Limited circulation. For review only
IEEE-TAC Submission no.: FP-13-810.4

Synthesis of Optimal Insertion Functions for Opacity Enforcement

Yi-Chin Wu and Stéphane Lafortune

***Abstract*—Our prior work has studied the enforcement of opacity security properties using insertion functions. Given a system that is not opaque, the so-called All Insertion Structure (AIS) is a game structure, played by the system and the insertion function, that embeds all valid insertion functions. In this paper, we first propose a more compact AIS that can be constructed with lower computational complexity. We then introduce the *maximum total cost* and the *maximum mean cost*, and use them as quantitative objectives to solve for optimal insertion functions. Specifically, we first determine if an insertion function with a finite total cost exists. If such an insertion function exists, we synthesize an optimal *total-cost* insertion function. Otherwise, we construct an optimal *mean-cost* insertion function. In either case, we find an optimal insertion strategy on the AIS, with respect to the corresponding cost objective. The algorithmic procedures are adapted from results developed for *minimax games* and *mean payoff games*. The resulting optimal strategy is represented as a subgraph of the AIS that consists of all the system actions and the optimal insertion actions. Finally, we use this subgraph to synthesize an optimal insertion function that is encoded as an I/O automaton.**

I. Introduction

With the rapid development of networked devices and network technologies, network services have become increasingly popular. While such services bring much convenience, they often entice users to share their personal information and thus pose a major privacy threat to users. In this paper, we study an important security property called "opacity", which characterizes whether a given secret can be inferred by intruders. The notion of opacity was first introduced in the computer science community [14]. It then quickly became an active research topic in Discrete Event Systems (DES) as DES theory provides suitable models and analytical techniques for formulating and studying opacity properties [3], [4], [15].

We consider opacity properties in DES modeled as finite-state automata. The settings of an opacity problem are: (i) the system is partially observable and/or nondeterministic; (ii) the system has a *secret* that can be modeled as states, strings, or a combination of them; (iii) the *intruder* is an observer that has full knowledge of the system structure. Opacity holds if for every secret behavior, there is a non-secret behavior that is observationally-equivalent. Hence, by observing the observable behavior of the system, the intruder is never sure whether the secret has occurred or not. Depending on how the secret is modeled, various notions of opacity are defined. In this paper, we consider four opacity notions: language-based opacity, initial-state opacity, current-state opacity, and initial-and-final-state opacity. They will be collectively referred to as "opacity" as the four notions can be mapped to one another [21].

Methods for verifying if a given opacity property holds have been investigated in [6], [13], [16], [21]. If opacity is violated, researchers have provided various mechanisms for enforcing opacity [2], [6], [8], [10], [17], [22]. In our prior work [22], we have proposed to use insertion functions to enforce opacity. As shown in Figure 1, an insertion function is a *run-time* monitoring interface placed at the output of the system. The intruder is assumed to have no knowledge

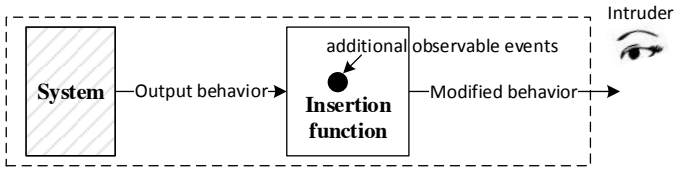


Fig. 1. The insertion mechanism.

of the insertion function at the outset and thus is expecting to observe behaviors that are consistent with the system structure. We have formally characterized the specifications for insertion functions as *i-enforceability* in [22]. A given insertion function is *i-enforcing* if (i) it allows all the system output behaviors; and (ii) every modified behavior from the insertion function is observationally equivalent to an *existing* non-secret behavior.

The "All Insertion Structure" (AIS) was introduced in [22] to embed *all* i-enforcing insertion functions. It is a bipartite directed graph with transition labels that enumerates all the system's output events at "system" states and all insertion choices at "insertion" states. Such a transition structure describes a *game* where the insertion function tries to react to the output of the system. This paper builds on the results in [22] to solve *optimal insertion problems*. Moreover, we present a more compact AIS that embeds all i-enforcing insertion functions using fewer states. Specifically, the state space of the AIS presented in [22] is exponential in the number of states of the state estimator used to verify opacity. In this paper, we reduce the state space of the AIS to polynomial (in the number of states of the state estimator), thereby achieving significant computational gains. The more compact AIS still embeds all i-enforcing insertion functions in its structure. We then consider the synthesis of optimal insertion functions by introducing quantitative objectives to quantify insertion functions, a problem not treated in [22].

In this regard, the *maximum total cost* and the *maximum mean cost* are introduced. The first cost captures the total insertion cost and the second cost considers the average insertion cost (per system output), both in the worst-case scenario. Specifically, we solve two optimization problems. We develop a test that determines if there is an insertion function that has a finite total cost. If such an insertion function exists, we then minimize the maximum total cost and synthesize an optimal *total-cost* insertion function. Otherwise, we minimize the maximum mean cost and synthesize an optimal *mean-cost* insertion function.

The synthesis of an optimal insertion function is solved by first finding an optimal strategy for the insertion function on the AIS, and then using the optimal strategy to construct an *insertion automaton*. A strategy of the insertion function is a mapping from every historical interaction of the system and the insertion function to an insertion action. It uniquely represents a given insertion function. An insertion automaton, on the other hand, is a compact encoding of an insertion function that can be easily composed with the system automaton. To

Y.-C. Wu is with the Department of EECS at the University of Michigan and the Department of EECS at the University of California, Berkeley (yichin.wu@berkeley.edu). S. Lafortune is with the Department of EECS at the University of Michigan (stephane@umich.edu). This work was partially supported by the NSF Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering (grant CCF-1138860) and by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

find an optimal strategy, we leverage results from *minimax games* for the maximum total cost objective, and from *mean payoff games*, developed in [24], for the maximum mean cost objective. After the optimal strategy is found, we construct an insertion automaton, which is an I/O automaton that encodes the optimal insertion function. Our approach is inspired by [7] and [6], where an optimal dynamic observer is synthesized for fault diagnosis and opacity enforcement, respectively. But here we use insertion functions instead of dynamic observers.

Our main contributions in this paper are twofold. First, we propose a more compact AIS whose state space is significantly reduced. Second, we develop a methodology and algorithmic procedures for the synthesis of an optimal insertion function with respect to the maximum total cost or the maximum mean cost. The methodology exploits the structure of the AIS and adapts and extends theoretical results for games on weighted automata to the problem of opacity enforcement by insertion functions. For most definitions and algorithms in the paper, technical details and illustrative examples are provided to guide the reader through our discussion.

The remaining sections of this paper are organized as follows. Section II introduces the system model and defines the opacity problem. Section III reviews the insertion mechanism and provides a motivating example related to Location-Based Services (LBS). In Section IV, we present our new algorithm for the construction of the All Insertion Structure (AIS). Subsequently, we define in Section V the maximum total cost and the maximum mean cost for insertion functions represented as insertion strategies on the AIS. In Section VI, we consider the maximum total cost and present an algorithm for synthesizing an optimal total-cost insertion function. In Section VII, we consider the maximum mean cost and present an algorithm for synthesizing an optimal mean-cost insertion function. Finally, Section VIII concludes the paper.

II. Opacity Notions in Automata Models

A. Automata Models

We consider opacity problems in DES systems modeled as (potentially nondeterministic) automata. An automaton $G=(X,E,f,X_0)$ has a finite set of states, a set of events E, a partial state transition function $f: X\times E\to 2^X$, and a set of initial states X_0. The transition function is extended to domain $X\times E^*$ in the standard manner [5]. In opacity problems, the initial state need not be known *a priori* by the intruder and thus we include a set of initial states X_0 in the definition of G. The language generated by G is the system behavior that is defined by $\mathscr{L}(G,X_0):=\{t\in E^*:(\exists x\in X_0)[f(x,t) \text{ is defined}]\}$. For simplicity, we write $\mathscr{L}(G)$ if X_0 is clearly defined and write $\mathscr{L}(G,x)$ if $X_0=\{x\}$. The system is partially observable in general. Hence, the event set is partitioned into an observable set E_o and an unobservable set E_{uo}. Given a string $t\in E^*$, its observation is the output of the natural projection $P: E^*\to E_o^*$, which is recursively defined as $P(t)=P(t'e)=P(t')P(e)$ where $t'\in E^*$ and $e\in E$. Projection of an event is $P(e)=e$ if $e\in E_o$ and $P(e)=\varepsilon$ if $e\in E_{uo}\cup\{\varepsilon\}$ where ε is the empty string.

B. Current-State Opacity

We consider opacity properties in DES modeled as finite-state automata. The settings of an opacity problem are: (1) G has a *secret*; (2) G is partially observable and/or nondeterministic; (3) the intruder is an observer of G that has full knowledge of the structure of G. Hence, the intruder, with the knowledge of G and its observation, can infer the real system behavior by constructing estimates. Opacity holds if no intruder's estimate reveals the occurrence of the secret. In other words, the system is opaque if *for any secret behavior, there exists another non-secret behavior that is observationally equivalent to the intruder*. Therefore, the intruder is never sure whether the secret has occurred or not.

We consider four notions of opacity studied in the literature: current-state opacity (CSO), initial-state opacity (ISO), language-based opacity (LBO), and initial-and-final-state opacity (IFO). Based on [21], these four notions can be mapped to one and another. Thus, deriving our results in this paper using one notion of opacity is sufficient to show that these results apply to the other three notions of opacity. For simplicity, we choose to present only the formal definition of current-state opacity and use it to demonstrate our results in the remainder of this paper.

Definition 1 (Current-State Opacity (CSO))**.** *Given system $G=(X,E,f,X_0)$, projection P, and the set of secret states $X_S\subseteq X$, the system is current-state opaque if $\forall i\in X_0$ and $\forall t\in\mathscr{L}(G,i)$ such that $f(i,t)\subseteq X_S$, $\exists j\in X_0$, $\exists t'\in\mathscr{L}(G,j)$ such that: (i) $f(j,t')\cap(X\setminus X_S)\neq\emptyset$ and (ii) $P(t)=P(t')$.*

To verify opacity, one can build the corresponding *forward state estimator* and check if any estimate contains only the secret information (specifically, current states, initial states, or initial-and-final-state pairs). A forward state estimator is an automaton where the state reached by string $s\in P[\mathscr{L}(G)]$ is the intruder's [current-state; initial-state; initial-and-final-state] *estimate* when the intruder observes string s. Specifically, CSO and LBO can be verified by the standard *observer automaton* defined in Section 2.5.2 of [5]; ISO and IFO can be verified by the trellis-based initial-state estimator introduced in [16]. For simplicity, we will call a forward state estimator an *estimator* and denote it by $\mathscr{E}$ hereafter.

III. Insertion Mechanism for Opacity Enforcement

In [22], we proposed to enforce opacity using insertion functions when opacity is violated. As shown in Figure 1, the insertion function is a special monitoring interface that inserts additional events to the system output when necessary. Given an observed *event* from the system, the insertion function possibly inserts extra observable events before the system event and outputs the resulting *string*. For the intruder, the inserted events are indistinguishable from the genuine observable events of the system. The intruder, observing at the output of the system, cannot tell if the observed event is inserted or genuine.

