-state y^e which reaches the leaf states within at most k steps. By the inductive hypothesis, $\min_{\pi_e \in \Pi_e} V_{leaf}(r_f(y^e;\pi_s^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(y^e;\pi_s,\pi_e))$ for any r'_f with $Last(r'_f) = y^e$. Again from Algorithm 14, we know that $V_F(z^e) = \min_{y^e} V_F(y^e) = \min_{y^e} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(y^e;\pi_s^*,\pi_e)) = \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(z^e;\pi_s^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(z^e;\pi_s,\pi_e))$, thus the result holds for runs whose last states reach the leaf states of the FCEIC$_w$ within $k+1$ steps. Furthermore, $V_F(y'^e) = \max_{z^e} V_F(z^e) = \max_{z^e} \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(z^e;\pi^*,\pi_e)) = \min_{\pi_e \in \Pi_e} V_{leaf}(r_f(y'^e;\pi^*,\pi_e)) = \max_{\pi_s \in \Pi_s} \min_{\pi_e \in \Pi_e}$ $V_{leaf}(r_f(y'^e;\pi_s,\pi_e))$. Therefore the result holds for $k+2$, completing the proof. □

This proposition further illustrates the structure of the optimal control strategy obtained from Algorithm 14. If the supervisor follows the strategy indicated by Algorithm 14 from its current position, then its onward decisions still constitute an optimal strategy in the remaining game, which can be viewed as a "subgame". In other words, the supervisor has no incentive to deviate from its optimal strategy given that the environment does its best to minimize the supervisor's mean payoff. As is seen from the proof, this result is due to the backward induction process of maximization and minimization in Algorithm 14. Finally, we end this section with an example.

Example V.6.1. *We revisit Example V.5.1 and find an optimal control strategy to solve Problem V.3.1 and Problem V.3.2 completely. First we obtain the EICS w.r.t. the $FCEIC_w$ in Figure V.6. For simplicity of the graph, we still preserve the state names from G and use dashed rectangles to indicate the Y-states or Z-states of the $FCEIC_w$. For example, the top green dashed rectangle corresponds to three states in the EICS, i.e. (z^e_0, x_0), (z^e_0, x_1) and (z^e_0, x_2) where $Est(I_E(z^e_0)) = \{x_0, x_1, x_2\}$. Specifically, blue and green dashed rectangles correspond to the Y-states and Z-states of the $FCEIC_w$ respectively. As is seen, the EICS is a tree-like structure whose leaf states (y^e_{1-2}, x_3), (y^e_{1-2}, x_4), (y^e_{1-3}, x_3), (y^e_{1-3}, x_4), (y^e_{1-4}, x_3), (y^e_{1-4}, x_4), (y^e_{1-5}, x_3), (y^e_{1-5}, x_4), (y^e_{2-2}, x_{12}) and (y^e_{3-2}, x_{13}) are marked in double dark blue lines.*

With the EICS built, we proceed to find the optimal control strategy by Algorithm 14. We start by calculating the values of V_{leaf} for each leaf state of the $FCEIC_w$. For example, in the EICS, there are two simple cycles between Y-states y^e_1 and y^e_{1-2}, i.e., $x_3 \xrightarrow{o_1} x_3$ and $x_3 \xrightarrow{c_1} x_5 \xrightarrow{b_1} x_7 \xrightarrow{o_1} x_3$. Then we obtain $V_{sl}(o_1) = 1$ (for x_3), $V_{sl}(c_1 b_2 o_1) = 2$, $V_{sl}(o_1) = 1$ (for x_4). Therefore, $V_F(y^e_{1-2}) = \min\{1, 2\} = 1$. Similarly, we obtain the V_T values for other leaf states in the $FCEIC_w$, which are shown in Figure V.7. Next, we apply backward induction from the leaf states until the root state to determine an optimal control strategy. In this process, we always choose to minimize at Z-states and maximize at Y-states. By Algorithm 14, we know $V_F(z^e_1) = \min\{2, \frac{2}{3}\} = \frac{2}{3}$ and $V_F(z^e_4) = V_F(z^e_6) = 1$. Thus we have the supervisor's decisions at each Y-state, which are indicated *by solid red lines in Figure V.7. An optimal supervisor enables c_1 upon observing o_1, as shown in Figure V.8. Actually, it is also optimal to disable both c_1 and c_2 at y_1^e, which yields the same maximum worst case mean payoff.*

