Automatica 108 (2019) 108476

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Enforcing opacity by insertion functions under multiple energy constraints[☆]

Yiding Ji[a], Xiang Yin[b,*], Stéphane Lafortune[a]

[a] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA
[b] Department of Automation, Shanghai Jiao Tong University, Shanghai, China

ARTICLE INFO

Article history:
Received 30 July 2018
Received in revised form 6 June 2019
Accepted 11 June 2019
Available online xxxx

Keywords:
Discrete event systems
Opacity enforcement
Insertion function
Partial observation
Quantitative games

ABSTRACT

This paper investigates the enforcement of opacity by insertion functions subject to multiple quantitative constraints capturing resource or energy limitations. There is a malicious intruder attempting to infer secrets of the system from its observations. To prevent the disclosure of secrets, the insertion function inserts fictitious events to the output of the system to obfuscate the intruder. The system is initialized with several types of resources, referred to as energy. The energy is consumed or replenished with event occurrences while always consumed with event insertions. The insertion function must enforce opacity while ensuring that each type of resource is never depleted. This problem is then reduced to a two-player game between the insertion function and the system (environment), with properly defined objectives. A game structure called the Energy Insertion Structure, denoted by *EIS* is proposed, which provably contains solutions to the energy constrained opacity enforcement problem. Then we further study the bounded cost rate insertion problem on the insertion function's winning region of *EIS*, which requires that the long run average rate of insertion cost be bounded. This problem is formulated as a multidimensional mean payoff game and a special method called hyperplane separation technique is applied to efficiently solve it.

1. Introduction

Opacity is an information-flow based property that characterizes whether the secrets of a system can be inferred by an outside intruder with malicious purposes. The outside intruder is typically modeled as an observer with knowledge of the structure of the system; its intention is to infer system secrets by observing system outputs. The system is opaque if the intruder is never able to *unambiguously* determine any of the system secrets from its observations.

Opacity has received significant attention in the context of Discrete Event Systems (DES). Different notions of opacity have been proposed and studied for finite-state automata, e.g., language-based opacity (Lin, 2011), current-state opacity (Saboori & Hadjicostis, 2007), initial-state opacity (Saboori & Hadjicostis, 2013), K-step opacity (Yin & Lafortune, 2017) and infinite-step opacity (Zhang, Yin, & Zamani, 2019). Opacity has also been discussed in other models, like infinite state systems (Chédor, Morvan, Pinchinat, & Marchand, 2015), Petri nets (Tong, Li, Seatzu, & Giua, 2017), modular systems (Masopust & Yin, 2019) and timed systems (Cassez, 2009). Opacity under the so-called Orwellian observation is studied in Mullins and Yeddes (2014). Additionally, many works investigate opacity quantitatively in stochastic settings, e.g., Bérard, Mullins, and Sassolas (2015), Chen, Ibrahim, and Kumar (2017), Keroglou and Hadjicostis (2018), Yin, Li, Wang, and Li (2019). The review paper (Jacob, Lesage, & Faure, 2016) provides a comprehensive summary of research topics on opacity in DES.

Violations of opacity give rise to the problem of opacity enforcement, see, e.g., Barcelos and Basilio (2018), Falcone and Marchand (2015), which has been investigated under various mechanisms. Supervisory control can be used to disable non-opaque behaviors, thereby preventing disclosure of secrets (Dubreil, Darondeau, & Marchand, 2010; Takai & Oka, 2008; Tong, Li, Seatzu, & Giua, 2018; Yin & Lafortune, 2016; Zhang, Shu, & Lin, 2015). Another popular framework is sensor activation (Cassez, Dubreil, & Marchand, 2012; Yin & Lafortune, 2019), which dynamically changes the observability of certain events but does not intervene with the operation of the system. Differently from these approaches, Ji, Wu and Lafortune (2018) study opacity enforcement using insertion functions, which may insert fictitious events

[☆] Research supported in part by the US National Science Foundation under grant CNS-1738103, also by National Natural Science Foundation of China under grants 61803259 and 61833012. The material in this paper was partially presented at the 14th International Workshop on Discrete Event Systems, May 30–June 1, 2018, Sorrento Coast, Italy. This paper was recommended for publication in revised form by Associate Editor Christoforos Hadjicostis under the direction of Editor Christos G. Cassandras.

[*] Corresponding author.
E-mail addresses: jiyiding@umich.edu (Y. Ji), yinxiang@sjtu.edu.cn (X. Yin), stephane@umich.edu (S. Lafortune).

https://doi.org/10.1016/j.automatica.2019.06.028
0005-1098/© 2019 Elsevier Ltd. All rights reserved.

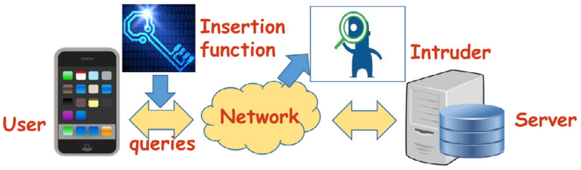

Fig. 1. Location-based service and insertion mechanism.

into the system's output to modify the intruder's observation for obfuscation purposes. This method is further generalized to edit functions (Ji, Yin, & Lafortune, 2019; Mohajerani, Ji, & Lafortune, 2018) which may erase events from the output of the system, along with event insertions.

All the above works concentrate on logical properties of opacity enforcement. However, in many applications, the execution of system events may gain or consume certain types of resources of the system, which we refer to as "*energy*". Besides, secrecy obfuscation may also consume some types of resources so that some strategies may be preferred due to lower costs. Those resources may be interpreted as budget for insertion of fictitious events, processing time, storage space, power supply, and so forth. Motivated by this practical situation, it is meaningful to investigate opacity enforcement under quantitative constraints. We assume that the system has several types of resources whose amounts are all fixed. The system's energy 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.

In this work, we consider opacity enforcement by leveraging the technique of insertion functions (Ji, Wu, et al., 2018) 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 Fig. 1. Since inserting queries may cost certain resources like electricity, bandwidth and money, 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 not violated. In addition, the resources should not be consumed too sharply so that the insertion functions work economically, i.e., the rate of insertion cost should be bounded from above.

These requirements lead us to study opacity enforcement by insertion functions with *multiple quantitative objectives*. This problem is discussed under *imperfect information* due to the insertion function's partial observation of the system, i.e., it is only aware of the occurrence of observable events. The insertion function aims to enforce opacity under the constraint that each type of resource of the system never drops below zero, for all possible system behaviors (worst-case analysis). 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, denoted by *EIS*. The insertion function plays by inserting events, which consumes resources, while the system plays by executing events, which consumes or gains resources. So the system's resource levels dynamically change, which are reflected in *EIS*.