A. Insertion Functions and Insertion Automata

While inserted and genuine observable events are indistinguishable for the intruder, we need to distinguish them for the sake of discussion. In the formulation of insertion functions, we attach to each inserted event a *virtual* insertion label. That is, we denote inserted events by $e_i\in E_i:=\{e_i:e\in E_o\}$ instead of $e\in E_o$. Formally, an insertion function is defined as a (potentially partial) function $f_I: E_o^*\times E_o\to E_i^*E_o$ that outputs a string with inserted events based on the past observed behavior and the current observed event. Given string $t\in\mathscr{L}(G)$ where $P(t)=se_o$, an insertion function is defined such that $f_I(s,e_o)=s_Ie_o$ when string $s_I\in E_i^*$ is inserted before e_o. We also define a *string-based* insertion function f_I^{str} from f_I: $f_I^{str}(\varepsilon)=\varepsilon$ and $f_I^{str}(s_n)=f_I(\varepsilon,e_1)f_I(e_1,e_2)\cdots f_I(e_1e_2\ldots e_{n-1},e_n)$ where $s_n=e_1e_2\ldots e_n\in E_o^*$. Given G, the modified language output by the insertion function is $f_I^{str}(P[\mathscr{L}(G)])=\{\tilde{s}\in(E_i^*E_o)^*:\tilde{s}=f_I^{str}(s)\wedge s\in P[\mathscr{L}(G)]\}$.

We encode a given insertion function as an I/O (possibly infinite state) automaton $IA=(X_{ia},E_o,E_i^*E_o,f_{ia},q_{ia},x_{0,ia})$ and call it an insertion automaton. Specifically, given IA, the state set is X_{ia}, the input set is E_o, the output set is a set of *strings* in $E_i^*E_o$, the transition function f_{ia} defines the dynamics of IA, the output function

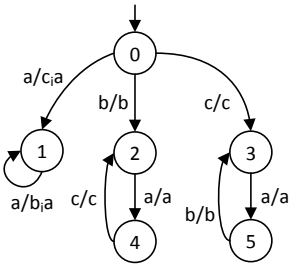


Fig. 2. Insertion automaton IA.

q_{ia} is defined such that $q_{ia}(x,e_o) = s_I e_o$ where $f_{ia}(x_{0,ia},s) = x$, if $f_I(s,e_o) = s_I e_o$, and finally $x_{0,ia}$ is the initial state. In Figure 2, we show an insertion automaton that does the following: If the first output is b or c, then the insertion automaton does not insert anything; however, if the first output is a, then the insertion automaton inserts c_i before a, and subsequently inserts b_i for all subsequent a. More formally, this insertion function is defined as follows: $f_I(\varepsilon,a) = c_i a, f_I(s,e_o) = b_i a$ for $se_o = a^n a, n \geq 1$, and $f_I(s,e_o) = e_o$ for $se_o \in P[\mathscr{L}(G)] \setminus \{a^n a, n \geq 1\}$.

In general, an insertion automaton can have an infinite number of states. If an insertion function can be encoded as a finite-state insertion automaton, we say it is "finite". Based on the results in [22], an i-enforcing insertion function exists if and only if a finite one exists. Hence, with no loss of generality, we consider only finite insertion functions hereafter.

B. I-Enforceability Property

In [22], we have formally characterized *i-enforceability*, a property that captures the input and output specifications for insertion functions. Specifically, two requirements must be satisfied. First, the modified output of an insertion function should be always consistent with the original system structure. We assume that the intruder does not know the implementation of the insertion function at the outset and thus is expecting to observe behaviors that are consistent with the system structure. Hence, we require that the modified output $f_I^{str}(P[\mathscr{L}(G)])$ should always be within $L_{safe} := P[\mathscr{L}(G)] \setminus (P[\mathscr{L}(G)] \setminus P(L_{NS}))E_o^*$, where $L_{NS} = \{t \in \mathscr{L}(G,X_0) : \exists i \in X_0, f(i,t) \cap (X \setminus X_S) \neq \emptyset\}$. The concatenation of E_o^* implements the idea that "once the secret is revealed, it cannot be recovered." We will call the insertions satisfying this specification *safe*. Second, the insertion function should not exclude any behavior from the system. That is, $f_I(s,e_o)$ should be defined for all $se_o \in P[\mathscr{L}(G)]$. Such insertions are called *admissible*. A given insertion function is called *i-enforcing* if it satisfies both the safety and the admissibility requirements. A given opacity property is i-enforceable if there exists an i-enforcing insertion function.

C. Motivating Example

Systems where the users query servers while requiring the secret information to be hidden from the servers are suitable for the insertion mechanism, as i-enforcing insertion functions can conceal the secret information without interfering with the querying process. Here, we discuss a simpler version of the example in [23] that applies the insertion mechanism to the problem of preserving location privacy in Location-Based Services (LBS). We refer the interested reader to [23] for a complete discussion. In LBS, the users query the server using their locations; but at the same time, they may also want to hide their exact locations from that server [18]. One popular technique to protect users' location privacy is the *anonymizer* technique, introduced in [11] and adopted in many subsequent works. The anonymizer generalizes the user's exact location in a given query to a region containing that location. While this technique works for individual queries, the server may still be able to infer the user's exact location based on its knowledge of the user's mobility patterns. The challenge is how can a user hide its exact location from the LBS server while continuously using the service.

We explain how to address location privacy as a current-state opacity problem in DES. First, the automaton model for the LBS application is obtained from the user's mobility patterns, where states are point locations on the physical map and where a transition exists if the user can move from one location to another (based on existing walking paths or roads for instance). Figure 3 shows such an automaton built from a given user's mobility patterns. The set of initial states is the entire state space as the user can start from any location. Transitions are labeled with the *region* information that the server receives in the queries under the anonymizer mechanism. For instance, the server receives region $a = \{0,1\}$ when the user moves from location 0 or location 1. The other regions are defined as $b = \{2,3\}, c = \{4,5\}, d = \{6,7\}$. Let us consider the CSO specification that the user wants to hide its visits to location 6. We let 6 be the only secret state in G.

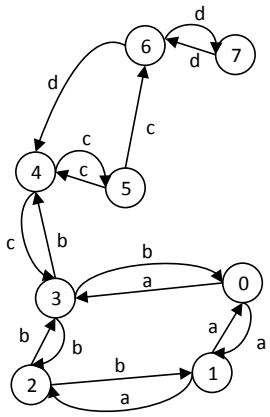


Fig. 3. The automaton G built from the mobility patterns.

To verify the above CSO requirement, we build in Figure 4 the current-state estimator of G. Because estimate state $\{6\}$ contains only the secret state, G is not opaque. That is, the server will know for sure that the user is currently visiting location 6 if it receives a query sequence such as cdd. To avoid revelation of location 6, we propose to insert fictitious queries to the anonymizer's original query sequences using an i-enforcing insertion function. The insertion mechanism is suitable for LBS as insertion functions can insert fictitious queries and drop replies to fictitious queries without interfering with the user's querying process. Also, i-enforcing insertion functions guarantee that fictitious queries are inserted in a "convincing" manner where every modified behavior is consistent with an original mobility pattern. However, fictitious queries are "costly" as they introduce overhead in terms of delay and bandwidth, and they also consume energy. Hence, it is desirable to obtain an *optimal* insertion function with respect to a given insertion cost model that captures delay, bandwidth, and/or energy. In Sections V, VI, and VII, we develop algorithms that synthesize optimal insertion functions for a general cost model. The optimal insertion function for the above LBS example will be presented later in this paper.

IV. The All Insertion Structure (AIS)

To synthesize an optimal i-enforcing insertion function, we construct the All Insertion Structure (AIS) and perform optimization on it. The AIS, first developed in [22], is an automaton that embeds in a

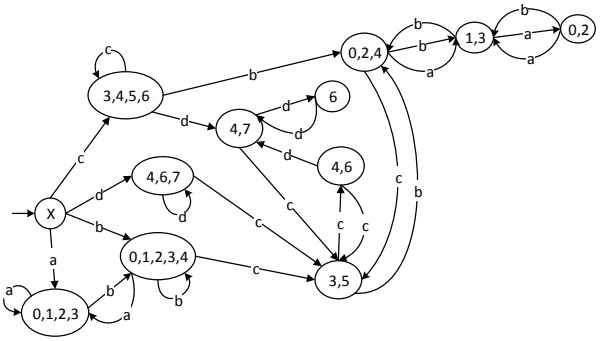


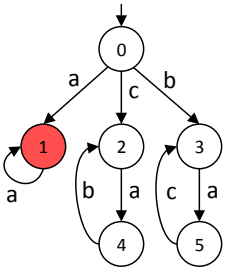
Fig. 4. The current-state estimator.

finite game structure *all deterministic i-enforcing insertion functions for a given secret of the system*. In this section, we propose a more compact AIS that has a smaller state space. Specifically, the algorithm for constructing this new AIS comprises three stages: (1) constructing the i-verifier V; (2) constructing the unfolded i-verifier V_u; and (3) pruning and obtaining the AIS. The construction of the meta-observer, which was an intermediate stage in the construction algorithm in [22], is removed. With the removal of the meta-observer construction and other improvements, the state space of the AIS is reduced from exponential to polynomial, in the state space of the estimator that verifies opacity. The resulting new AIS is theoretically more compact than that in [22], with multiple states potentially combined into one state. To distinguish the two AIS, hereafter, we relabel the AIS obtained by the algorithm in [22] as AIS^e. Note that the AIS constructed in this paper, simply denoted by AIS, has the same properties as the AIS^e relative to synthesis. That is, the AIS embeds all i-enforcing insertion functions, and the AIS is not the empty automaton if and only if opacity is i-enforceable. This will become clear after we present the construction procedure.

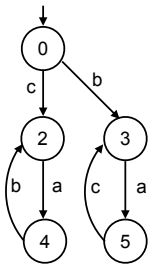
A. Construction of the AIS

Stage 1: Constructing the i-verifier V

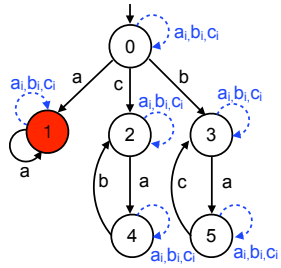
Recall the safety specification for insertion functions that is defined in Section III. The purpose of constructing the i-verifier is to identify all safe insertions, i.e., insertions for which the modified output is within L_{safe}. The construction of the i-verifier in the new algorithm is the same as that in the algorithm in [22]. It depends on the system's estimator that is used to verify opacity. Specifically, consider, for example, the current-state estimator $\mathscr{E}$ in Figure 5(a) for a system with $E_o = \{a,b,c\}$. For simplicity, the states of $\mathscr{E}$ are numbered from 0 to 5 (normally, these states would be sets of system states.) Assume that state 1 reveals the secret, but the other states do not. We build the *desired estimator* $\mathscr{E}^d$, as shown in Figure 5(b), by deleting in $\mathscr{E}$ all estimates that reveal the secret and taking the accessible part. Then, we build in Figure 5(c) the *feasible estimator* $\mathscr{E}^f$ by adding a self-loop for every inserted event $e_i \in E_i$ at every state in $\mathscr{E}$. In the figure, inserted events are represented using dashed transitions. These self-loops generate all possible inserted strings while not triggering real state transitions in the system. Finally, to obtain the i-verifier, we compose $\mathscr{E}^d$ and $\mathscr{E}^f$ using an operation called *dashed parallel composition* and construct $V := \mathscr{E}^d||_d\mathscr{E}^f = (M_v, E_o \cup E_i, \delta_v, m_{v,0})$ in Figure 5(d). Dashed parallel composition is a special parallel composition that synchronizes solid transitions $e \in E_o$ in $\mathscr{E}^d$ and all transitions ($e \in E_o$ and $e_i \in E_i$) in $\mathscr{E}^f$ if the two transitions are labeled by the same event, subject to the virtual inserted label. The resulting transition is dashed if and only if the corresponding transition in $\mathscr{E}^f$ is dashed. For example, transition $0 \xrightarrow{b} 3$ in Figure 5(b) synchronizes



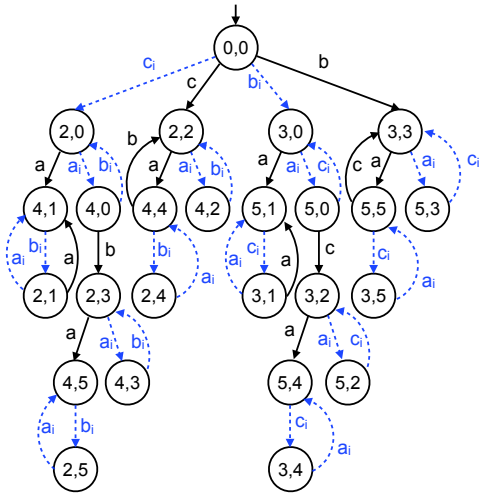
(a) The current-state estimator $\mathscr{E}$; state 1 reveals the secret.