Notice that choosing γ_4 or γ_6 at y_1^e is optimal in the sense that the environment also follows its "optimal strategy" to minimize the supervisor's limit mean payoff. If the supervisor deviates from γ_4 or γ_0 and chooses γ_1 at y_1^e, then the environment may choose o_1 at z_1^e, which leads to leaf state y_{1-5}^e and a potentially lower limit mean payoff $\frac{2}{3}$. Interestingly, if the environment also deviates from choosing o_1 from z_1^e, i.e., if it chooses o_2 or o_3, then the supervisor should choose γ_0 at y_2^e and y_3^e, which yields a better limit mean payoff for the supervisor compared with the case of choosing γ_4 at y_1^e. Those two decisions are optimal in the following "subgame" given that y_2^e or y_3^e is reached and viewed as starting points of the "subgame". This result is consistent with Proposition V.6.1.

V.7 Conclusion

We presented an approach for synthesizing partial observation supervisors that optimize the limit mean payoff of the system. The system is initialized with a certain amount of energy and its energy level dynamically changes with the occurrence of events. We considered two scenarios, i.e., optimization of the worst case mean payoff with and without the constraint of nonnegative energy level, then formulated two problems correspondingly. This chapter is the first to investigate such problems. To this end, we defined energy information states and a novel bipartite structure called First Cycle Energy Inclusive Controller (FCEIC) for each problem. Based on the FCEIC, each problem was transformed into a finite safety game with perfect information. Then both problems were solved sequentially. We first showed that winning strategies for the supervisor in the FCEIC lead to partial solutions to both problems, i.e., solutions to the so-called mean payoff decision problems. Finally we completely solved both problems by finding the optimal control strategy among partial solutions, by leveraging results from min-max games. In the future, it would be of interest to leverage the notion of the FCEIC and the solution methodology in this chapter to other

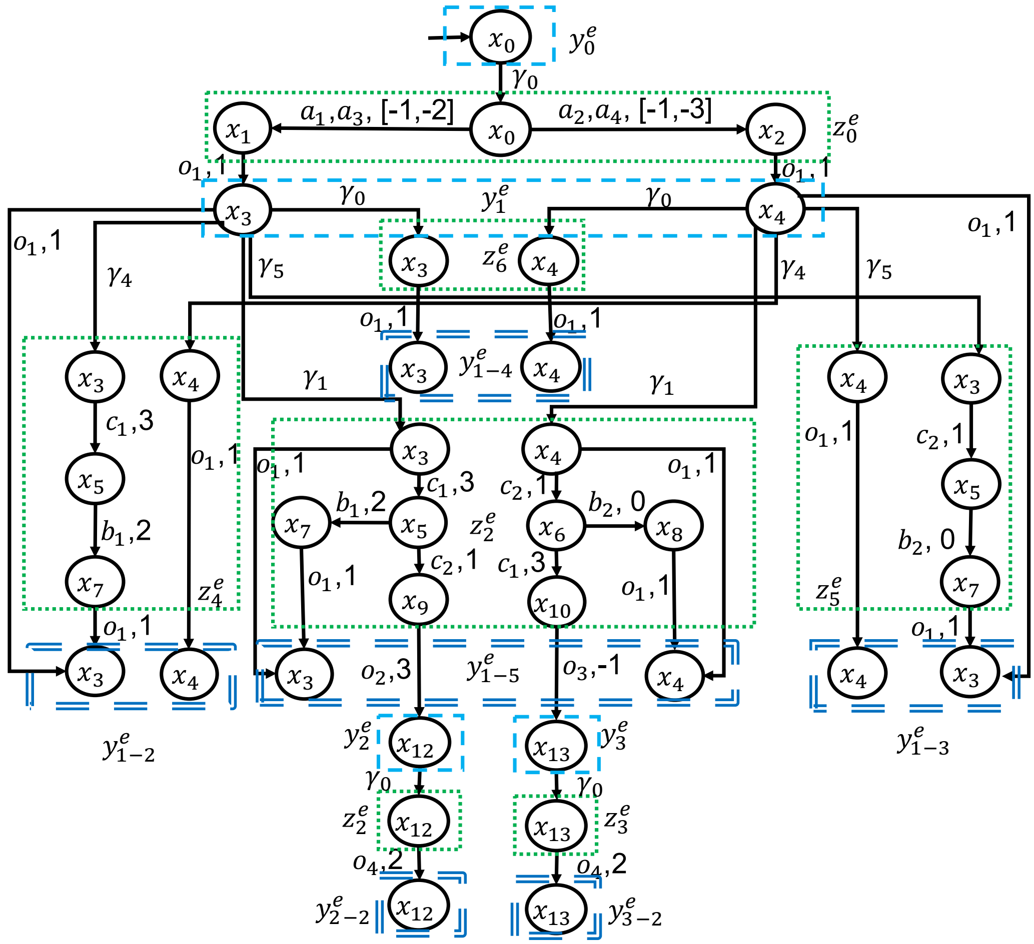

Figure V.6: The energy inter-connected system w.r.t. the FCEIC$_w$ in Example V.5.1. The blue and green dashed rectangles correspond to the Y-states and Z-states in the FCEIC$_w$, respectively. The leaf states are marked in dark blue.

quantitative performance objectives. It would also be of interest to consider other assumptions that retain decidability of the quantitative games under partial information.