Based on *EIS*, we first find the strategies of the insertion function, which enforce opacity while not violate the energy level constraints. Among them, we are particularly interested in the strategies working in an "economical" way. In other words, there should exist an upper bound for the rate of insertion cost so that only a reasonable amount of resource is consumed per step of insertion. Motivated by this requirement, we further formulate the bounded insertion cost rate problem as a multidimensional mean payoff game and solve it by leveraging the *hyperplane separation technique* originally proposed in Chatterjee and Velner (2017).

Our work is inspired by some results on quantitative two-player games in theoretical computer science, specifically, energy games and mean payoff games (Apt & Grädel, 2011; Ehrenfeucht & Mycielski, 1979). 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 ACK-complete (Pérez, 2017) with fixed amount of initial energy, while mean payoff games are in general undecidable (Degorre, Doyen, Gentilini, Raskin, & Toruńczyk, 2010). Another generalization is multidimensional game (Chatterjee & Velner, 2017), where both players have several quantitative objectives. The above works also inspired the work (Pruekprasert & Ushio, 2017), which studies supervisory control for DES using energy games with partial observation. We adapt some methodology from Pruekprasert and Ushio (2017) to the different problem of opacity enforcement by obfuscation. To the best of our knowledge, this paper is the first to investigate opacity enforcement under multiple quantitative objectives.

The rest of this paper is organized as follows. Section 2 describes our system model. Section 3 formulates the energy constrained opacity enforcement problem. Section 4 introduces *EIS* and presents its construction algorithm. Section 5 solves the energy constrained opacity enforcement problem based on *EIS*. Section 6 formulates the bounded cost rate insertion strategy synthesis problem and solves it by the hyperplane separation technique. Finally, Section 7 concludes the paper.

A preliminary version of this paper appears in Ji, Yin and Lafortune (2018) and the improvement is three-fold. First, Ji, Yin, et al. (2018) only show the soundness of obtaining insertion functions from *EIS*, while this work also shows the completeness. Second, we extend the one-dimensional quantitative objective in Ji, Yin, et al. (2018) to the multidimensional case. Finally, we solve the bounded cost rate insertion strategy synthesis problem, which was not treated in Ji, Yin, et al. (2018).

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 (Cassandras & Lafortune, 2008) 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 ith 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. A run is *initial* if its initial state is the initial state of the system. Also, a run forms a *cycle* if $x_1 = x_n$ and the cycle is *simple* if $\forall i, j \in \{1, 2, \ldots, n-1\}$, $i \neq j \Rightarrow x_i \neq x_j$. If r is a cycle, 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 ith 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 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: $UR(q) = \{x' \in X : \exists x \in q, \exists s \in E_{uo}^*, \text{ s.t. } f(x, s) = x'\}$. 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* of the system, which is a subset of X.

3. Problem formulation

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

Definition 1 (*Current-State Opacity (CSO)*)**.** Given system G, projection P, and secret state set X_S, G is CSO if $\forall t \in L_S := \{t \in \mathcal{L}(G) : f(x_0, t) \in X_S\}$, $\exists t' \in L_{NS} := \{t \in \mathcal{L}(G) : f(x_0, t) \in (X \setminus X_S)\}$ such that $P(t) = P(t')$.

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 Ji, Wu, et al. (2018). 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 Ji, Wu, et al. (2018) for more details of this concept.

Definition 2 (*Insertion Function*)**.** An insertion function is defined as: $f_{in} : E_o^* \times E_o \to E_o^* E_o$ such that for $l \in E_o^*$ and $e_o \in E_o$, $f_{in}(l, e_o) = s_l e_o$ where $s_l \in E_o^*$.

By definition, the insertion function inserts s_l before the next observable event e_o given that l has been observed, then it outputs $s_l e_o$. It is likely that s_l is ϵ when no event is inserted. An insertion function f_{in} may be encoded as an input/output (I/O) automaton $IA = (X_{ia}, E_o, E_o^+, \delta_{ia}, \delta_{oa}, x_{ia,0})$. 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; $\delta_{oa} : X_{ia} \times E_o \to E_o^+$ is the output function such that $\delta_{oa}(x_{ia}, e_o) = s_l e_o$ where $\delta_{ia}(x_{ia}, e_o)!$ and $\delta_{ia}(x_{ia,0}, s) = x_{ia}$, if $f_{in}(s, e_o) = s_l e_o$; $x_{ia,0} \in X_{ia}$ is the initial state.

We also define a string-based version of f_{in} and with a slight abuse of notation, denote it by f_{in} as well (it will be clear from the argument which form of f_{in} is being considered): $f_{in}(\epsilon) = \epsilon$ and $f_{in}(le_o) = f_{in}(l)f_{in}(l, e_o)$.

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 1. *Given* $s = \xi_0 e_0 \cdots \xi_{n-1} e_{n-1} \xi_n \in \mathcal{L}(G)$ *where* $\forall j \leq n$, $\xi_j \in E_{uo}^*$ *and* $e_j \in E_o$, *if* $f_{in}(e_0 e_1 \cdots e_{j-1}, e_j) = \theta_j e_j$ *where* $\forall j \leq n$, $\theta_j \in E_o^*$, *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.

Next, we present the notion of *private safety* from Ji, Wu, et al. (2018), which indicates that every string in $P[\mathcal{L}(G)]$ is mapped to a safe string under certain insertion choices.

Definition 3 (*Private Safety*)**.** Given system G with projection P and safe language L_{safe}, insertion function f_{in} is privately safe if $\forall s \in P[\mathcal{L}(G)]$, $f_{in}(s) \in L_{safe}$.

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 ith 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 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 $s_{f_{in}}$ if s is mapped to s' by f_{in}. Hence, $V_m(s, s_{f_{in}})$ is the energy level of the system after string s is modified by insertion function f_{in}.

Given a non-opaque system G with initial energy vector v_0, we aim to design an insertion function f_{in} 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., $\forall s \in P[\mathcal{L}(G)]$, $V_m(s, s_{f_{in}}) \geq \overrightarrow{0}$. Since insertion always costs energy, we made Assumption 1 earlier to ensure some energy margins for the insertion function. We now formally formulate the energy constrained opacity enforcement problem.

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

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 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 1.