(b) The desired estimator $\mathscr{E}^d$.



(c) The feasible estimator $\mathscr{E}^f$.



(d) The i-verifier V.

Fig. 5. The desired estimator, feasible estimator, and the i-verifier V constructed from the estimator in Figure 5(a).

with (dashed) transition $0 \xrightarrow{b_i} 0$ in Figure 5(c), resulting in (dashed) transition $(0,0) \xrightarrow{b_i} (3,0)$ in Figure 5(d).

Stage 2: Constructing the unfolded i-verifier V_u

Having V that identifies all safe insertions, we now "unfold" the structure of V and obtain the unfolded i-verifier V_u. Consider again the estimator $\mathscr{E}$ in Figure 5(a) and the i-verifier V in Figure 5(d). The unfolded i-verifier is shown in Figure 6.

The unfolded i-verifier V_u enumerates all safe insertions in a two-player game structure and thus can be used to identify insertions that do not react to all strings in $P[\mathscr{L}(G)]$. Specifically, V_u is a finite game structure describing the interaction between the system and the insertion function. It is a bipartite graph defined as an automaton $V_u = (Y \cup Z, E_o \cup M_v, f_{yz} \cup f_{zy}, y_0)$ that is built from $\mathscr{E}$ and V.

We follow the terminologies used in game theory and call the participants "players" and their decisions "actions". In this game structure, the first player is the "system player" G that moves at Y (square-shaped) states; the second player is "insertion function

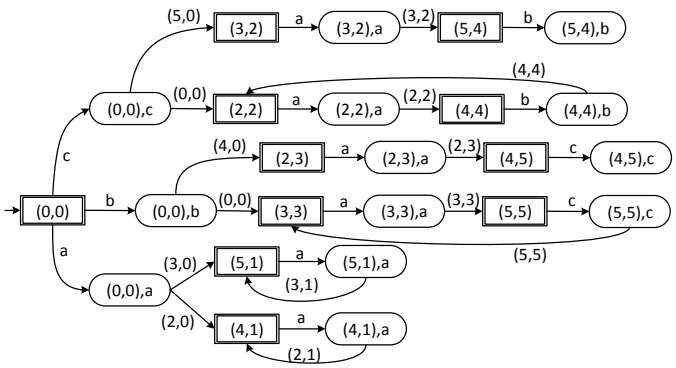


Fig. 6. The unfolded i-verifier V_u.

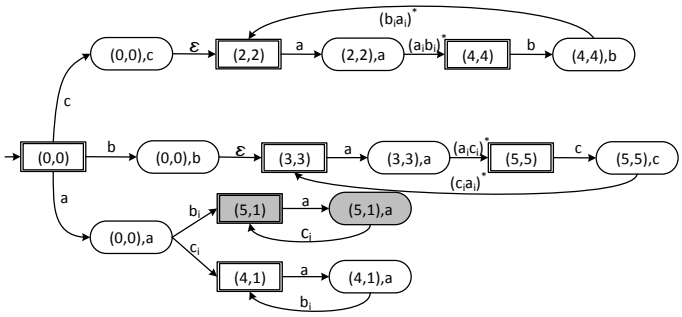


Fig. 7. The AIS. The subgraph without the shaded states is the obtained optimal strategy.

player" I that moves at Z (ellipse-shaped) states. Y and Z states are *information states* of players G and I, respectively. That is, each state contains enough information for the corresponding player to enumerate its actions. A given Y state, say y, is a state $m=(m_d,m_f)\in M_v$ in V and each action at y is an output event $e\in E_o$ from the system. Because m_f is the real state estimate of the system, we can examine all event transitions from m_f in $\mathscr{E}$ to enumerate the possible actions for player G. On the other hand, a given Z state, say $z=(y,e)$, consists of its predecessor state y and the action of observable event e that player G has just made. Because a Y state is also an M_v state in V, we also write $z=(m,e)$. Each action at $z=(m,e)$ is a state $m'\in M_v$ in V that compactly represents a set of inserted strings given by function $Ins(m,m')=\{s_I\in E_i^*:\delta_v(m,s_I)=m'\}$ where $m,m'\in M_v$. To enumerate all safe insertion strings, we can search on V using m and e. The transition function from Y to Z is denoted by $f_{yz}:Y\times E_o\to Z$, and the transition function from Z to Y is denoted $f_{zy}:Z\times M_v\to Y$. As the system is the first player, the initial state of V_u is defined as $y_0=m_{v,0}$. The formal procedure for constructing V_u is presented in Algorithm 1.

Algorithm 1: Construct V_u

input : $V=(M_v,E_o\cup E_i,\delta_v,m_{v,0})$ and $\mathscr{E}=(X_{\mathscr{E}},E_o,f_{\mathscr{E}},x_{\mathscr{E},0})$
output: $V_u=(Y\cup Z,E_o\cup M_v,f_{yz}\cup f_{zy},y_0)$

$y_0:=m_{v,0}$, $Y:=\{y_0\}$
for $y=(m_d,m_f)\in Y$ *that have not been examined* **do**
  for $e\in E_o$ **do**
    if $f_{\mathscr{E}}(m_f,e)$ *is defined* **then**
      $f_{yz}(y,e):=(y,e)$
      $Z:=Z\cup f_{yz}(y,e)$
for $z=(y,e)=(m,e)\in Z$ *that have not been examined* **do**
  for $m'\in M_v$ **do**
    if $\exists s_I\in E_i^*$ *such that* $m'=\delta_v(m,s_I)$ *and* $\delta_v(m',e)$ *is defined* **then**
      $f_{zy}(z,m'):=\delta_v(m',e)$
      $Y:=Y\cup f_{zy}(z,m')$
Go back to step 2; repeat until all accessible part has been built

Stage 3: Pruning and obtaining the All Insertion Structure (AIS) Once we obtain V_u, we can prune the insertions that do not react to all strings in $P[\mathscr{L}(G)]$. Observe that there can be blocking Z states in V_u (e.g., $z=((4,5),c)$ in Figure 6). Because player I plays at Z states, a blocking state $z\in Z$ means that player I cannot respond to player G when it reaches z; that is, insertions that lead to z cannot enforce opacity for the event that was just output by the system. Therefore, we need to prune z away. When we prune away z, we must also prune away its incoming actions. However, its incoming actions should not be pruned because they are actions of player G. Thus, we have to prune some earlier insertion actions in order to prevent the AIS from generating this deadlocked state z. But such earlier pruning may create other deadlocked Z states. Hence, this pruning process needs to be performed iteratively until no deadlocked states exist. Finally, we relabel all the insertion actions by the corresponding set of inserted strings and obtain the AIS. The reader can find in Figure 7 the AIS that is pruned from V_u in Figure 6.

The pruning process can be formally formulated as an instance of the "Basic Supervisory Control Problem - Nonblocking Case" (BSCP-NB), in the terminology of [5]. The formal construction is presented in Algorithm 2. Specifically, given V_u, we *mark* all Y states and leave all Z states unmarked, as the insertion function player completes inserting strings at Y states. In the context of BSCP-NB, we let the system's output events (E_o) be *uncontrollable* events and insertion actions (M_v) be *controllable* events as the insertion function player can only control the insertion actions. The specification language for BSCP-NB is the language marked by the trimmed automaton of V_u, denoted by V_u^{trim}. (The "trim" operation consists of taking the accessible and co-accessible part of an automaton.) The desired AIS is then obtained as the solution of BSCP-NB, i.e., it is the minimally restrictive nonblocking supervisor of V_u, which is a sub-automaton of V_u by construction in this problem instance. In the algorithm, the notation $Y|_{AIS}$ means the restriction of the set Y to the AIS.

Algorithm 2: Construct the AIS

input : $V_u=(Y\cup Z,E_o\cup M_v,f_{yz}\cup f_{zy},y_0)$
output: AIS $=(Y\cup Z,E_o\cup 2^{E_i^*},f_{AIS,yz}\cup f_{AIS,zy},y_0)$

Mark all the Y states in V_u
Let E_o be uncontrollable and M_v be controllable
Trim V_u and let V_u^{trim} be the specification automaton
Obtain the AIS as the automaton obtained from $[\mathscr{L}_m(V_u^{trim})]^{\uparrow C}$ w.r.t $\mathscr{L}(V_u)$ by following the $\uparrow C$ algorithm in [5]
for $z\xrightarrow{m'}y$ *where* $z=(m,e)\in Z|_{AIS}$ *and* $y\in Y|_{AIS}$ **do**
  Replace transition label m' by $Ins(m,m')$
Return the AIS

Notice that, in Algorithm 2, the AIS can be the empty automaton when $[\mathscr{L}_m(V_u^{trim})]^{\uparrow C}=\emptyset$. Moreover, we can verify that the AIS obtained by Algorithm 2 has the same properties as the AIS^e in [22] in terms of: (i) enumerating all i-enforcing insertion functions; and (ii) the AIS is not the empty automaton if and only if the given opacity property is i-enforceable. These results follow from the fact that, by construction, each state in the AIS is an information state

that contains all the information needed for the corresponding player. The key difference between the AIS and the AIS^e is the information contained in the information states. The state space reduction is achieved by removing redundant information from the information states in the AIS^e. Specifically, the construction of the meta-observer in [22] potentially introduced states in the AIS^e that had the same future behavior. These states are now merged in the AIS. An example contrasting the AIS^e with the AIS is given in Chapter IV.8 of [20].