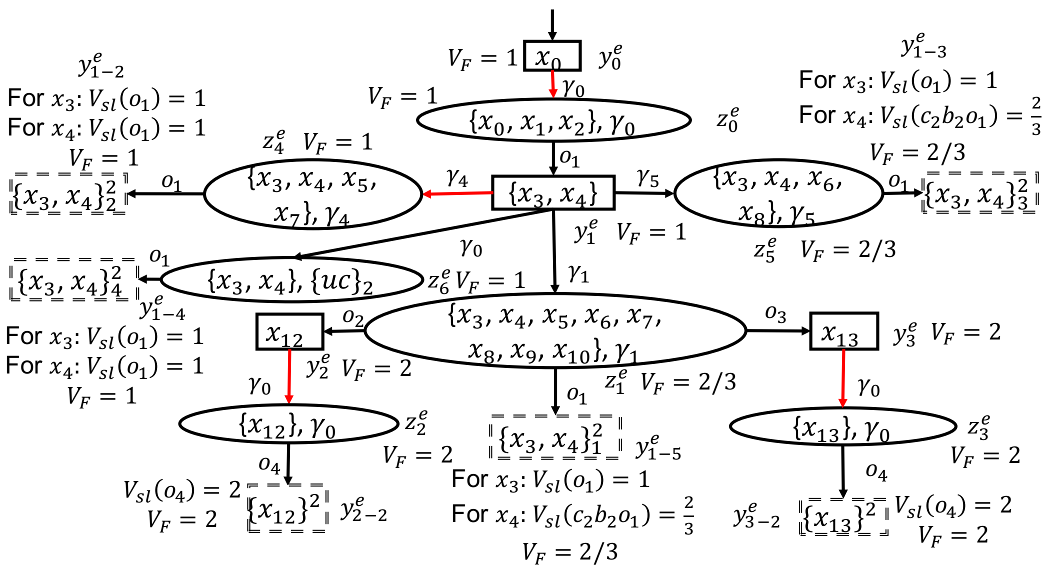

Figure V.7: Optimal decisions of the supervisor at each Y-state (indicated in red) and the V_F values for each state of the FCEIC$_w$

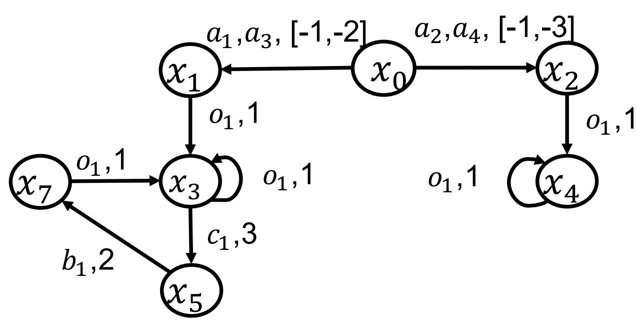

Figure V.8: An optimal supervisor solving Problem V.3.1 and Problem V.3.2

CHAPTER VI

Conclusion and Future Work

VI.1 Conclusion

In this dissertation, we solved two important problems in discrete event systems: opacity enforcement and optimal supervisory control under partial observation.

For the opacity enforcement problem, we inherited and further extended the method of insertion/edit functions originally proposed in [119, 122]. For both insertion functions and edit functions, we considered two enforcement scenarios where the intruder may or may not know the implementation of insertion/edit functions. Correspondingly, we discussed private safety and public safety for insertion/edit functions. By transforming the opacity enforcement problem to a two-player game between the insertion function and the environment, we showed that privately and publicly safe insertion functions always exist if privately safe insertion functions exist. Then we proposed the greedy-maximal criterion and developed an algorithm for synthesizing privately safe insertion functions based on the game structure called All Insertion Structure, following this criterion. As an extension, the problem of opacity enforcement by edit functions under constraints was also discussed. We defined a three-player game structure called All Edit Structure to embed all privately safe edit functions satisfying the generic edit constraints. It was also shown that nondeterministic edit functions may outperform deterministic ones in enforcing public safety.