4. Energy insertion structure

In this section we define *energy information states* and *Energy Insertion Structure*, which is denoted by *EIS*. By introducing these concepts, we transform Problem 1 into a reachability game with perfect information between the insertion functions and environment. Then we solve Problem 1 on *EIS*.

4.1. Building the verifier

We first review the concept of *verifier* proposed in Ji, Wu, et al. (2018). 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* (Ji, Wu, et al., 2018). 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; $\delta_{sl} : X_f \times E_o \to X_f$ 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* (Ji, Wu, et al., 2018) 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 4 (*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 $x_v \in X_v$.

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 5 (*Energy Information State*). Given G, an energy information state is: $q^e = ((x^d, x^f), [v(1), \ldots, v(|x^f|)]) \in X_v \times \cup_{i=1}^{|X|}\mathbb{Z}^{k\times|i|}$. Let $I(q^e)$ and $E_L(q^e)$ denote the state estimate and energy level components, respectively, so $q^e = (I(q^e), E_L(q^e))$.

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* $h_{q^e} : X \to \mathbb{Z}^k$. Specifically, for $q^e \in Q^E$ where $I(q^e) = (x^d, x^f) \in X_v$, we have $E_L(q^e) = \{h_{q^e}(x) : x \in x^f\}$. We usually put $E_L(q^e)$ in a column vector's form: $[h_{q^e}(x_1), \ldots, h_{q^e}(x_{|x^f|})]$. By convention, elements in $E_L(q^e)$ are placed in an increasing order w.r.t. state names in x^f. Our definition is inspired by the belief function in Degorre et al. (2010) and the observation function in Pruekprasert and Ushio (2017). In the following discussion, we use $h_{q^e}^{(i)}(x)$ to denote the ith 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 $v_1 = [v_1(1), v_1(2), \ldots, v_1(k)]$, $v_2 = [v_2(1), v_2(2), \ldots, v_2(k)] \in \mathbb{Z}^k$, 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 $m_1 = [v_1, v_2, \ldots, v_n]$, $m_2 = [v_1', v_2', \ldots, v_n'] \in \mathbb{Z}^{k\times n}$, 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 $I(q^e) = (x^d, x^f)$, $h_{q^e}(x) \geq \overrightarrow{0}$. We define an order $\preccurlyeq$ over the set of energy information states: for $q_1^e, q_2^e \in Q^E$, $q_1^e \preccurlyeq q_2^e$ if $I(q_1^e) = I(q_2^e)$ and $E_L(q_1^e) \leq E_L(q_2^e)$. We also say that q_2^e *subsumes* q_1^e if $q_1^e \preccurlyeq q_2^e$, i.e., q_1^e and q_2^e share the same verifier state component but the energy level vector of q_2^e is no less than that of q_1^e at every possible current state in $I(q_2^e)$. By Dickson's lemma (see Levy (2002)), the order $\leq$ on $\mathbb{N}^m$ is a *well-quasi-ordering* for any $m \in \mathbb{N}$. In addition, 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 (Nash-Williams, 1963), 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 $q_1^e, q_2^e \cdots \in Q^E$, $\exists i, j \in \mathbb{N}$, s.t. $i < j$ and $q_i^e \preccurlyeq q_j^e$.

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 $I(I_E(q^{ae})) = (x^d, x^f)$, $h_{q^{ae}}(x) \geq \overrightarrow{0}$. 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) $I(q^e) = (x'^d, x'^f) = \delta_{vs}(\delta_{vd}((x^d, x^f), \theta), e_o)$ where $I(I_E(q^{ae})) = (x^d, x^f)$; (ii) $\forall x' \in x'^f$, $\forall 1 \leq i \leq k$, $h_{q^e}^{(i)}(x') = \min_{\xi \in E_{uo}^*}\{h_{q^{ae}}^{(i)}(x) + \omega^{(i)}(e_o) + \omega^{(i)}(\xi) + \omega_{in}^{(i)}(\theta) : \exists x \in x^f,\ \text{s.t.}\ f(x, e_o\xi) = x'\}$.

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 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: $\rho = y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$ where $\forall i \leq n, \theta_i \in E_o^*, e_i \in E_o, y_i^e \in Q^E, z_i^e \in Q^E \times E_o$, z_i^e is an e_i-execution successor of y_i^e and y_{i+1}^e is a (θ_i, e_i)-insertion successor of z_i^e. Such a sequence may be finite or infinite.

Lemma 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 $I(y_i^e) = (x_i^d, x_i^f)$ for all $1 \leq i < n$ and let $l = e_1e_2\cdots e_{n-1}$ and $l' = \theta_1e_1\cdots\theta_{n-1}e_{n-1}$, then $\delta_d(x_1^d, l') = x_n^d$ in $Obs_d(G)$ and $\delta(x_1^f, l) = x_n^f$ in $Obs_f(G)$.*

Proof. By induction. First, consider $y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e$. Since z_1^e is an e_1-execution successor of y_1^e 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 $\delta_d(x_1^d, \theta_1e_1) = x_2^d$ 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 $\delta_d(x_k^d, \theta_ke_k) = x_{k+1}^d$ and $\delta(x_k^f, e_k) = x_{k+1}^f$. Combining the inductive hypothesis, we know $\delta_d(x_1^d, \theta_1e_1\cdots\theta_ke_k) = x_{k+1}^d$ 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 1 illustrates that in an insertion-execution sequence, the "original string" $e_1e_2\cdots e_{n-1}$ before insertion is defined in the feasible observer and the string $\theta_1e_1\cdots\theta_{n-1}e_{n-1}$ 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 jth prefix of ρ. Also we let $V_m^{(i)}(s, s')$ denote the ith component of the k-dimensional vector $V_m(s, s')$.