B. Complexity of the Construction Procedure

We use current-state opacity to analyze the space complexity for the construction of the AIS. Given G with $|X|$ states, the current-state estimator has at most $|X_{\mathscr{E}}| = 2^{|X|}$ states. We build the i-verifier V by dashed parallel composing two special current-state estimators; hence, V has at most $|M_v| = |X_{\mathscr{E}}|^2$ states. The unfolded i-verifier V_u, which uses M_v states to enumerate actions, has worst-case state space complexity $|X_{v_u}| = |M_v| + |E_o||M_v|$. Specifically, the first and the second terms account for the information states of player G and player I, respectively. Finally, the AIS has at most $|X_{v_u}|$ states because the AIS is the recognizer for $\mathscr{L}_m(V_u^{trim})^{\uparrow C}$. In all, the space complexity of the AIS is $O\big((|E_o|+1)|X_{\mathscr{E}}|^2\big)$. Because of the computation of connectivity between states of V in Stage 2, the time complexity is $O(|M_v|^3) = O(|X_{\mathscr{E}}|^6)$. The worst-case complexity of the AIS is polynomial while that of the AIS^e is exponential, both in the state space of the estimator.

C. Synthesis of One I-enforcing Insertion Function

The AIS embeds all i-enforcing insertion functions in a game structure. With the AIS, one can synthesize one random i-enforcing insertion function by selecting states and transitions on the AIS, in a breadth-first search manner. Specifically, we first select all outgoing transitions from every Y state and one random inserted string for every Z state, resulting in an *insertion strategy* that is also a sub-automaton of the AIS. Then we use this sub-automaton to synthesize an insertion automaton. The complexity for constructing one such insertion function is $O(|f_{AIS}|)$ where $|f_{AIS}|$ is the number of transitions in $f_{AIS} := f_{AIS,yz} \cup f_{AIS,zy}$. Note that $|f_{AIS}|$ is at most $|E_o||X_{\mathscr{E}}|^2 + |E_o||X_{\mathscr{E}}|^2|X_{\mathscr{E}}|^2$; the first and the second terms account for the transitions from Y states and Z states, respectively. For the formal procedure of this synthesis algorithm, we refer the interested reader to Algorithm 5 in [22]. The remainder of this paper will focus on synthesizing optimal insertion functions from the AIS.

V. Cost of An Insertion Function

To perform optimization on the AIS, we first define costs for all the insertion functions that are embedded in the transition structure of the AIS. Because our optimization procedures rely on finding optimal *insertion strategies* and since an insertion strategy uniquely defines an insertion function, we will define costs for insertion functions using insertion strategies.

Specifically, given a game, a player's strategy is a mapping from every game history where the player should move to an action. An *insertion strategy*, which is a strategy of player I, maps every path on the AIS that ends at a Z state to an outgoing edge of that Z state.

Definition 2 (Insertion Strategy)**.** *An insertion strategy on the AIS is a mapping $\pi : (E_o 2^{E_i^*})^* E_o \to 2^{E_i^*}$ that assigns an insertion action $L_i \in 2^{E_i^*}$ to every history where player I should play.*

We can represent an insertion strategy as a (possibly infinite-state) bipartite graph $H = (Y_H \cup Z_H, E_o \cup 2^{E_i^*}, f_H, y_{H,0})$ where each Y_H state enumerates *all* system output events and each Z_H state selects *one* insertion action. In some cases, the bipartite graph representation H can be obtained by selecting outgoing actions for states of the AIS; such an insertion strategy is called *AIS-state-based*, or simply *state-based* hereafter. In general, we can obtain H by splitting the state space of the AIS as necessary using standard automata procedures. Only finite-state Hs are considered for our problem domains in this paper. In the remainder of this section, we assume all Hs are finite. In Section VI, we will prove that there exists an optimal state-based strategy when an optimal one exists for the the considered cost objective.

A. The weight function w on the AIS

To define costs for insertion strategies, one needs to define a cost structure on the AIS. We begin with assigning a cost value to every inserted event. Cost function $c : E_i \to \{0,1,2,\dots C_{max}\}$ maps each inserted event to a finite natural number. The domain of c is extended to E_i^* in a recursive additive manner by defining $c(\varepsilon) = 0$ and $c(se) = c(s) + c(e)$ where $s \in E_i^*, e \in E_i$. With function c, Definition 3 that follows defines a weight function w on the transitions of the AIS. The weight value of a transition is the *minimum* insertion cost.

Definition 3 (Weight Function w on the AIS)**.** *Given the AIS $= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and cost function c, we define weight function $w : (Y \times E_o \times Z) \cup (Z \times 2^{E_i^*} \times Y) \to \{0,1,2,\dots W_{max}\}$ that maps each transition to its minimum insertion cost. Specifically, $w(y \xrightarrow{e_o} z) = 0$ and $w(z \xrightarrow{L_i} y) = \min\{c(s_I) : s_I \in L_i\}$ where $y \in Y, z \in Z, e_o \in E_o$ and $L_i \in 2^{E_i^*}$ is the set of inserted strings that labels the given transition.*

Note that transition $y \xrightarrow{e_o} z$ has weight zero as e_o is a system output event that contains no inserted event. For transition $z \xrightarrow{L_i} y$, we select from set L_i one string that achieves the minimum cost and assign that cost to the transition. This minimum is always well defined since there is at least one insertion string that is bounded in length. Also, recall from Algorithm 2 that the above L_i's are obtained from $Ins(m,m')$, which contains all dashed (i.e., inserted) strings between states m and m' in V. By selecting from set L_i the string that achieves the minimum cost, we upper bound W_{max} by $k_d C_{max}$, where k_d is the longest dashed path in V.

Example 1. *Consider the AIS in Figure 7. We calculate the weight function w with respect to cost function $c(a_i) = c(b_i) = 1, c(c_i) = 2$. Every $y \xrightarrow{e_o} z$ has a zero weight value. For $z \xrightarrow{L_i} y$, we find the minimum cost among all strings in L_i. Specifically, for transition $((2,2),a) \xrightarrow{(a_i b_i)^*} (4,4)$, we find the minimum insertion cost $c(\varepsilon) = 0$ and assign the transition weight to zero. Other transitions that are labeled with $(a_i b_i)^*$ or $(a_i c_i)^*$ are also assigned zero weight. For transitions labeled with b_i or c_i, we assign them weight 1 or 2, respectively. Finally, the AIS can be represented as the weighted graph in Figure 8, after the state names are relabeled by numbers and transitions are labeled only by the edge weight.*

B. The Maximum Total Cost of An Insertion Strategy

We now extend the domain of weight function w to *paths* on the AIS and calculate the total cost of a path on the AIS.

Definition 4 (Paths)**.** *A path of k rounds that is generated by the AIS is a sequence of transitions ending at a Y state: $p = y_0 \xrightarrow{e_1} z_0 \xrightarrow{L_1} y_1 \xrightarrow{e_2} z_1 \xrightarrow{L_2} \cdots y_{k-1} \xrightarrow{e_k} z_{k-1} \xrightarrow{L_k} y_k$, where $e_{i+1} \in E_o$, $L_{i+1} \in 2^{E_i^*}$, $z_i = f_{AIS,yz}(y_i, e_{i+1})$, and $y_{i+1} = f_{AIS,zy}(z_i, L_{i+1})$ for $0 \le i \le k-1$. The set of paths generated by the AIS is $\mathscr{P}aths(AIS) := \cup_{k \ge 0} \mathscr{P}ath_k(AIS)$, where $\mathscr{P}ath_k(AIS)$ is the set of all k-round paths.*

Next, we define the total costs of paths.

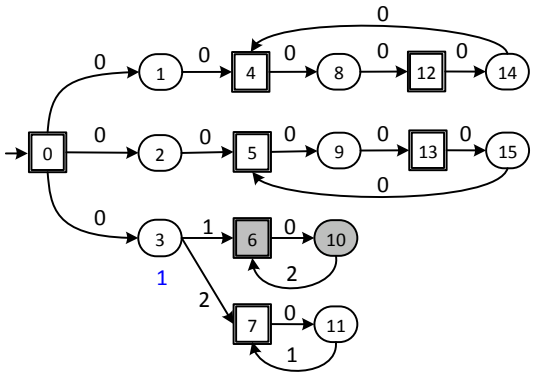


Fig. 8. The same AIS as in Figure 7 where information states are relabeled by numbers, transitions are labeled by the edge weights, and event labels are omitted. The blue number is the $\bar{c}^*(3)$ computed in Example 7.

Definition 5 (Total Cost of A Path)**.** *Consider path* $p = y_0 \xrightarrow{e_1} z_0 \xrightarrow{L_1} y_1 \xrightarrow{e_2} z_1 \xrightarrow{L_2} \cdots y_{k-1} \xrightarrow{e_k} z_{k-1} \xrightarrow{L_k} y_k \in \mathscr{P}ath_k(AIS)$*. The total cost of* p *is* $w(p) = \sum_{i=1}^{k}[w(e_i) + w(L_i)] = \sum_{i=1}^{k} w(L_i)$.

The last equality holds because $w(e_i) = 0,\ \forall e_i \in E_o$.

An insertion strategy has an infinite number of paths in general; each path leads to a total cost. We consider the worst-case scenario by looking at the largest total cost. This cost is the *maximum total cost* of the insertion function that the insertion strategy encodes.

Definition 6 (Maximum Total Cost of An Insertion Function)**.** *Given insertion function* f_I*, its maximum total cost is* $c_t(f_I) := c_t(H) := \limsup_{k\to\infty}\{\max\{w(p) : p \in S_k\}\}$*, where* $S_k := \{p : p \in \cup_{i=0}^{k} \mathscr{P}ath_i(AIS)|_H)\}$ *and* H *is the insertion strategy that uniquely defines* f_I.

The notation $\mathscr{P}ath_k(\text{AIS})|_H$ refers to the restriction of H to k-round paths. We consider the limit superior because the system generates arbitrarily long strings in general and the total cost may not converge.

C. Calculation of the Maximum Total Cost

Algorithm 3 computes the maximum total cost for an insertion function. Given insertion function f_I, we first construct insertion strategy H by selecting actions on the AIS according to f_I and splitting the state space of the AIS when needed. We then compute $c_t(f_I)$ directly on the structure of H. In the algorithm we denote by $n_H := |Y_H \cup Z_H|$ the cardinality of the state space of H.

Algorithm 3: The maximum total cost of f_I

input : Insertion function f_I and cost function c
output: $c_t(f_I)$

1 Encode f_I as insertion strategy $H = (Y_H \cup Z_H, E_o \cup 2^{E_i^*}, f_H, y_{H,0})$
2 Find all strongly connected components $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ on H
3 **for** $C_i \in \mathscr{C}$ **do**
  if $\exists u \to u' \in C_i$ *s.t.* $w(u \to u') \neq 0$ **then**
    Return ∞
4 **for** $u \in Y_H \cup Z_H$ **do**
  Calculate $V_{|n_H|-1}(u)$ where
  $\forall u, V_k(u) := \max_{u \to u'}\{w(u \to u') + V_{k-1}(u')\}$ and $V_0(u) := 0$
5 Return $V_{|n_H|-1}(y_{H,0})$

The maximum total cost $c_t(f_I)$ is the maximum cost-to-go from the initial state of H. First, we determine if H contains any strongly-connected component (SCC) that has a non-zero edge weight. If there exists a non-zero-cost SCC (i.e., an SCC whose sum of all edge weights is non-zero), then there exists a path that loops in that SCC and incurs an infinite cost-to-go. In this case, we can immediately return $c_t(f_I) = \infty$. If, otherwise, there is no non-zero-cost SCC, then the maximum cost-to-go from the initial state equals the maximum *simple-path* cost-to-go from the initial state. Hence, we iteratively compute the $(|n_H| - 1)$-step cost-to-go from the initial state.