On the other hand, we also extended the method of insertion functions to quantitative settings and discussed opacity enforcement under multiple constraints termed as energy constraints. We leveraged some results from energy games and transformed the problem into a two-player game between the supervisor and the environment. The game structure Energy Insertion Structure was defined and we synthesized insertion functions based on it. We also investigated the problem of synthesizing bounded cost rate insertion strategies. A special geometric technique called hyperplane separation was applied to solve this problem.

For optimal supervisory control, we designed supervisors to optimize the limit mean payoff of weighted discrete event systems under partial observation. These weights capture variations of a given resource, i.e., energy, consumed or replenished during the operation of the system. Two cases were considered under this framework. In the first scenario, we assumed that the system has a fixed amount of initial energy to support its operation. The goal was to design a supervisor such that the energy never gets depleted while the worst-case limit average weight of infinite event sequences is optimized. In the second scenario, we synthesized a supervisor to ensure that all limit average weights are above a certain threshold, with the worst-case value optimized. The two cases are closely related and both may be viewed as a two-player quantitative game between the supervisor and the environment, with asymmetric information and quantitative objectives. To cope with partial observation of the system, we introduced energy information states which incorporate both state information and energy information for the decision making of the supervisor. Based on this concept, we transformed the two supervisory control problems into two-player safety games with complete information and proposed a finite bipartite structure called the First Cycle Energy Inclusive Controller (FCEIC) for each problem. The supervisor synthesis algorithms in both cases were performed in a backward induction manner on the corresponding FCEIC.

VI.2 Future Work

There are several potential directions for the future work. First, we only consider enforcement of current-state opacity in Chapter II and Chapter III. It would be interesting to consider enforcement of other types of opacity, like initial-state opacity, K-step opacity and infinite-step opacity by insertion and edit functions. From the results in Chapter III, synthesizing privately and publicly safe edit functions requires building the reachability tree of the All Edit Structure, which is computationally intensive . Therefore, it would be meaningful to study alternative formulations of this problem that mitigate this issue by, e.g., relaxing the notion of public safety or by solving the problem on a reduced solution space. Also, we may extend the methodology developed in Chapters II and III to the setting of timed opacity.

Second, we may extend the secrecy obfuscation problems discussed from Chapters II to Chapter IV to the setting of active intruders. The intruder in those chapters is passive as it only observes the system's output while does not interfere with the system's operation. Suppose the system's operation is governed by some supervisor while the intruder has certain capacity to override the decisions made by the supervisor, then the obfuscation problem would become even more complicated. How to properly model such a problem and find appropriate solutions, maybe by exploring other frameworks of games, would be challenging and interesting.

Third, regarding the materials in Chapter V, investigating optimal non-blocking supervisory control under the framework of mean payoff parity games is an interesting avenue for future research. The marked states and unmarked states would be assigned with different priorities and the quantitative objective may be in terms of the mean payoff function or the total sum function. In this context, there would be a priority-based liveness criterion on the marked states together with the quantitative objective. Extending our results to stochastic settings to study the supervisory control problem under the framework of stochastic games is also of considerable interest.

Finally, it would be worthwhile to develop abstraction and compositional methods for opacity verification and enforcement, as a way to achieve more scalability in the context of modular models of discrete event systems. Some preliminary results on this problem have been reported in [77,78]; this area has great potential for future development.

BIBLIOGRAPHY

[1] M. V. S. Alves, J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira. Robust supervisory control against intermittent loss of observations. In *Proceedings of the 12th IFAC International Workshop on Discrete Event Systems*, pages 294–299, 2014.

[2] B. Aminof and S. Rubin. First-cycle games. *Information and Computation*, 254:195–216, 2017.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[4] K. R. Apt and E. Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

[5] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[6] B. Bérard, K. Chatterjee, and N. Sznajder. Probabilistic opacity for Markov decision processes. *Information Processing Letters*, 115(1):52–59, 2015.

[7] B. Bérard, O. Kouchnarenko, J. Mullins, and M. Sassolas. Opacity for linear constraint markov chains. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 28(1):83–108, 2018.

[8] B. Bérard, J. Mullins, and M. Sassolas. Quantifying opacity. *Mathematical Structures in Computer Science*, 25(Special issue 2):361–403, 2015.

[9] D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 2012.

[10] D. Berwanger and L. Doyen. On the power of imperfect information. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 2. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[11] H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1-3):365–378, 2004.