Theorem 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 $I(y_i^e) = (x_i^d, x_i^f)$ for all $1 \leq i \leq n$ and let $l = e_1\cdots e_{n-1}$, then $\forall x \in x_n^f$, $\forall 1 \leq i \leq k$, $h_{y_n^e}^{(i)}(x) = \min_s\{V_m^{(i)}(s, s') : \exists x' \in x_1^f, s \in P^{-1}(l),\ \text{s.t.}\ f(x', s) = x, \delta_d(x_1^d, P(s')) = x_n^d\}$ where string s is mapped to s' following Convention 1 under insertions indicated by ρ.*

Proof. Proof by induction on the length of l. Suppose $s = \xi_1e_1\cdots\xi_{n-1}e_{n-1}\xi_n$, $P(s) = l = e_1\cdots e_n$ and s is mapped to $s' = \xi_1\theta_1e_1\cdots\xi_n\theta_ne_n\xi_{n+1}$ where $\theta_j \in E_o^*$ and $P(s') = \theta_1e_1\cdots\theta_ne_n = l'$. Let $l_j = e_1\cdots e_j$ and $l'_j = \theta_1e_1\cdots\theta_je_j$ be the jth prefix of l and l', respectively. Let $l_0 = \epsilon$ and $s_j = \xi_1e_1\cdots\xi_{j-1}e_j\xi_{j+1}$, 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: $n = 0$, 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, $\delta_{vd}((x_j^d, x_j^f), \theta_j) = (x_{j+1}'^d, x_{j+1}'^f)$ and $\delta_{vs}(\delta_{vd}((x_{j+1}'^d, x_{j+1}'^f), \theta_j), e_j) = (x_{j+1}^d, x_{j+1}^f)$ 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, $\forall x' \in x_{j+1}^f$, $\forall 1 \leq i \leq j$, $h_{y_{j+1}^e}^{(i)}(x') = \min_{\xi_{j+1} \in E_{uo}^*}\{h_{y_j^e}^{(i)}(x) + \omega^{(i)}(e_j) + \omega^{(i)}(\xi_{j+1}) + \omega_{in}^{(i)}(\theta_j) : \exists x \in x_j^f,\ \text{s.t.}\ f(x, e_j\xi_{j+1}) = x'\}$. From the inductive hypothesis, we have $h_{y_{j+1}^e}^{(i)}(x') = \min_{s_{j-1}} \min_{\xi_{j+1} \in E_{uo}^*}\{V_m^{(i)}(s_{j-1}, s'_{j-1}) + \omega^{(i)}(e_j) + \omega^{(i)}(\xi_{j+1}) + \omega_{in}^{(i)}(\theta_j) : \exists x'' \in x_1^f, x \in x_j^f,\ \text{s.t.}\ f(x'', s_{j-1}) = x, \delta_d(x_1^d, P(s'_{j-1})) = x_j^d, f(x, e_j\xi_{j+1}) = x'\}$. That is, $h_{y_{j+1}^e}^{(i)}(x') = \min_{s_j}\{V_m^{(i)}(s_j, s'_j) : \exists x'' \in x_1^f, s_j \in P^{-1}(l_j), \text{s.t.}\ f(x'', s_j) = x', \delta_d(x_1^d, P(s'_j)) = x_{j+1}^d\}$. Thus the result holds when $n = j$, completing the whole proof. □

Given an energy information state $y^e \in Q^E$, for every $x \in x^f$ where $I(y^e) = (x^d, x^f)$, 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.

4.3. Building the energy insertion structure

We now formally define *EIS* by construction in Algorithm 1. *EIS* is a two-player game structure which reflects the update of energy and augmented energy information states with event insertion and execution. It is of the form: $EIS = (Q_Y^E, Q_Z^E, E_o, f_{yz}^E, f_{zy}^E, y_0^e, v_0, Q_I^E)$ where $Q_Y^E \subseteq Q^E$ is the set of energy information states; $Q_Z^E \subseteq Q^E \times E_o$ is the set of augmented energy information states; $f_{yz}^E : Q_Y^E \times E_o \to Q_Z^E$ is the transition function from Q_Y^E states to Q_Z^E states; $f_{zy}^E : Q_Z^E \times E_o^* \to Q_Y^E$ is the transition function from

Q_Z^E states to Q_Y^E states; E_o is the set of observable events; $y_0^e \in Q_Y^E$ is the initial state; $v_0 \in \mathbb{N}^k$ is the initial energy vector; and Q_l^E is the set of leaf states. We call a Q_Y^E state as Y-state and a Q_Z^E state as Z-state. A Z-state z^e is *deadlocking* if $\nexists \theta \in E_o^*$, s.t. $f_{zy}^E(z^e, \theta)!$. Deadlocking Z-states are undesirable and will be pruned away in constructing *EIS*.

Algorithm 1: Construction of *EIS*

```
Input: $Obs(G)$, $G_v$, $v_0$
Output: $EIS = (Q_Y^E, Q_Z^E, E, f_{yz}^E, f_{zy}^E, E_o, y_0^e, v_0, Q_l^E)$
1: $Q_Y^E = \{y_0^e\}$ where $I(y_0^e) = (x_{obs,0}, x_{obs,0})$, $\forall x \in x_{obs,0}$, $\forall i \leq k$, $h_{y_0^e}^i(x) = \min_{\xi \in E_{uo}^*}\{V^i(\xi) : f(x_0, \xi) = x\}$, and $Q_Z^F = \emptyset$, $Q_l^E = \emptyset$;
2: $EIS_{pre} = DoDFS(y_0^e, Obs(G), G_v)$;
3: $EIS = Prune(EIS_{pre})$;
4: procedure $DoDFS(y^e, Obs(G), G_v)$
5:     for $e_o \in E_o$, s.t. $\delta(x^f, e_o)!$ in $Obs(G)$, where $I(y^e) = x_v = (x^d, x^f)$ do
6:         let $z^e$ be an $e_o$-execution successor of $y^e$;
7:         add transition $y^e \xrightarrow{e_o} z^e$ to $f_{yz}^E$;
8:         if $z^e \notin Z^E$ then
9:             $Q_Z^E = Q_Z^E \cup \{z^e\}$;
10:            for $\theta \in E_o^*$, s.t. $\exists \tilde{x}_v = \delta_{vd}(x_v, \theta)$, $\delta_{vs}(\tilde{x}_v, e_o)!$ do
11:                let $y'^e$ be an $(\theta, e_o)$-insertion successor of $z^e$;
12:                add transition $z^e \xrightarrow{\theta} y'^e$ to $f_{zy}^E$;
13:                if $y'^e \notin Q_Y^E$ then
14:                    if $y'^e$ is energy safe then
15:                        $Q_Y^E = Q_Y^E \cup \{y'^e\}$;
16:                        if there exists a run $r_e = y_0^e \xrightarrow{e_0} z_0^e \xrightarrow{\theta_0} y_1^e \cdots \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y'^e$ and $\exists j \leq n$, s.t. $y_j^e \preccurlyeq y'^e$ then
17:                            let $Sub(y'^e) = y_j^e$, stop searching from $y'^e$, $Q_l^E = Q_l^E \cup \{y'^e\}$;
18:                        else $DoDFS(y'^e, Obs(G), G_v)$;
19:                    if $y'^e$ is not energy safe then
20:                        $Q_Y^E = Q_Y^E \cup \{y'^e\}$, $Q_l^E = Q_l^E \cup \{y'^e\}$, stop searching from $y'^e$, ignore all $\theta'$ s.t. $\theta < \theta'$;
21: procedure $Prune(EIS_{pre})$
22:     for $z^e \in Q_Z^E$ that is deadlocking do
23:         remove $z^e$ and all $y^e \in Q_Y^E$, s.t. $f_{yz}^E(y^e, e_o) = z^e$ for some $e_o \in E_o$;
24:         take the accessible part of the structure;
```