Denote by $|f_H|$ the number of transitions in H. In Algorithm 3, step 2 can be computed by using Tarjan's strongly connected components algorithm [19], which runs in $O(|n_H| + |f_H|)$. Step 4 can be computed in $O(|n_H||f_H|)$. Thus, $c_t(f_I)$ can be computed in $O(|n_H||f_H|)$.

Example 2. *Consider insertion function* f_I *encoded by the insertion automaton in Figure 2. We construct insertion strategy* H *that defines* f_I *by selecting in Figure 7 all states but the shaded states and selecting* ε *for transitions labeled by* $(b_i a_i)^*, (a_i b_i)^*, (a_i c_i)^*$*, or* $(c_i a_i)^*$*. Given cost function* $c(a_i) = 1, c(b_i) = 0, c(c_i) = 2$*, we calculate* $c_t(f_I)$ *by following Algorithm 3. First, we find all the SCC (i.e.,* $\{(2,2),((2,2),a),(4,4),((4,4),b)\}, \{(3,3),((3,3),a),(5,5),((5,5),c)\}, \{(4,1),((4,1),a)\}$*) on* H*. Because the edge weights of all the SCCs are zero, we then iteratively compute* $V_{|n_H|-1}(0) = V_{13}(0) = 2$*. Thus,* $c_t(f_I) = 2$*. Now, let us change* $c(b_i) = 1$*. The edge weights of the AIS are shown in Figure 8. The SCC* $\{(4,1),((4,1),a)\}$ *(or* $\{7,11\}$ *in Figure 8) with the new cost function becomes a non-zero-cost self-loop. Hence, Algorithm 3 returns infinity.*

D. The Maximum Mean Cost of An Insertion Strategy

Consider insertion function f_I for system G. The maximum total cost $c_t(f_I)$ always exists when G generates only strings of finite length. However, if G has a cycle, f_I may insert one or more events when G loops in that cycle, thereby resulting in an infinite $c_t(f_I)$. In this case, we compute the *maximum mean cost* of f_I, denoted by $\bar{c}(f_I)$, which considers the average insertion cost per system output event.

Definition 7 (Maximum Mean Cost of An Insertion Function)**.** *Given insertion function* f_I*, the maximum mean cost is* $\bar{c}(f_I) := \bar{c}(H) := \limsup_{k\to\infty}\{\max\{\frac{1}{k}w(p) : p \in S_k\}\}$*, where* $S_k := \{p : p \in \cup_{i=0}^{k} \mathscr{P}ath_i(AIS)|_H)\}$ *and* H *is the insertion strategy that uniquely defines* f_I.

The limit superior is taken as the maximum mean cost may not converge in the limit. If all paths on H are of finite length, then $\bar{c}(H) = 0$.

E. Calculation of the Mean Cost

We calculate $\bar{c}(f_I)$ on its insertion strategy H by treating H as a weighted graph using the given weight function. As seen in Definition 7, the maximum mean cost is defined in terms of *rounds*. Since a round corresponds to two steps on H, the maximum mean cost of f_I is *double* of the "maximum mean weight" of H in the terminology of weighted graphs. We can calculate $\bar{c}(f_I)$ using the version of Karp's Theorem presented in [1] for *maximum* mean weight.[1]

Theorem 1. *(Karp's Theorem [1]) Consider a weighted directed graph* (X, f) *with* $|X| = n$*, where* X *is the set of vertices and* f *is the set of edges. The maximum mean weight for a given initial vertex* x_0 *is,*

$$\lambda^* = \max_{x \in X} \min_{0 \le k \le n-1} \frac{F_n(x) - F_k(x)}{n - k} \tag{1}$$

[1]The original version of Karp's Theorem in [12] is for *minimum* mean weight.

where $F_k(x)$ is the maximum weight of an edge progression of length k from x_0 to x.

With the maximum mean weight of H defined, we now compute $\bar{c}(f_I)$ in Algorithm 4. The computation finishes in $O(|n_H|+|f_H|)$.

Algorithm 4: The maximum mean cost of f_I

input : Insertion function f_I and cost function c
output: $\bar{c}(f_I)$

1 Encode f_I as insertion strategy $H=(Y_H\cup Z_H, E_o\cup 2^{E_i^*}, f_H, y_{H,0})$
2 Compute the maximum mean weight λ^* of weighted graph $(Y_H\cup Z_H, f_H)$ using Equation (1)
3 Return $2\lambda^*$

Example 3. *Consider again the insertion function f_I in Figure 2; we construct insertion strategy H for f_I by removing the shaded states in Figure 7. Let the cost function be $c(a_i)=c(b_i)=1, c(c_i)=2$. We have shown in Example 2 that $c_t(f_I)$ goes to infinity. Here, we calculate the maximum mean cost using Algorithm 4. Consider the weighted graph $(Y_H\cup Z_H, f_H)$ of H, as shown in Figure 8 without the shaded states. The maximum mean weight is $\lambda^*=\frac{F_{14}(7)-F_{12}(7)}{14-12}=\frac{7-6}{2}=\frac{1}{2}$. Thus, $c_t(f_I)=2\lambda^*=1$. The insertion function costs one per system output in the worst case.*

VI. Synthesis of An Optimal Finite-Cost Insertion Function

We have introduced the maximum total cost and the maximum mean cost for insertion functions. Given G that is not opaque, we want to find an optimal insertion function with respect to each cost. The two optimization problems are formulated as follows. In the problem statements, we use $H\in$ AIS to denote that insertion strategy H is obtained from the AIS after potential state splitting.

Problem 1. *Consider G that is not opaque and cost function c for inserted events. Find:*
(a) the optimal maximum total cost $c_t^*=\min\{c_t(H): H\in AIS\}$
(b) an optimal total-cost insertion function that achieves c_t^*

Problem 2. *Consider G that is not opaque and cost function c for inserted events. Find:*
(a) the optimal maximum mean cost $\bar{c}^*=\min\{\bar{c}(H): H\in AIS\}$
(b) an optimal mean-cost insertion function that achieves $\bar{c}^*$

Notice that if c_t^* is finite, there is no need to solve Problem 2 as $\bar{c}^*$ is known to be zero and an optimal total-cost insertion function is an optimal mean-cost insertion function. Hence, our goal is to synthesize an optimal total-cost insertion function if c_t^* is finite, and an optimal mean-cost insertion function otherwise. Problem 1 is solved in Sections VI-A to VI-C while Problem 2 is solved in Section VII.

A. Minimax Game Formulation for An Optimal Total-Cost Insertion Function

Recall that the AIS is a game structure that enumerates, in alternate turns, the actions of the system player and those of the insertion function player. To solve Problem 1, we consider a *minimax game* on the AIS and find an optimal insertion strategy. In the minimax game, the system player tries to maximize $\liminf_{k\to\infty}\sum_{i=1}^k w(u\xrightarrow{e}u')$ and the insertion function player tries to minimize $\limsup_{k\to\infty}\sum_{i=1}^k w(u\xrightarrow{e}u')$, where $u,u'\in Y\cup Z$ and $e\in E_o\cup 2^{E_i^*}$. Because the optimal maximum total cost c_t^* is defined in terms of the worst-case scenario, it is indeed the resulting $\limsup_{k\to\infty}\sum_{i=1}^k w(u\xrightarrow{e}u')$ in the minimax game. Moreover, the optimal insertion strategy is the resulting strategy that minimizes $\limsup_{k\to\infty}\sum_{i=1}^k w(u\xrightarrow{e}u')$. As we will show later, there is an optimal state-based insertion strategy, meaning that the strategy can be represented as a subgraph of the AIS. We will use the subgraph to construct an optimal insertion function in Section VI-C.

B. Finding the Optimal Total Cost

The system we consider generates arbitrarily long strings in general, and thus the game described by the AIS is infinite horizon. In this section, we solve Problem 1 by solving a *finite-horizon minimax game* played on the AIS, where player P_1 maximizes the total cost at Y states and player P_2 minimizes the total cost at Z states, for a finite number of steps. Specifically, we use the optimal total cost for the finite-horizon minimax game to determine c_t^*, and then use c_t^* to find an optimal insertion strategy.

Consider the AIS $=(Y\cup Z, E_o\cup 2^{E_i^*}, f_{AIS,yz}\cup f_{AIS,zy}, y_0)$. Denote by $V_k(u)$ the optimal total cost and by $a_k(u)$ an optimal action in the k-step game assuming the game starts at state u, where $u\in Y\cup Z$. We calculate the cost and the action using the following recursive equations.

$$V_k(u)=\begin{cases}\max\limits_{(u\to u')}\{w(u\xrightarrow{e}u')+V_{k-1}(u')\}, \text{ if } u\in Y\\ \min\limits_{(u\to u')}\{w(u\xrightarrow{e}u')+V_{k-1}(u')\}, \text{ if } u\in Z\end{cases} \tag{2}$$

where $V_0(u)=0$ for $u\in Y\cup Z$.

$$a_k(u)=\begin{cases}\arg\max\limits_{(u\to u')}\{w(u\xrightarrow{e}u')+V_{k-1}(u')\}, \text{ if } u\in Y\\ \arg\min\limits_{(u\to u')}\{w(u\xrightarrow{e}u')+V_{k-1}(u')\}, \text{ if } u\in Z\end{cases} \tag{3}$$

Note that the strategy found using Equation (3) is not state-based in general, as the optimal action depends also on k.

It turns out that finite-horizon minimax games can be used to analyze the *infinite-horizon minimax game*. Denote by $V(y_0)$ the optimal total cost and by π^* an optimal strategy for P_1 and P_2 in the infinite-horizon minimax game.[2] Let n be the number of states in the AIS. We will prove in Theorem 2 that $V(y_0)$ and a state-based π^* can be found in the n^2W_{max}-step game. In the following, we let $V_k^\pi(y_0)$ be the total cost in the k-step game when the game starts at y_0 and strategy π is used, and let π_k^* be an optimal strategy in the k-step game.

Lemma 1. *Let $l=n^2W_{max}$. If $V_l(y_0)<nW_{max}$, then there exists a state-based optimal strategy π_l for the l-step game such that $V_{l'}^{\pi_l}(y_0)=V_l^{\pi_l}(y_0), \forall l'\geq l$.*

Proof: Consider outcome path p_l on the AIS, resulting from the players playing π_l^*. Label actions on p_l by $a_1a_2\dots a_l$. We will first construct a state-based strategy π_l that is as good as π_l^*. Then, we show that if π_l is used for l'-step game ($\forall l'\geq l$), the optimal total cost would be the same as that for the l-step game.

Partition p_l into $a_1\dots(a_{q_1}\dots a_{r_1})\dots(a_{q_2}\dots a_{r_2})\dots(a_{q_{N_c}}\dots a_{r_{N_c}})\dots a_l$, where $(a_{q_i}\dots a_{r_i})$ is the i-th cycle C_i on p_l, as shown in Figure 9. Because a cycle is formed within at most n steps, $N_c\geq l/n=n^2W_{max}/n=nW_{max}$. First, we argue that each of these cycles has a zero cycle cost. Suppose there exists a non-zero cost cycle C_{nz} (say C_1). If x_{q_1} is played by P_1, then P_1 has a better strategy for which the outcome path reaches C_{nz} and loops there until step l. In this case, the number of times in C_{nz} must be greater than N_c and the path would incur a total cost greater than nW_{max}. This contradicts the hypothesis

[2]Only one player will move at a given history of the game. Thus, we can treat the strategies of the two players as a single strategy.