[12] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *International Conference on Formal Modeling and Analysis of Timed Systems*, volume 5215, pages 33–47, 2008.

[13] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[14] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342, 1994.

[15] Y. Brave and M. Heymann. On optimal attraction in discrete-event processes. *Information sciences*, 67(3):245–276, 1993.

[16] T. Brázdil, P. Jančar, and A. Kučera. Reachability games on extended vector addition systems with states. In *International Colloquium on Automata, Languages, and Programming*, pages 478–489. Springer, 2010.

[17] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.

[18] Véronique Bruyere, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Information and Computation*, 254:259–295, 2017.

[19] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008.

[20] J. W. Bryans, M. Koutny, and P. Y.A. Ryan. Modelling opacity using petri nets. *Electronic Notes in Theoretical Computer Science*, 121:101–115, 2005.

[21] K. Cai, R. Zhang, and W. M. Wonham. Relative observability of discrete-event systems and its supremal sublanguages. *IEEE Transactions on Automatic Control*, 60(3):659–670, 2015.

[22] L. K. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune. Detection and mitigation of classes of attacks in supervisory control systems. *Automatica*, 97:121–133, 2018.

[23] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems – 2nd Edition*. Springer, 2008.

[24] F. Cassez. The dark side of timed opacity. In *International Conference on Information Security and Assurance*, pages 21–30. Springer, 2009.

[25] F. Cassez, J. Dubreil, and H. Marchand. Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, 40(1):88–115, 2012.

[26] A. Chakrabarti, L. De Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133. Springer, 2003.

[27] K. Chatterjee. Concurrent games with tail objectives. *Theoretical Computer Science*, 388(1-3):181–198, 2007.

[28] K. Chatterjee, L. de Alfaro, and T. A. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *Journal of computer and system sciences*, 79(5):640–657, 2013.

[29] K. Chatterjee and L. Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012.

[30] K. Chatterjee, L. Doyen, and T. Henzinger. Quantitative languages. In *Computer Science Logic*, pages 385–400. Springer, 2008.

[31] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. In *International Workshop on Computer Science Logic*, volume 6, pages 287–302. Springer, 2006.

[32] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 178–187. IEEE, 2005.

[33] K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *International Conference on Concurrency Theory*, pages 115–131. Springer, 2012.

[34] K. Chatterjee and Y. Velner. Hyperplane separation technique for multidimensional mean-payoff games. *Journal of Computer and System Sciences*, 88:236–259, 2017.

[35] S. Chédor, C. Morvan, S. Pinchinat, and H. Marchand. Diagnosis and opacity problems for infinite state systems modeled by recursive tile systems. *Discrete Event Dynamic Systems: Theory and Application*, 25(1-2):271–294, 2015.

[36] J. Chen, M. Ibrahim, and R. Kumar. Quantification of secrecy in partially observed stochastic discrete event systems. *IEEE Transactions on Automation Science and Engineering*, 14(1):185–195, 2017.

[37] H. Cho and S.I. Marcus. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. *Mathematics of Control, Signals and Systems*, 2(1):47–69, 1989.

[38] P. Darondeau, H. Marchand, and L. Ricker. Enforcing opacity of regular predicates on modal transition systems. *Discrete Event Dynamic Systems: Theory and Applications*, 25(1-2):251–270, 2015.

[39] L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3):188–217, 2007.

[40] A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Toruńczyk. Energy and mean-payoff games with imperfect information. In *Computer Science Logic*, pages 260–274. Springer, 2010.

[41] J. Dubreil, P. Darondeau, and H. Marchand. Supervisory control for opacity. *IEEE Transactions on Automatic Control*, 55(5):1089–1100, 2010.

[42] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems: Theory and Applications*, 27(2):209–260, 2017.

[43] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[44] U. Fahrenberg, L. Juhl, K. G. Larsen, and J. Srba. Energy games in multiweighted automata. In *Theoretical Aspects of Computing-ICTAC*, volume 6916, pages 95–115. Springer, 2011.

[45] Y. Falcone and H. Marchand. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems: Theory and Applications*, 25(4):531–570, 2015.

[46] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 2012.

[47] V. K. Garg, R. Kumar, and S. I. Marcus. A probabilistic language formalism for stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 44(2):280–293, 1999.

[48] A. Giua and C. Seatzu. A systems theory view of Petri nets. In *Advances in Control Theory and Application*. Springer, 2007.

[49] C. Gu, X. Wang, Z. Li, and N. Wu. Supervisory control of state-tree structures with partial observation. *Info. Sciences*, 465:523–544, 2018.