Algorithm 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 *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 states of the observer and the desired observer. For the energy level matrix $E_L(y_0^e)$, we track the minimum energy level of the system by unobservable strings. In Line 5, 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 10, 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 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 $I(y_j^e)$ is reached again, i.e., $I(y_j^e) = I(y'^e)$. Let $I(y_j^e) = (x_j^d, x_j^f)$, 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 $\theta_j, \ldots, \theta_{n-1}$ 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_j^e, 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 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 cannot be blocked from happening. More reasoning can be found in Ji, Wu, et al. (2018), where a similar pruning process is conducted. *Prune* works like calculating the *supremal controllable sublanguage* (Cassandras & Lafortune, 2008) 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. Next, we show Algorithm 1 stops after a finite number of steps and returns a finite structure, namely, *EIS*.

Theorem 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., Levy (2002)), there exists an infinite run $y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \xrightarrow{\theta_2} y_3^e \cdots$ in *EIS*. From Algorithm 1, every state in the run is energy safe and it is never the case that $\exists i < j$, s.t. $y_i^e \preccurlyeq y_j^e$. However, this contradicts the well-quasi ordering $\preccurlyeq$ on safe energy information states. □

The size of *EIS* is bounded by Ackermann function (Rackoff, 1978) following a similar augment as in Degorre et al. (2010), which also presented a procedure of "unfolding" the game graph until some simple cycles are formed or the energy level drops below 0. The complexity of *EIS* exceeds its counterpart without energy constraint (Ji, Wu, et al., 2018).

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 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 1. Let the automaton G in Fig. 2 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 shown in Fig. 2. Additionally, 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 Fig. 3 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 Fig. 4 following Definition 4, 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 1 to build *EIS* in Fig. 5, 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 $I(y_1^e) = (C, D)$ as $\delta_{vd}((A, A), a) = (B, A)$ and $\delta_{vs}((B, A), b) = (C, D)$ in G_v. Also $h_{y_1^e}^{(1)}(x_5) = \min\{h_{y_0^e}^{(1)}(x_3) + \omega^{(1)}(b) + \omega_{in}^{(1)}(a), h_{y_0^e}^{(1)}(x_4) + \omega^{(1)}(b) + \omega_{in}^{(1)}(a)\} = 5$, $h_{y_1^e}^{(2)}(x_5) = \min\{h_{y_0^e}^{(2)}(x_3) + \omega^{(2)}(b) + \omega_{in}^{(2)}(a), h_{y_0^e}^{(2)}(x_4) + \omega^{(2)}(b) + \omega_{in}^{(2)}(a)\} = 3$, $h_{y_1^e}^{(1)}(x_6) = \min\{h_{y_1^e}^{(1)}(x_5) + \omega^{(1)}(u_4), h_{y_1^e}^{(1)}(x_5) + \omega^{(1)}(u_5)\} = 0$ and $h_{y_1^e}^{(2)}(x_6) = \min\{h_{y_1^e}^{(2)}(x_5) + \omega^{(2)}(u_4), h_{y_1^e}^{(2)}(x_5) + \omega^{(2)}(u_5)\} = 0$. Hence $y_1^e = \{(C, D), \begin{bmatrix} 5, & 0 \\ 3, & 0 \end{bmatrix}\}$. The other states are calculated similarly.

The first component of $h_{y_1^e}^{(2)}(x_5) = [5, 3]^T$ 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 1. We list the energy and augmented energy information states obtained from *DoDFS* in Table 1.

After *DoDFS*, we find $y_2^e \preccurlyeq y_4^e$, $y_{21}^e \preccurlyeq y_{19}^e$ and $y_{23}^e \preccurlyeq y_{19}^e$, so we stop searching from y_4^e, y_{21}^e and y_{23}^e. Besides, $y_5^e, y_7^e, y_8^e, y_9^e, y_{10}^e, y_{11}^e, y_{12}^e, y_{16}^e, y_{17}^e, y_{18}^e, y_{24}^e$ are not energy safe so they are the bad leaf states. Furthermore, Z-state z_5^e is deadlocking since no transition is defined out of it. Then we prune away z_5^e and its preceding Y-state y_{13}^e in process *Prune* of Algorithm 1. The final *EIS* is shown in Fig. 5, where the dashed lines represent deleted states in the pruning process from EIS_{pre} to *EIS*.

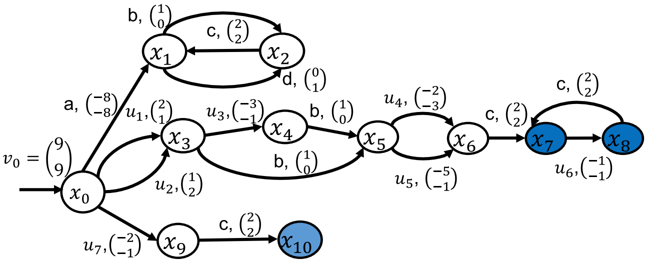

Fig. 2. System G with secret states x_7, x_8, x_{10}.

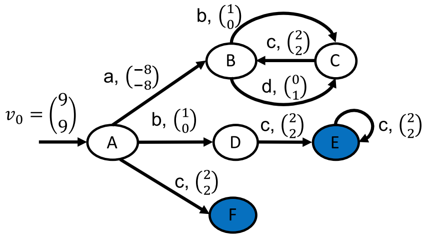

Fig. 3. The observer $Obs(G)$.

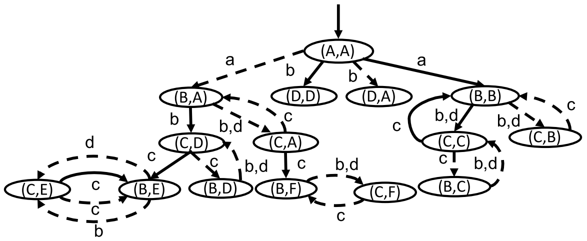

Fig. 4. The verifier G_v where dashed transitions are δ_{vd} transitions and solid transitions are δ_{vs} transitions.