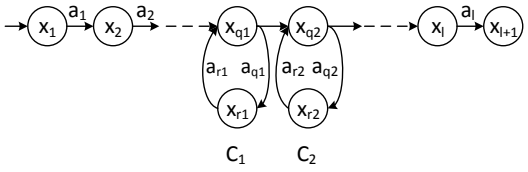


Fig. 9. An outcome path used to illustrate the proofs of Lemmas 1 and 2.

in the statement of the lemma that $V_l(y_0) < nW_{max}$ and that p_l is the optimal outcome path. On the other hand, if x_{q_1} is played by P_2, then P_2 has a better strategy for which the outcome path skips C_{nz} and loops in a zero-cost cycle for the extra steps. This also contradicts the hypothesis that p_l is the optimal outcome path. Hence, all the cycles have zero-cost. Then, we argue that actions $a_{r_{N_c}+1}, \ldots, a_l$ are also zero-cost using a similar reasoning. Suppose any of them has a non-zero cost. Then, P_1 has a better strategy for which the outcome path skips C_1 to C_{N_c} and loops in a cycle containing $a_{r_{N_c}+1}, \ldots, a_l$. Also, P_2 has a better strategy that loops in zero-cost cycle C_{N_c} until step l. Either case contradicts the hypothesis that p_l is the optimal outcome path. Hence, $a_{r_{N_c}+1}, \ldots, a_l$ are zero-cost.

Now, let us construct a new path p'_l from p_l as follows. First, find on p_l the smallest i such that all actions after a_{r_i} have a zero cost. Then, remove cycles $C_1, \ldots C_{i-1}$ and all actions after a_{r_i}. The resulting p'_l is $a_1 \ldots a_{q_1-1} a_{r_1+1} \ldots a_{q_2-1} a_{r_2+1} \ldots (a_{q_i} \ldots a_{r_i})$, which ends in cycle C_i. Define state-based strategy π_l by assigning to each state the only outgoing action according to p'_l. Use π_l as the strategy for the l-step game. The resulting outcome path would begin with p'_l as a subpath and then loop in C_i until step l. Because this path differs in p_l only in the replacement of some zero-cost actions, the corresponding total cost is the same as that for p_l. That is, $V_l^{\pi_l}(y_0) = V_l(y_0)$. Therefore, π_l is an optimal strategy for the l-step game.

Finally, because C_i is a zero-cost cycle, using π_l for any l'-step game ($l' \geq l$) would not increase the total cost as the outcome path will cycle in C_i. Hence, $V_{l'}^{\pi_l}(y_0) = V_l^{\pi_l}(y_0), \forall l' \geq l$. ■

Lemma 2. *Let $l = n^2 W_{max}$. If $V_l(y_0) \geq nW_{max}$, then $V_{k+n}(y_0) > V_k(y_0), \forall k \geq 0$.*

Proof: Consider the same setups used in Lemma 1. We partition p_l again into $a_1 \ldots (a_{q_1} \ldots a_{r_1}) \ldots (a_{q_2} \ldots a_{r_2}) \ldots (a_{q_{N_c}} \ldots a_{r_{N_c}}) \ldots a_l$. This time, it is hypothesized that $V_l(y_0) \geq nW_{max}$. We will argue that all the cycle costs are non-zero.

If all the cycles are zero cost, then $V_l(y_0) < nW_{max}$ because at most $n-1$ steps do not belong to a cycle. But this violates the hypothesis of the lemma. Suppose there is a zero-cost cycle C_z (say C_2). Then, if x_{q_2} is played by P_1, then P_1 can do better by skipping C_z and using the extra steps on a non-zero cost cycle. On the other hand, if x_{q_2} is played by P_2, then P_2 can do better by staying in C_z until step l. Both cases contradict our assumption that p_l is an optimal outcome path. Therefore, all the cycle costs on p_l must be non-zero.

Consider the k-step game for a given $k \geq 0$ and let p_k be an optimal outcome path for the k-step game. If $k < l$, all the cycle costs on p_k must be non-zero. Otherwise, for P_1, using the first k steps of p_l results in a better strategy; for P_2, using p_k for the first k step of the l-step game results in a path better than p_l. If $k > l$, all the cycles on p_k must be non-zero cost as well. Otherwise, if there exists a zero-cost cycle, then P_1 could perform at least as well by skipping that cycle and using the extra steps on a non-zero cost cycle; P_2 would prefer to stay in that zero-cost cycle. Also, if all the cycles are zero-cost, then $V_k(y_0) < nW_{max}$, which would lead to the wrong conclusion that $V_k(y_0) < V_l(y_0)$. Therefore, if $V_l(y_0) \geq nW_{max}$, then all the cycles on p_k are non-zero cost, $\forall k \geq 0$.

Now, let us compare $V_{k+n}(y_0)$ and $V_k(y_0)$. We partition p_k and p_{k+n} into segments of cycles, as we have done for p_l. Because a cycle is formed within n steps, p_{k+n} must have at least one more cycle than p_k. Since all the cycles are non-zero cost, $V_{k+n}(y_0) > V_k(y_0)$. ■

Theorem 2. *Let $l = n^2 W_{max}$. If $V_l(y_0) < nW_{max}$, then $V(y_0) = V_l(y_0)$ and is achieved by a state-based optimal strategy π_l for the l-step game. Otherwise, $V(y_0)$ goes to infinity.*

Proof: When $V_l(y_0) < nW_{max}$, we have constructed in Lemma 1 a state-based optimal strategy π_l for the l-step game. Use π_l for the infinite game. We have $\lim_{l' \to \infty} V_{l'}^{\pi_l}(y_0) = V_l(y_0) \leq \lim_{l' \to \infty} V_{l'}^{\pi^*}(y_0)$. The first equality is according to Lemma 1; the second inequality is because π^* is an optimal strategy for the infinite game.

Now, use π^* for the l-step game. Because π_l is the optimal strategy for the l-step game, we have $V_l^{\pi^*}(y_0) \leq V_l(y_0) < nW_{max}$. Partition the optimal outcome path p_∞ for the infinite game into $a_1 \ldots (a_{q_1} \ldots a_{r_1}) \ldots (a_{q_2} \ldots a_{r_2}) \ldots$, where $(a_{q_i} \ldots a_{r_i})$ is the i-the cycle C_i on p_∞. Because p_∞ is an optimal outcome path, the cycle costs must be either all zero or all non-zero. Otherwise, it would not be optimal, by the same argument as in the proofs of Lemmas 1 and 2. Since $V_l^{\pi^*}(y_0) < nW_{max}$, all cycles in the first l steps must be zero-cost according to Lemma 1. Thus, all cycle costs on p_∞ are zero and there are at most $n-1$ non-zero cost edges on p_∞. Now, let us construct p'_∞ from p_∞ by skipping cycles until the last non-zero-cost edge and then looping in a zero-cost cycle for the extra steps. The total cost of p'_∞ is the sum of the first $n-1$ steps, which equals that of p_∞. That is, p'_∞ corresponds to another optimal strategy π'^* for the infinite game. Consequently, we have $V(y_0) = \lim_{l' \to \infty} V_{l'}^{\pi'^*}(y_0) = V_{n-1}^{\pi'^*}(y_0) = V_l^{\pi'^*}(y_0) \leq V_l^{\pi_l}(y_0) = \lim_{l' \to \infty} V_{l'}^{\pi_l}(y_0)$.

Finally, combine the above two inequalities. We obtain $V(y_0) := \lim_{l' \to \infty} V_{l'}^{\pi'^*}(y_0) = \lim_{l' \to \infty} V_{l'}^{\pi_l}(y_0) = V_l^{\pi_l}(y_0) = V_l(y_0)$; state-based strategy π_l is optimal for the infinite game and $V(y_0) = V_l(y_0)$.

On the other hand, when $V_l(y_0) \geq nW_{max}$, we have $V_{k+n}(y_0) > V_k(y_0), \forall k \geq 0$ by Lemma 2. Take k to infinity; the optimal total cost for the infinite game $V(y_0)$ goes to infinity. ■

We have proven in Theorem 2 that $c_t^* = V(y_0)$ can be calculated in a finite-horizon minimax game. Hence, we now combine all the above results and compute c_t^* in Algorithm 5. Notice that if the AIS is acyclic, then $V(y_0)$ will converge within n steps and $V(y_0) < nW_{max}$; that is, $c_t^* < \infty$.

Algorithm 5: The optimal total cost c_t^*

input : AIS $= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and weight function w
output: c_t^*

Compute $V_l(y_0)$ for $l = n^2 W_{max}$ using Equation (2)
if $V_l(y_0) < nW_{max}$ **then**
  Return $V_l(y_0)$
else
  Return ∞

Example 4. *Let us consider the AIS in Figure 7 with cost function $c(a_i) = 1$, $c(b_i) = 0$, $c(c_i) = 2$. We compute c_t^* by following Algorithm 5. In step 1, $V_l((0,0)) = 2$, where $l = n^2 W_{max} = 16^2 \cdot 2 = 512$. Because $V_l((0,0)) < nW_{max} = 32$, we have $c_t^* = V_l((0,0)) = 2$. Now, if we change $c(b_i) = 1$, then $V_l((0,0)) = 257 > nW_{max} = 32$. Therefore, c_t^* goes to infinity in this case.*

Corollary 1. *There exists optimal state-based total-cost insertion functions if and only if $V_l(y_0) < nW_{max}$.*

Proof: The proof follows directly from Theorem 2. ■

C. Synthesis of the Optimal Total-Cost Insertion Function

Algorithm 5 calculates c_t^*. If $c_t^* < \infty$, then there exists an optimal total-cost insertion function and we will synthesize one using Algorithm 6. Otherwise, we go to Section VII and solve Problem 2.

When $V_l(y_0) < nW_{max}$, by Corollary 1, there is an optimal state-based strategy. In Algorithm 6 that follows, we build optimal state-based strategy H that selects all the actions at Y states and optimal actions at Z states from the AIS, resulting in a subgraph of the AIS. Note that Algorithm 6 computes an optimal insertion strategy in a breadth-first manner, thereby ignoring Z states that are never reached.

Algorithm 6: Find an optimal total-cost insertion strategy

input : AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and weight function w
output: Optimal strategy $H = (Y_H \cup Z_H, E_o \cup 2^{E_i^*}, f_H, y_{H,0})$

1 **for** $u \in Y \cup Z$ **do**
  Compute $a_l(u)$ for $l = n^2 W_{max}$ using Equation (3)
2 $y_{H,0} := y_0$, $Y_H := \{y_{H,0}\}$
3 **for** $u \in Y_H$ *that has not been examined* **do**
  for $e \in E_o$ **do**
    $f_H(u,e) := f_{AIS,yz}(u,e)$ if $f_{AIS,yz}(u,e)$ is defined
4 **for** $u \in Z_H$ *that has not been examined* **do**
  $e := a_l(u)$ the optimal action for u
  $f_H(u,e) := f_{AIS,yz}(u,e)$
5 Go back to step 2 until all selected states have been examined

Once we obtain optimal state-based insertion strategy H from Algorithm 6, we build an insertion function from H. Without loss of generality, the insertion function is encoded as an insertion automaton, using Algorithm 7.