[50] P. Hunter, A. Pauly, G. A. Pérez, and J.-F. Raskin. Mean-payoff games with partial observation. *Theoretical Computer Science*, 735:82–110, 2018.

[51] Paul Hunter, Arno Pauly, Guillermo A Pérez, and Jean-François Raskin. Mean-payoff games with partial observation. *Theoretical Computer Science*, 2017.

[52] R. Jacob, J.-J. Lesage, and J.-M. Faure. Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 2016.

[53] Y. Ji and S. Lafortune. Enforcing opacity by publicly known edit functions. In *Proceedings of the 56th IEEE Conference on Decision and Control*, pages 4866–4871, 2017.

[54] Y. Ji, Y.-C. Wu, and S. Lafortune. Enforcement of opacity by public and private insertion functions. *Automatica*, 93:369–378, 2018.

[55] Y. Ji, X. Yin, and S. Lafortune. Mean payoff supervisory control under partial observation. In *Proceedings of the 57th IEEE Conference on Decision and Control*, pages 3981–3987, 2018.

[56] Y. Ji, X. Yin, and S. Lafortune. Opacity enforcement by insertion functions under energy constraints. In *Proceedings of the 14th International Workshop on Discrete Event Systems*, pages 302–308, 2018.

[57] Y. Ji, X. Yin, and S. Lafortune. Enforcing opacity by insertion functions under multiple energy constraints. *Automatica*, accepted, 2018.

[58] Y. Ji, X. Yin, and S. Lafortune. Opacity enforcement using nondeterministic publicly-known edit functions. *IEEE Transactions on Automatic Control*, to appear, 2019.

[59] Y. Ji, X. Yin, and S. Lafortune. Optimal supervisory control with quantitative objectives and under partial observation. *IEEE Transactions on Automatic Control*, under review, 2018.

[60] M. Jurdziński, R. Lazić, and S. Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In *International Colloquium on Automata, Languages, and Programming*, pages 260–272, 2015.

[61] C. Keroglou and C. N. Hadjicostis. Probabilistic system opacity in discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 28(2):289–314, 2018.

[62] J. Komenda and T. Masopust. Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication. *Discrete Event Dynamic Systems: Theory and Applications*, 27(4):585–608, 2017.

[63] J. Komenda, T. Masopust, and J. H van Schuppen. Coordination control of discrete-event systems revisited. *Discrete Event Dynamic Systems: Theory and Applications*, 25(1-2):65–94, 2015.

[64] R. Kumar and V. K. Garg. Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, 46(4):593–606, 2001.

[65] R. Kumar and V.K. Garg. Optimal supervisory control of discrete event dynamical systems. *SIAM Journal on Control and Optimization*, 33(2):419–439, 1995.

[66] R. Kumar and V.K. Garg. *Modeling and control of logical discrete event systems*, volume 300. Springer Science & Business Media, 2012.

[67] S. Lafortune, F. Lin, and C.N. Hadjicostis. On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45:257–266, 2018.

[68] B. Lennartson and M. Noori-Hosseini. Visible bisimulation equivalencea unified abstraction for temporal logic verification. In *Proceedings of the 14th IFAC International Workshop on Discrete Event Systems*, pages 400–407, 2018.

[69] A. Levy. *Basic set theory*, volume 13. Courier Corporation, 2002.

[70] F. Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011.

[71] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information sciences*, 44(3):199–224, 1988.

[72] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[73] H. Marchand, O. Boivineau, and S. Lafortune. On optimal control of a class of partially observed discrete event systems. *Automatica*, 38(11):1935–1943, 2002.

[74] T. Masopust and X. Yin. Complexity of detectability, opacity and A-diagnosability for modular discrete event systems. *Automatica*, 101:290–295, 2019.

[75] L. Mazaré. Using unification for opacity properties. *Proceedings of the 4th IFIP WG1*, 7:165–176, 2004.

[76] R. Milner. *Communication and Concurrency*, volume 84. Prentice Hall New York, 1989.

[77] S. Mohajerani, Y. Ji, and S. Lafortune. Efficient synthesis of edit functions for opacity enforcement using bisimulation-based abstractions. In *2018 IEEE Conference on Decision and Control*, pages 4849–4854, 2018.

[78] S. Mohajerani, Y. Ji, and S. Lafortune. Compositional and abstraction-based approach for synthesis of edit functions for opacity enforcement. *IEEE Transactions on Automatic Control*, under review, 2019.