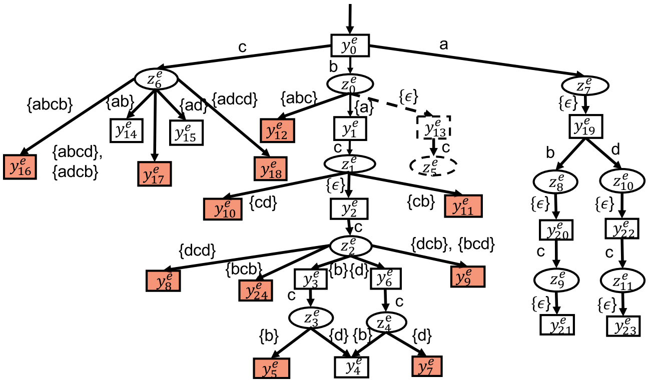

Fig. 5. Energy Insertion Structure (without dashed states).

5. Solution to the 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 solutions to Problem 1.

The runs in *EIS* are finite insertion-execution sequences and 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

Table 1
Energy and augmented energy information states.

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

$Run_y(EIS)$ (respectively $Run_z(EIS)$) the set of runs whose last states are Y-states (respectively Z-states).

Given an initial run $r_e = y_0^e \xrightarrow{e_0} z_0^e \xrightarrow{\theta_0} \cdots y_{n-1}^e \xrightarrow{e_{n-1}} z_{n-1}^e \xrightarrow{\theta_{n-1}} y_n^e$, the *edit projection* $P_e : Run(EIS) \rightarrow P[\mathcal{L}(G)]$ 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 $P_e(r_e) = l$. 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 $l_g(r_e)$, i.e., $l_g(r_e)$ is the string after insertion. By Lemma 1, $\delta_d(x_{obs,0}, l_g(r_e))$ 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 $\pi_{in} : Run_z(EIS) \rightarrow E_o^*$ and the *environment's strategy* as $\pi_{en} : Run_y(EIS) \rightarrow E_o$. 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 *positional* if the decisions only depend on the current energy (augmented energy) information state. In other words, $\pi_i \in \Pi_i$ is positional 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, positional strategies for the insertion function and the environment can be represented as $\pi_{in} : Q_Z^E \rightarrow E_o^*$ and $\pi_{en} : Q_Y^E \rightarrow E_o$, respectively. From results in Apt and Grädel (2011) and Ehrenfeucht and Mycielski (1979), positional strategies are sufficient for players to win a reachability game, thus we simply 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.

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* (Apt & Grädel, 2011). 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. (Apt & Grädel, 2011). Then we present Algorithm 2 to compute Win_{in}.

Algorithm 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 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 2.

Lemma 2. *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. 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 $Last_Y(r_e) \in Q_{lb}^E$ holds 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 2. 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 2. 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 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 extended 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 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 3.

Lemma 3. *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 2, which contradicts the assumption. □

Now we are now ready to state one of the main results in this paper. Given a winning insertion strategy in *EIS*, we can always construct an insertion function solving Problem 1. Conversely, if there exists an insertion function solving Problem 1, we can always find a winning insertion strategy in *EIS*.

Theorem 3. *There exists an insertion function solving Problem 1 if and only if there exists a winning strategy for the insertion function in EIS.*

Proof. The "only if" part: by contrapositive, i.e., if no winning insertion strategy exists in *EIS*, then no insertion function solves Problem 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 2 returns an empty set. So by Lemma 2, $\exists s \in \mathcal{L}(G)$ 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 2, every initial run $r_e(l)$ is removed, thus the initial state of *EIS* is also removed and EIS_w becomes empty. From the construction process in Algorithm 1, for all feasible insertion choices $\theta_0, \ldots, \theta_{n-1}$ s.t. s is mapped to s' by Convention 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 1.

The "if" part. Suppose that π_{in} is a winning insertion strategy in *EIS*. Since we follow Algorithm 2 to obtain Win_{in} and EIS_w, then

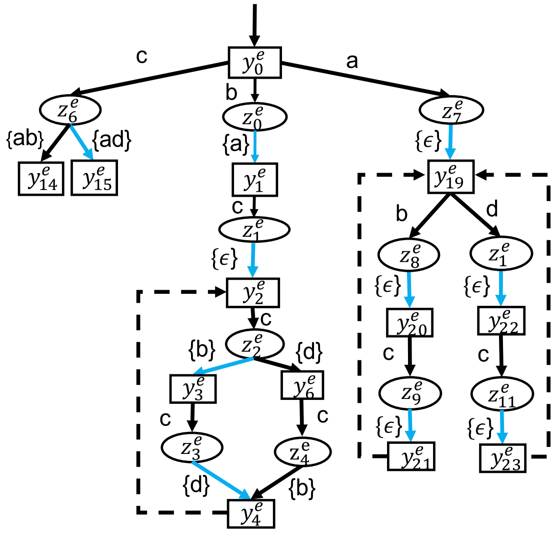

Fig. 6. EIS_w with a winning insertion strategy indicated by blue lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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 1, we know that $\forall s \in P^{-1}(l) \cap \mathcal{L}(G)$, $V_m(s, s_{\pi_{in}}) \geq \vec{0}$, therefore π_{in} solves Problem 1. □

The above theorem shows the completeness and soundness of Algorithms 1 and 2. Therefore, Problem 1 can be solved by first building EIS and then finding the insertion function's winning strategies if they exist. As was shown in the last section, the state space of EIS is bounded by Ackermann function, which is not primitive recursive. Also, both the winning set and strategies for a reachability game can be computed in linear time with respect to the size of EIS from results in Apt and Grädel (2011). Therefore we have the complexity bound for solving Problem 1. We end this section by revisiting our running example.

Example 2.

We revisit Example 1 and synthesize insertion functions to solve Problem 1. We follow Algorithm 2 and build EIS_w in Fig. 6. In Algorithm 2, 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 Fig. 6 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 1 and is indicated by blue lines in Fig. 6. Finally, we encode this selected insertion function as an I/O automaton in Fig. 7, where the insertion decisions are explicitly shown.

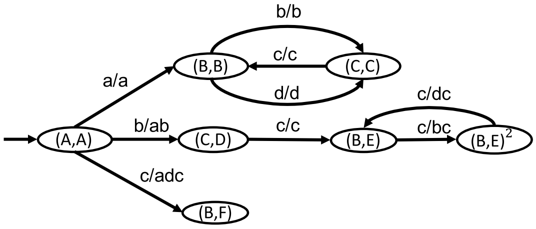

Fig. 7. An insertion function that solves Problem 1.