Algorithm 7: Construct an insertion automaton from an insertion strategy

input : $H = (Y_H \cup Z_H, E_o \cup 2^{E_i^*}, f_H, y_{H,0})$, and weight function w
output: IA$= (X_{ia}, E_o, E_i^* E_o, f_{ia}, q_{ia}, x_{ia,0})$

1 $x_{ia,0} := y_{H,0}$, $X_{ia} := \{x_{ia,0}\}$
2 **for** $x \in X_{ia}$ *that has not been examined* **do**
  for $x \xrightarrow{e_o} z \xrightarrow{L_i} y$ *where* $x, y \in Y_H, z \in Z_H$ **do**
    $X_{ia} := X_{ia} \cup \{y\}$
    $s_I := \{s \in L_i : c(s) = w(z \xrightarrow{L_i} y)\}$
    $f_{ia}(x, e_o) := y$
    $q_{ia}(x, e_o) := s_I e_o$
3 Go back to step 2 until all states in X_{ia} have been examined

Example 5. *Consider again the AIS in Figure 7 with cost function $c(a_i) = 1,\ c(b_i) = 0,\ c(c_i) = 2$. We have computed $c_t^* = 2$ in Example 4 and concluded that an optimal total-cost insertion function exists. In this example, we want to synthesize an optimal total-cost insertion function. First, we apply Algorithm 6 to obtain an optimal strategy H. Specifically, we select all outgoing actions for square-shaped states, action $((0,0),a) \xrightarrow{c_i} (4,1)$ for state $((0,0),a)$, and the only actions for the other ellipse-shaped states. Such a selection results in an optimal strategy H that is the AIS in Figure 7 without the shaded states. Then, we apply Algorithm 7 by taking H as input to construct an insertion automaton. For each $y \xrightarrow{e_o} z \xrightarrow{L_i} y'$ where $y, y' \in Y|_H, z \in Z|_H$, we first find the inserted string s_I from Figure 7 as follows: select ε for transitions labelled with $(a_i b_i)^*$ or $(b_i a_i)^*$; select the only string for the other transitions. Then, using the chosen s_I, we redefine the transition to be $y \xrightarrow{e_o/s_I e_o} y'$. The resulting optimal insertion automaton is shown in Figure 10, with state names relabeled according to Figure 8.*

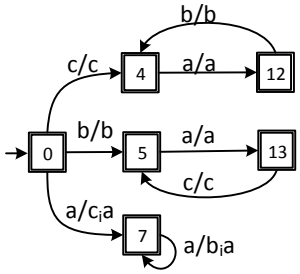


Fig. 10. The optimal IA in Examples 5 and 8, where the state names are relabeled according to Figure 8.

Theorem 3. *Applying Algorithms 5, 6 and 7 solves Problem 1(b).*

Proof: Algorithm 5 follows from Theorem 2. In Algorithm 6, an insertion action is chosen for every system output event. Hence, the resulting insertion strategy is i-enforcing. The strategy is optimal as all insertion actions are optimized. Since the insertion automaton in Algorithm 7 is constructed from the optimal strategy, it encodes an insertion function that achieves c_t^*. ■

In all, given the AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$, computing an optimal total-cost insertion function can be done in $O(n^2 |f_{AIS}| W_{max})$, where n is the number of states in the AIS and $|f_{AIS}|$ is the number of transitions in $f_{AIS} = f_{AIS,yz} \cup f_{AIS,zy}$. In terms of the size of the state estimator of the system, the complexity is $O(|E_o|^3 |X_{\mathscr{E}}|^8 W_{max})$.

VII. Synthesis of An Optimal Mean-Cost Insertion Strategy

A. Mean Payoff Game Formulation of the Synthesis Problem

To solve Problem 2, we solve a *mean payoff game* on the AIS. Similarly to Section VI, we let the AIS be our game structure. Here, the insertion function player tries to minimize $\limsup_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} w(u \xrightarrow{e} u')$ and the system player tries to maximize $\liminf_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} w(u \xrightarrow{e} u')$ where $u, u' \in Y \cup Z$ and $e \in E_o \cup 2^{E_i^*}$. The optimal maximum mean cost $\bar{c}^*$ is *double* of the resulting $\limsup_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} w(u \xrightarrow{e} u')$ because $\bar{c}^*$ is the worst-case average cost per *round*. Also, the optimal insertion strategy is the resulting strategy that minimizes $\limsup_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} w(u \xrightarrow{e} u')$. In a mean payoff game, both players have state-based optimal strategies [9]. Hence, the resulting optimal strategy will be a subgraph H of the AIS that selects all the actions of the system at Y states but only *one* optimal action at Z states. In the following, we first find the optimal mean cost $\bar{c}^*$ in Section VII-B, and then synthesize the optimal insertion function that achieves $\bar{c}^*$ in Section VII-C.

B. Finding the Optimal Mean Cost

We begin with solving Problem 2(a) and finding $\bar{c}^*$. This problem is a special instance of the problem of calculating the *value* of the mean payoff game on weighted automata that is defined in [9], and for which a general algorithmic solution is provided in [24]. Hence, we can compute $\bar{c}^*$ by adapting the results in [9] and customizing the algorithms in [24]. Specifically, here, the AIS is our weighted

automaton. Computing $\overline{c}^*$ on the AIS differs from computing the game value on a general weighted graph in the following aspects: (1) the game value is the average cost *per step in the game*, whereas $\overline{c}^*$ is the average cost *per round* (i.e., per system output event); (2) all edge weights in the AIS are non-negative while edge weights in [24] can be negative; and (3) every edge from a Y state of the AIS must have a zero weight while edge weights in [24] can be non-zero in general. Let us denote by $\bar{V}(u)$ the game value assuming that the game starts from state u. To find $\overline{c}^*$ on the AIS, we first address differences (2) and (3) by establishing in Theorem 4 a tighter bound for $\bar{V}(u)$, where u is a state of the AIS. This bound allows us to determine the correct value for $\bar{V}(u)$. Then, we address difference (1) by doubling the value of $\bar{V}(y_0)$ to obtain $\overline{c}^*$ in Algorithm 8.

Theorem 4. *For every state u of the AIS, we have*

$$\frac{V_k(u)}{k} - \frac{n-1}{2k}W_{max} \leq \bar{V}(u) \leq \frac{V_k(u)}{k} + \frac{n-1}{2k}W_{max}$$

Proof: The proof follows the reasoning in the proofs of Theorems 2.2 and 2.3 in [24] but it is adapted for the special cost structure of the AIS. Consider a k-step game and the outcome path that is resulted from players playing π_k^*. The resulting k-step cost-to-go from state u on the outcome path is $V_k(u)$. We have $V_k(u) \leq k\bar{V}(u) + \lceil \frac{n-1}{2} \rceil W_{max}$. The first term on the right-hand side is because at most k steps are in a cycle, and if P_2 plays according to its optimal strategy, then the average cost of that cycle is at most $\bar{V}(u)$. The second terms is because there can be at most $\lceil \frac{n-1}{2} \rceil$ non-zero steps before the cycle and each of them is at most W_{max}. Similarly, consider that P_1 plays according to its optimal strategy. We have $V_k(u) \geq (k-(n-1))\bar{V}(u) + 0$. On the right-hand side, the first term is because there are at least $k-(n-1)$ steps in the cycle; since all edge weights are non-negative, the second term, which is the minimum path weight before the cycle, is zero. Because $\bar{V}(u) \leq \frac{1}{2}W_{max}$, this inequality implies $V_k(u) \geq k\bar{V}(u) - \frac{n-1}{2}W_{max}$. By rearranging the above two inequalities, we have $\frac{V_k(u)}{k} - \frac{n-1}{2k}W_{max} \leq \bar{V}(u) \leq \frac{V_k(u)}{k} + \frac{n-1}{2k}W_{max}$. ∎

According to Theorem 4, as k increases, the bound for $\bar{V}(u)$ becomes tighter. Because each edge is assumed to have an integer cost value and that a cycle is formed within at most n steps, $\bar{V}(u)$ is a rational number with a denominator at most n. Hence, the minimum distance between two possible values of $\bar{V}(u)$ is $\frac{1}{n(n-1)}$. Now, let us choose $k = n^3W_{max}$. The value of $\bar{V}(u)$ is then bounded in $\frac{V_k(u)}{k} - \frac{1}{2n(n-1)} < \frac{V_k(u)}{k} - \frac{n-1}{2n^3} \leq \bar{V}(u) \leq \frac{V_k(u)}{k} + \frac{n-1}{2n^3} < \frac{V_k(u)}{k} + \frac{1}{2n(n-1)}$ where only one valid value exists. Therefore, we can determine $\bar{V}(u)$ by searching within $[V_h(y_0)/h - \frac{1}{2n(n-1)},\ V_h(y_0)/h + \frac{1}{2n(n-1)}]$ for $h = n^3W_{max}$. Notice that, because of the differences (2) and (3) pointed out above, the value of h is reduced by a factor of 4, compared to that in [24].

Finally, we find the optimal mean cost by doubling the value of $\bar{V}(y_0)$. The whole process is captured in Algorithm 8.

Algorithm 8: Find the optimal mean cost for state u

input : AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and the weight function w and state $u \in Y \cup Z$
output: $\overline{c}^*(u)$

1 Compute $V_h(u)$ for $h = n^3W_{max}$ using Equation (2)
2 Compute the h-step mean cost $V_h(u)/h$
3 Find the only rational number r with a denominator at most n that lies in the interval $[V_h(u)/h - \alpha,,\ V_h(u)/h + \alpha]$ with $\alpha = \frac{1}{2n(n-1)}$
4 Return $2r$

In Algorithm 8, $\overline{c}^*(u)$ is the optimal mean cost assuming that the game begins at state u. When $u = y_0$, the returned $\overline{c}^*(y_0)$ is the optimal mean cost $\overline{c}^*$. Note that we can compute $V_h(y_0)$ by continuing the computation of $V_l(y_0)$ in Algorithm 5, as $h = n^3W_{max}$ is greater than $l = n^2W_{max}$. In the next section, we will use $\overline{c}^*(u)$ to compute the optimal insertion function.

Example 6. *Consider again the weighted graph in Figure 8. We have shown in Example 4 that no optimal total-cost insertion function exists when the cost structure is* $c(a_i) = 1,\ c(b_i) = 1,\ c(c_i) = 2$. *Here, we will synthesize an optimal mean-cost insertion function. In this example, we first calculate the optimal mean cost by following Algorithm 8. Then, we will finish the synthesis in Examples 7 and 8. In step 1,* $V_h(0) = 4097$ *where* $h = n^3W_{max} = 16^3 \cdot 2 = 8192$. *Dividing* $V_h(0)$ *by h, we obtain the h-step mean cost* $V_h(0)/h = 0.50012$. *Finally, searching within the interval* $[\frac{4097}{8192} - \frac{1}{480}, \frac{4097}{8192} + \frac{1}{480}] = [0.498, 0.502]$, *we find that* $\frac{1}{2}$ *is the only valid value. The optimal mean cost is* $\overline{c}^* = 2 \cdot \frac{1}{2} = 1$.