[79] J. Mullins and M. Yeddes. Opacity with orwellian observers and intransitive non-interference. In *Proceedings of the 12th International Workshop on Discrete Event Systems*, pages 344–349, 2014.

[80] C. St. J. A. Nash-Williams. On well-quasi-ordering finite trees. In *Mathematical Proceedings of the Cambridge Philosophy Society*, volume 59, pages 833–835. Cambridge University Press, 1963.

[81] M. Noori-Hosseini, B. Lennartson, and C. Hadjicostis. Compositional visible bisimulation abstraction applied to opacity verification. In *Proceedings of the 14th IFAC International Workshop on Discrete Event Systems*, pages 434–441, 2018.

[82] M. Noori-Hosseini, B. Lennartson, and C. Hadjicostis. Incremental observer reduction applied to opacity verification and synthesis. *arXiv preprint arXiv:1812.08083*, 2018.

[83] M.J. Osborne and A. Rubinstein. *A course in game theory*. Massachusetts Institute of Technology press, 1994.

[84] V. Pantelic and M. Lawford. Optimal supervisory control of probabilistic discrete event systems. *IEEE Transactions on Automatic Control*, 57(5):1110–1124, 2012.

[85] Kevin M Passino and Panos J Antsaklis. Optimal stabilization of discrete event systems. In *29th IEEE Conference on Decision and Control*, pages 670–671. IEEE, 1990.

[86] K.M. Passino and P.J. Antsaklis. On the optimal control of discrete event systems. In *Proceedings of the 28th IEEE Conference on Decision and Control*, pages 2713–2718. IEEE, 1989.

[87] G. A Pérez. The fixed initial credit problem for partial-observation energy games is ack-complete. *Information Processing Letters*, 118:91–99, 2017.

[88] S. Pruekprasert and T. Ushio. Optimal stabilizing controller for the region of weak attraction under the influence of disturbances. *IEICE Transactions on Information and Systems*, 99(6):1428–1435, 2016.

[89] S. Pruekprasert and T. Ushio. Optimal stabilizing supervisor of quantitative discrete event systems under partial observation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 99(2):475–482, 2016.

[90] S. Pruekprasert and T. Ushio. Supervisory control of partially observed quantitative discrete event systems for fixed-initial-credit energy problem. *IEICE Transactions on Information and Systems*, 100(6):1166–1171, 2017.

[91] S. Pruekprasert, T. Ushio, and T. Kanazawa. Quantitative supervisory control game for discrete event systems. *IEEE Transactions on Automatic Control*, 61(10):2987–3000, 2016.

[92] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[93] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[94] A. Ray, J. Fu, and C. Lagoa. Optimal supervisory control of finite state automata. *International Journal of Control*, 77(12):1083–1100, 2004.

[95] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.

[96] S. L. Ricker, T.F. Lidbetter, and H. Marchand. Inferencing and beyond: further adventures with parity-based architectures for decentralized discrete-event systems. In *Proceedings of 20th IFAC World Congress*, pages 13447–13452, 2017.

[97] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE transactions on automatic control*, 37(11):1692–1708, 1992.

[98] S. J. Russell and P. Norvig. *Artificial Intelligence: a modern approach– 3nd Edition*. Prentice Hall, 2009.

[99] A. Saboori and C. N. Hadjicostis. Verification of infinite-step opacity and complexity considerations. *IEEE Transactions on Automatic Control*, 57(5):1265–1269, 2012.

[100] A. Saboori and C. N. Hadjicostis. Current-state opacity formulations in probabilistic finite automata. *IEEE Transactions on Automatic Control*, 59(1):120–133, 2014.

[101] A. Saboori and C.N. Hadjicostis. Opacity-enforcing supervisory strategies via state estimator constructions. *IEEE Transactions on Automatic Control*, 57(5):1155–1165, 2012.

[102] A. Saboori and C.N. Hadjicostis. Verification of initial-state opacity in security applications of discrete event systems. *Information Sciences*, 246:115–132, 2013.

[103] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[104] K. W. Schmidt. Optimal supervisory control of discrete event systems: Cyclicity and interleaving of tasks. *SIAM Journal on Control and Optimization*, 53(3):1425–1439, 2015.

[105] C. Seatzu, J. H. van Schuppen, and M. Silva. *Control of discrete-event systems-Automata and Petri Net perspectives*. Springer, 2013.

[106] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, 36(2):488–541, 1998.

[107] S. Shu and F. Lin. Supervisor synthesis for networked discrete event systems with communication delays. *IEEE Transactions on Automatic Control*, 60(8):2183–2188, 2015.