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.

6.1. Motivation and problem formulation

The structure EIS_w obtained in the last section usually contains more than one insertion strategies that solve Problem 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 $V_c : Run(EIS_m) \rightarrow (\mathbb{Z} \setminus \mathbb{N})^k$ 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 $V_{mc} : Run_{inf}(EIS_m) \rightarrow \mathbb{R}^k$ as the limit mean insertion weight function for infinite runs in EIS_m. Given $r_m = y_1^e \xrightarrow{e_1} z_1^e \xrightarrow{\theta_1} y_2^e \xrightarrow{e_2} z_2^e \xrightarrow{\theta_2} \cdots$, $V_{mc}(r_m) = \liminf_{n\rightarrow\infty} \frac{1}{n} \sum_{i=1}^{n} \omega_{in}(\theta_i)$. Then we propose the *bounded cost rate insertion strategy synthesis problem.*

Problem 2. Synthesize a bounded cost rate insertion strategy π_{in} such that for any infinite initial run $r_m \in Run_{inf}(\pi_{in}, y_0^e)$, $-V_{mc}(r_m) \leq v_b$ 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 2 can be viewed as a *multidimensional mean payoff game* (Chatterjee & Velner, 2017) 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 2 and obtain $\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \omega_{in}(\theta_i) \geq -v_b$. Equivalently, we may show whether $\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}(\omega_{in}(\theta_i) + v_b) \geq \overrightarrow{0}$ holds. Hence, we add v_b to each insertion weight vector in EIS_m and discuss the equivalent mean payoff objective. For simplicity, we still denote the game graph by EIS_m. We further let $W = \max\{-\omega_{in}^{(i)}(\theta) : \exists z^e \in Q_Z^E, \theta \in E_o^*, \text{ s.t. } f_{zy}^E(z^e, \theta)!, 1 \leq i \leq k\}$ be the maximal absolute value of elements in insertion weight functions in EIS_m. Obviously, W is a positive integer.

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 Chatterjee and Velner (2017) to solve Problem 2. Originally, this technique was developed for general multidimensional mean payoff games. The main idea is to reduce the multidimensional mean payoff game in Problem 2 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 the 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 2. Intuitively, we are going to "separate" the convex combinations leading to each player to win the game. By linear space theory, 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., Boyd and Vandenberghe (2004). 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.

6.3. Synthesize bounded cost rate insertion strategies

We establish the relation between the original multidimensional mean payoff game and the induced one-dimensional mean payoff game after applying the hyperplane separation technique. Then we derive solutions to Problem 2.

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* (Ehrenfeucht & Mycielski, 1979), 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 of Chatterjee and Velner (2017) here: (i) *For every $\lambda \in \mathbb{R}^k$, we have $Win_{em}^{\lambda} \subseteq Win_{em}$; also if $Win_{em}^{\lambda} \neq \emptyset$, then $Win_{em} \neq \emptyset$*; (ii) *If for all $\lambda \in \mathbb{R}^k$ we have $Win_{em}^{\lambda} = \emptyset$, then $Win_{em} = \emptyset$*. These results establish the relation between the winning regions for both players in the original game and the new game.

These results illustrate a potential way to determine whether the environment player has a non-empty 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 Chatterjee and Velner (2017) 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 Chatterjee and Velner (2017), 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). Lemma 3 of Chatterjee and Velner (2017) is stated here: *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 $Win_{em}^{\lambda'} \neq \emptyset$*. The proof is omitted.

To summarize and strengthen the above results, we repeat Lemma 4 in Chatterjee and Velner (2017) as a theorem here.

Theorem 4. *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 Chatterjee and Velner (2017).

Based on the above results, we present Algorithm 3 to solve Problem 2. We first assume that states in EIS_m are numbered from 1 to n. At each state, 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 Brim, Chaloupka, Doyen, Gentilini, and Raskin (2011) 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 $Attr_{em}(Q) = \bigcup_{j \geq 0} Q_j$. The environment ensures to reach Q_j from Q_{j+1} within one transition regardless of the insertion function's strategies, so it 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 $Attr_{em}(Q)$. Similarly, we define the attractor for the insertion function.

Algorithm 3: Find solutions to Problem 2

```
Input: $EIS_m$
Output: Insertion strategies solving Problem 2
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 Brim et al. (2011) 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 2.
```

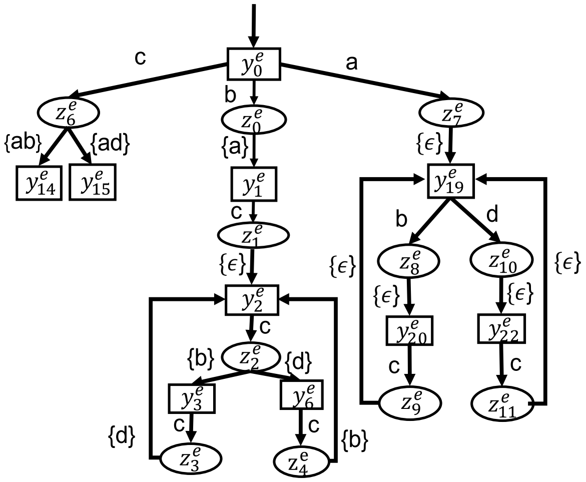

Fig. 8. EIS_m after merging states.

In Algorithm 3, we apply the method in Brim et al. (2011) 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 2 against all environment's strategies if the remaining structure is not empty. Otherwise, no solution exists for Problem 2 if all states of EIS_m are removed. Besides, as positional strategies suffice to win a mean payoff game with perfect information (Ehrenfeucht & Mycielski, 1979), we simply let strategies returned by Algorithm 3 be positional so that a finite number of strategies are returned. The correctness of Algorithm 3 is from Theorem 4 and more details concerning solving a one-dimensional mean payoff game are available in Brim et al. (2011).

Finally, we briefly discuss the complexity of Algorithm 3 following a similar argument as in Chatterjee and Velner (2017). 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 Brim et al. (2011), 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 3.

We revisit Example 2 and further discuss Problem 2 based on the solutions of Problem 1. We show EIS_m in Fig. 8 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 threshold $v_b = [3, 3]^T$ and see if Problem 2 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 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 3, we find that there exist insertion strategies solving Problem 2. 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.

[1] The pruning here is similar to calculating the supremal controllable sublanguage (Cassandras & Lafortune, 2008) 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.