C. Synthesis of the Optimal Infinite-Cost Insertion Function

With Algorithm 8 that solves $\overline{c}^*(u), \forall u \in Y \cup Z$, at hand, we now find an optimal action for a given Z state using Algorithm 9. This algorithm, adapted from [24], eliminates insertion actions using a binary search technique. Notice that Algorithm 9 is applied only to insertion states, i.e., Z states. For every state $z \in Z$, denote by $d(z)$ the number of outgoing actions at z. By construction, we have $d(z) \geq 1$ because there are no deadlocked states in AIS. Therefore, the algorithm always outputs a valid action when it terminates.

Algorithm 9: Find the optimal action for state $z \in Z$

input : AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$, weight function w, and a state $z \in Z$
output: Optimal action $L_i \in 2^{E_i^*}$

1 Compute $\overline{c}^*(z)$ by applying Algorithm 8
2 **while** $d(z) > 1$ **do**
  Remove $\lceil d(z)/2 \rceil$ *outgoing actions at z but leave at least one action*
  Recompute the optimal cost, say $\overline{c}^*(z)'$, *for the reduced AIS*
  if $\overline{c}^*(z)' = \overline{c}^*(z)$ **then**
    The optimal action is one of the remaining actions at z
  else
    The optimal strategy is one of the removed actions at z
3 **if** *d(z) = 1* **then**
  Return the only one action

Example 7. *We calculate the optimal action for state* 3 *in Figure 8 by following Algorithm 9. In step 1, we compute* $\overline{c}^*(3) = 1$, *as written in blue next to state* 3. *In step 2, we choose to remove edge* $3 \to 6$ *and recompute the optimal cost using the AIS without the removed edge. The resulting new optimal cost is* 1, *which is the same as the original optimal cost. Therefore, we know that the optimal action is the only remaining edge* $3 \to 7$.

Once we find an optimal insertion action for every Z state using Algorithm 9, we construct optimal insertion strategy H in Algorithm 10 that contains all the actions at Y states and only the actions selected in the optimal strategy at Z states. The resulting H is a subgraph of the AIS and it will be used to build the optimal insertion automaton using Algorithm 7.

Theorem 5. *Applying Algorithms 8, 9, 10 and 7 solves Problem 2.*

Algorithm 10: Find an optimal mean-cost insertion strategy

input : AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and weight function w

output: Optimal strategy $H = (Y_H \cup Z_H, E_o \cup 2^{E_i^*}, f_H, y_{H,0})$

1 $y_{H,0} := y_0$, $Y_H := \{y_{H,0}\}$

2 **for** $u \in Y_H$ *that has not been examined* **do**

  for $e \in E_o$ **do**

    $f_H(u,e) := f_{AIS,yz}(u,e)$ if $f_{AIS,yz}(u,e)$ is defined

3 **for** $u \in Z_H$ *that has not been examined* **do**

  $f_H(u,e) := f_{AIS,yz}(u,e)$ where e is the optimal action for u computed using Algorithm 9

4 Go back to step 2 until all selected states have been examined

Proof: Algorithms 8 and 9 follow the results in [24]. In Algorithm 10, all system actions are chosen and all insertion actions are optimized. Thus, the strategy is i-enforcing and optimal. Finally, the IA we obtain in Algorithm 7 is optimal because it is constructed from the strategy in Algorithm 10. ∎

In all, given the AIS$= (Y \cup Z, E_o \cup 2^{E_i^*}, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$, computing the optimal mean-cost insertion function can be done in $O(n^4|f_{AIS}|\log(\frac{|f_{AIS}|}{n})W_{max})$, where n is the number of states in the AIS and $|f_{AIS}|$ is the number of transitions in $f_{AIS} = f_{AIS,yz} \cup f_{AIS,zy}$. In terms of the size of the state estimator of the system, the complexity is $O(|E_o|^5|X_{\mathscr{E}}|^{12}\log(|X_{\mathscr{E}}|)W_{max})$.

Example 8. *We have computed in Example 7 the optimal action for state* 3*. In this example, we complete the optimal strategy using Algorithm 10 and build the optimal insertion automaton using Algorithm 7. In Algorithm 10, all insertion states other than state* 3 *in the weighted graph have degree* 1*. Thus, only edge* $3 \to 6$ *for state* 3 *needs to be removed. The resulting optimal state-based strategy H is the automaton without the shaded states in Figure 8. After H is obtained, we then follow Algorithm 7 to build the optimal IA from H, as it was done in Example 5. The resulting optimal insertion automaton is shown in Figure 10.*

Remark 1. *When finding the optimal action in Algorithm 9, there may be other actions that are as good as the selected one. As a consequence, there may be more than one solution to Problem 2(b). Our algorithmic procedure returns one of them.*

D. Optimal Insertion Function for the Motivating Example

Let us now go back to the motivating example in Section III-C and compute the optimal insertion function that inserts fictitious queries by using the techniques developed in this paper. The constructed AIS has 84 states and thus there exists an i-enforcing insertion function. To synthesize an optimal insertion function, we assign the same unit cost to each inserted query, and compute $V_l(0) = 28220$ using Equation (2), where $l = n^2 W_{max} = 84^2 \cdot 4 = 28224$. Because $V_l(0) \geq nW_{max} = 336$, we conclude from Theorem 1 that there is no optimal total-cost insertion function. Hence, we solve for an optimal mean-cost insertion function by applying Algorithms 8, 9, 10 and 7. The optimal maximum mean cost is found to be 2 and the resulting insertion automaton is shown in Figure 11. The insertion function encoded in Figure 11 will provably guarantee that visiting location 6 is never revealed.

Let us look at how this insertion automaton modifies the problematic query sequence cdd. The insertion automaton modifies cdd to cdc_ic_id by inserting c_ic_i, which induces the intruder to generate estimate $\{4,7\}$. It is worth noticing that, for this particular query sequence, the intruder's new inference does not even include the true actual location 6. However, this is not generally true.

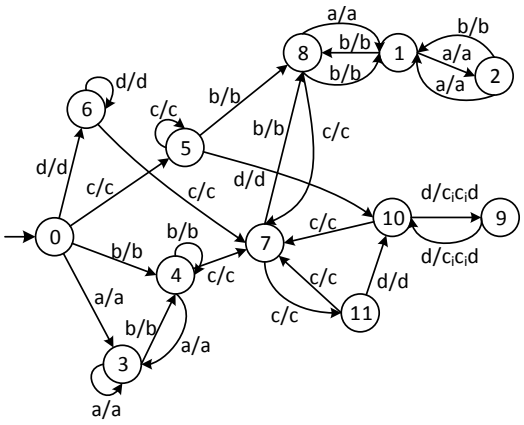


Fig. 11. The optimal insertion automaton.

VIII. Conclusion

We have considered the problem of synthesizing an optimal insertion function for enforcing opacity. The All Insertion Structure (AIS), which embeds all insertion functions, is used as a structure over which the optimization is performed. First, we proposed a more efficient algorithm for constructing a more compact AIS than presented in prior work. To quantify insertion functions, two costs were considered: the *maximum total cost* and the *maximum mean cost* For each cost, we presented an algorithm that computes the cost value of a given insertion function. We also presented algorithms that synthesize an optimal insertion function, for each cost. Specifically, we first minimize the maximum total cost and determine if an optimal total-cost insertion function exists. If such an optimal one exists, we synthesize an optimal total-cost insertion function. Otherwise, we synthesize an optimal mean-cost insertion function. The synthesis algorithms presented in this paper were developed by adapting and customizing results in game theory for minimax games and mean payoff games on weighted automata. We have also shown how to encode the resulting optimal insertion function as an I/O automaton. Finally, we have presented a case study that applies the insertion mechanism to location privacy in location-based services.

IX. Acknowledgment

The authors wish to acknowledge the anonymous reviewers for their detailed comments that helped to improve the presentation of the main results in this paper.

References

[1] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat. *Synchronization and linearity*, volume 3. Wiley New York, 1992.

[2] M. Ben-Kalefa and F. Lin. Opaque superlanguages and sublanguages in discrete event systems. In *Proc. of the 48th IEEE Conference on Decision and Control*, pages 199–204. IEEE, 2009.

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

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

[5] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems, 2nd Edition*. Springer, 2008.

[6] F. Cassez, J. Dubreil, and H. Marchand. Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, pages 1–28, 2012.

[7] F. Cassez and S. Tripakis. Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88(4):497–540, 2008.
[8] J. Dubreil, P. Darondeau, and H. Marchand. Supervisory control for opacity. *IEEE Transactions on Automatic Control*, 55(5):1089–1100, 2010.
[9] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.
[10] Y. Falcone and H. Marchand. Runtime enforcement of K-step opacity. In *Proc. of the 52nd IEEE Conference on Decision and Control*, 2013.
[11] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of the 1st International Conference on Mobile Systems, Applications and Services*, pages 31–42, 2003.
[12] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.
[13] F. Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011.
[14] L. Mazaré. Using unification for opacity properties. *Proc. of the 4th IFIP WG1*, 7:165–176, 2003.
[15] A. Saboori and C. N. Hadjicostis. Notions of security and opacity in discrete event systems. *Proc. of the 46th IEEE conference on Decision and Control*, pages 5056–5061, Dec 2007.
[16] A. Saboori and C. N. Hadjicostis. Verification of initial-state opacity in security applications of DES. *Proc. of the 9th International Workshop on Discrete Event Systems*, pages 328–333, May 2008.
[17] A. Saboori and C. N. Hadjicostis. Opacity-enforcing supervisory strategies via state estimator constructions. *IEEE Transactions on Automatic Control*, 57(5):1155–1165, 2012.
[18] K. G. Shin, X. Ju, Z. Chen, and X. Hu. Privacy protection for users of location-based services. *Wireless Communications, IEEE*, 19(1):30–39, 2012.
[19] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
[20] Y.-C. Wu. *Verification and Enforcement of Notions of Opacity Security Properties in Discrete Event Systems*. PhD thesis, University of Michigan, Ann Arbor, August 2014.
[21] Y.-C. Wu and S. Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems: Theory and Aplications*, 23(3):307–339, 2013.
[22] Y.-C. Wu and S. Lafortune. Synthesis of insertion functions for enforcement of opacity security properties. *Automatica*, 50(5):1336–1348, 2014.
[23] Y.-C. Wu, K. A. Sankararaman, and S. Lafortune. Ensuring privacy in location-based services: An approach based on opacity enforcement. *Proc. of the 14th International Workshop of Discrete Event Systems*, pages 33–38, 2014.
[24] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.

Yi-Chin Wu is a postdoctoral researcher at the TerraSwarm Research Center, affiliated with University of Michigan and University of California, Berkeley. She received her B.S. degree from National Taiwan University, Taipei, Taiwan, in 2008, and her PhD degree from the University of Michigan, Ann Arbor, in 2014, all in Electrical Engineering. Her research interests include modeling, verification, and enforcement of privacy properties in Discrete Event Systems, and their applications to cyber and cyber-physical 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. Dr. Lafortune is a Fellow of the IEEE (1999). 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). Dr. 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 is the lead developer of the software package UMDES and co-developer of DESUMA with L. Ricker. He co-authored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems - Second Edition* (Springer, 2008). Dr. Lafortune is Editor-in-Chief of the Journal of Discrete Event Dynamic Systems: Theory and Applications.