[108] R. Su. Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. *Automatica*, 94:35–44, 2018.

[109] R. Su, J. H. Van Schuppen, and J. E. Rooda. The synthesis of time optimal supervisors by using heaps-of-pieces. *IEEE Transactions on Automatic Control*, 57(1):105–118, 2012.

[110] S. Takai and Y. Oka. A formula for the supremal controllable and opaque sublanguage arising in supervisory control. *SICE Journal of Control, Measurement, and System Integration*, 1(4):307–311, 2008.

[111] S. Takai and T. Ushio. Effective computation of an L_m (G)-closed, controllable, and observable sublanguage arising in supervisory control. *Systems & Control Letters*, 49(3):191–200, 2003.

[112] Y. Tong, Z. Li, C. Seatzu, and A. Giua. Decidability of opacity verification problems in labeled petri net systems. *Automatica*, 80:48–53, 2017.

[113] Y. Tong, Z. Li, C. Seatzu, and A. Giua. Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, 62(6):2823–2837, 2017.

[114] Y. Tong, Z. Li, C. Seatzu, and A. Giua. Current-state opacity enforcement in discrete event systems under incomparable observations. *Discrete Event Dynamic Systems: Theory and Applications*, 28(2):161–182, 2018.

[115] T. Ushio and S. Takai. Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions. *IEEE Trans. on Auto. Control*, 61(3):799–804, 2016.

[116] Y. Velner, K. Chatterjee, L. Doyen, T. A. Henzinger, A. Rabinovich, and J.-F. Raskin. The complexity of multi-mean-payoff and multi-energy games. *Information and Computation*, 241:177–196, 2015.

[117] L. Wang, N. Zhan, and J. An. The opacity of real-time automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2845–2856, 2018.

[118] W. M. Wonham and K. Cai. *Supervisory control of discrete-event systems*. Springer, 2018.

[119] Y.-C. Wu and S. Lafortune. Synthesis of insertion functions for enforcement of opacity security properties. *Automatica*, 50(5):1336–1348, 2014.

[120] Y.-C. Wu and S. Lafortune. Synthesis of opacity-enforcing insertion functions that can be publicly known. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 3506–3513, 2015.

[121] Y.-C. Wu and S. Lafortune. Synthesis of optimal insertion functions for opacity enforcement. *IEEE Transactions on Automatic Control*, 61(3):571–584, 2016.

[122] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune, and S. A. Seshia. Synthesis of obfuscation policies to ensure privacy and utility. *Journal of Automated Reasoning*, (1):107–131, 2018.

[123] X. Yin. Supervisor synthesis for mealy automata with output functions: A model transformation approach. *IEEE Transactions on Automatic Control*, 62(5):2576–2581, 2017.

[124] X. Yin and S. Lafortune. A general approach for solving dynamic sensor activation problems for a class of properties. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 3610–3615, 2015.

[125] X. Yin and S. Lafortune. Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(5):1239–1254, 2016.

[126] X. Yin and S. Lafortune. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(8):2140–2154, 2016.

[127] X. Yin and S. Lafortune. A new approach for the verification of infinite-step and K-step opacity using two-way observers. *Automatica*, 80:162–171, 2017.

[128] X. Yin and S. Lafortune. Synthesis of maximally-permissive supervisors for the range control problem. *IEEE Transactions on Automatic Control*, 62(8):3914–3929, 2017.

[129] X. Yin and S. Lafortune. Synthesis of maximally permissive nonblocking supervisors for the lower bound containment problem. *IEEE Transactions on Automatic Control*, 63(12):4435–4441, 2018.

[130] X. Yin and S. Lafortune. A general approach for optimizing dynamic sensor activations for discrete event systems. *Automatica*, to appear, 2019.

[131] X. Yin and S. Li. Verification of opacity in networked supervisory control systems with insecure control channels. In *Proceedings of 2018 IEEE Conference on Decision and Control (CDC)*, pages 4851–4856, 2018.

[132] X. Yin, Z. Li, W. Wang, and S. Li. Infinite-step opacity and K-step opacity of stochastic discrete-event systems. *Automatica*, 99:266–274, 2019.

[133] B. Zhang, S. Shu, and F. Lin. Maximum information release while ensuring opacity in discrete event systems. *IEEE Transactions on Automation Science and Engineering*, 12(3):1067–1079, 2015.

[134] K. Zhang, X. Yin, and M. Zamani. Opacity of nondeterministic transition systems: A (bi) simulation relation approach. *IEEE Transactions on Automatic Control*, to appear, 2019.

[135] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.