However, if we change the threshold vector to $v_b' = [1, 1]^T$, then Problem 2 has no solution. From Fig. 8, 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 can enforce mean payoff threshold $[1, 1]^T$.

7. Conclusion

This work investigated opacity enforcement by insertion functions under multiple quantitative constraints for the first time in discrete event systems. 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. We transferred the constrained opacity enforcement problem to a two-player game between the insertion function and the environment. A bipartite information structure called Energy Insertion Structure was defined to characterize the game. It also provides a sound and complete characterization of the solution space. Then we subsequently considered the rate of insertion cost and formulated the bounded cost rate insertion strategy synthesis problem, which was characterized 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 game, we obtain valid solutions for the original problem.

References

Apt, K. R., & Grädel, E. (2011). *Lectures in game theory for computer scientists*. Cambridge University Press.

Barcelos, R. J., & Basilio, J. C. (2018). Enforcing current-state opacity through shuffle in event observations. In *Proceedings of the 14th international workshop on discrete event systems* (pp. 100–105).

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

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., & Raskin, J.-F. (2011). Faster algorithms for mean-payoff games. *Formal Methods in System Design*, *38*(2), 97–118.

Cassandras, C. G., & Lafortune, S. (2008). Introduction to discrete event systems – 2nd edition. Springer.

Cassez, F. (2009). The dark side of timed opacity. In *International conference on information security and assurance* (pp. 21–30). Springer.

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

Chatterjee, K., & Velner, Y. (2017). Hyperplane separation technique for multidimensional mean-payoff games. *Journal of Computer and System Sciences*, *88*, 236–259.

Chédor, S., Morvan, C., Pinchinat, S., & Marchand, H. (2015). 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.

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

Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., & Toruńczyk, S. (2010). Energy and mean-payoff games with imperfect information. In *Computer science logic* (pp. 260–274). Springer.

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

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

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

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

Ji, Y., Wu, Y.-C., & Lafortune, S. (2018). Enforcement of opacity by public and private insertion functions. *Automatica*, *93*, 369–378.

Ji, Y., Yin, X., & Lafortune, S. (2018). Opacity enforcement by insertion functions under energy constraints. In *Proceedings of the 14th international workshop on discrete event systems* (pp. 302–308).

Ji, Y., Yin, X., & Lafortune, S. (2019). Opacity enforcement using nondeterministic publicly-known edit functions. *IEEE Transactions on Automatic Control*, (in press). http://dx.doi.org/10.1109/TAC.2019.2897553.

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

Levy, A. (2002). *Basic set theory (Vol. 13)*. Courier Corporation.

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

Masopust, T., & Yin, X. (2019). Complexity of detectability, opacity and A-diagnosability for modular discrete event systems. *Automatica*, *101*, 290–295.

Mohajerani, S., Ji, Y., & Lafortune, S. (2018). Efficient synthesis of edit functions for opacity enforcement using bisimulation-based abstractions. In *Proc. of 57th IEEE conf. on decision and control* (pp. 4849–4854).

Mullins, J., & Yeddes, M. (2014). Opacity with Orwellian observers and intransitive non-interference. In *Proceedings of the 12th international workshop on discrete event systems* (pp. 344–349).

Nash-Williams, C. S. J. A. (1963). On well-quasi-ordering finite trees. *Mathematical Proceedings of the Cambridge Philosophy Society*, *59*(4), 833–835.

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

Pruekprasert, S., & Ushio, T. (2017). 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.

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

Saboori, A., & Hadjicostis, C. (2007). Notions of security and opacity in discrete event systems. In *Proceedings of the 46th IEEE conference on decision and control* (pp. 5056–5061). IEEE.

Saboori, A., & Hadjicostis, C. (2013). Verification of initial-state opacity in security applications of discrete event systems. *Information Sciences*, *246*, 115–132.

Takai, S., & Oka, Y. (2008). 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.

Tong, Y., Li, Z., Seatzu, C., & Giua, A. (2017). Decidability of opacity verification problems in labeled Petri net systems. *Automatica*, *80*, 48–53.

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

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

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

Yin, X., & Lafortune, S. (2019). A general approach for optimizing dynamic sensor activations for discrete event systems. *Automatica*, *105*, 376–383.

Yin, X., Li, Z., Wang, W., & Li, S. (2019). Infinite-step opacity and K-step opacity of stochastic discrete-event systems. *Automatica*, *99*, 266–274.

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

Zhang, K., Yin, X., & Zamani, M. (2019). Opacity of nondeterministic transition systems: A (bi) simulation relation approach. *IEEE Transactions on Automatic Control*, (in press). http://dx.doi.org/10.1109/TAC.2019.2908726.

Yiding Ji received his Bachelor of Engineering degree of Electrical Engineering from Tianjin University, Tianjin, China, in 2014, Master of Science degree and Doctor's degree of Electrical and Computer Engineering both from the University of Michigan, USA, in 2016 and 2019, respectively. His research interests include theory of discrete event systems, security and privacy in cyber and cyber–physical systems, formal methods and game theory.

He is now a member of the IEEE Control Systems Society Technical Committee on Discrete Event Systems.

Xiang Yin was born in Anhui, China, in 1991. He received the B.Eng. degree from Zhejiang University in 2012, the M.S. degree from the University of Michigan, Ann Arbor, in 2013, and the Ph.D. degree from the University of Michigan, Ann Arbor, in 2017, all in electrical engineering.

Since 2017, he has been with the Department of Automation, Shanghai Jiao Tong University, where he is an Associate Professor.

His research interests include formal methods, control of discrete-event systems, model-based fault diagnosis, security and their applications to cyber and cyber–physical systems.

Dr. Yin received the Outstanding Reviewer Awards from Automatica, the IEEE Transactions on Automatic Control and the Journal of Discrete Event Dynamic Systems. Dr. Yin also received the IEEE Conference on Decision and Control (CDC) Best Student Paper Award Finalist in 2016.

He is the co-chair of the IEEE CSS Technical Committee on Discrete Event Systems.

Stéphane Lafortune received the B.Eng. degree from Ecole Polytechnique de Montréal in 1980, the M.Eng. degree from McGill University in 1982, and the Ph.D. degree from the University of California at Berkeley in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science.

Lafortune is a Fellow of the IEEE (1999) and of IFAC (2017). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper co-authored with S.-L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett).

Lafortune's research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He co-authored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems — Second Edition* (Springer, 2008). Lafortune is Editor-in-Chief of the Journal of Discrete Event Dynamic Systems: Theory and Applications.