Synthesis of Opacity-Enforcing Insertion Functions That Can Be Publicly Known

Yi-Chin Wu and Stéphane Lafortune

***Abstract*— Our prior work has studied the enforcement of opacity, a security property, using insertion functions that insert fictitious events at the output of the system, thereby preventing an intruder from inferring the given system's secret. The insertion functions previously considered enforce opacity under the assumption that the intruder does not know about the implementation of the insertion function. In this paper, we relax that assumption and consider a stronger class of insertion functions that enforce opacity whether or not the intruder knows the insertion function. This property is formally characterized as *public-and-private enforceability*, or *PP-enforceability* for short. A PP-enforcing insertion function is guaranteed to output only behaviors consistent with the non-secret behaviors of the system and thus it enforces opacity when the intruder has no knowledge of the insertion function (private case). Moreover, a PP-enforcing insertion function guarantees that the intruder can never infer the occurrence of the secret, even when the intruder knows the exact implementation of the insertion function (public case). We characterize the property of PP-enforceability and present an algorithm that provably synthesizes a PP-enforcing insertion function.**

I. Introduction

Opacity is an emerging privacy property that characterizes whether the secret of the system can be inferred by an outside observer termed the *intruder*. The notion of opacity was first introduced in the computer science community [10] to analyze cryptographic protocols. It later became an active topic in the Discrete Event Systems (DES) community as DES theory provides suitable models and analytical techniques for formulating and studying opacity properties [3], [4], [11].

In this paper, we consider opacity in systems modeled as finite-state automata. Specifically, the system is a partially observable and/or nondeterministic finite-state automaton. It has a *secret* that needs to be hidden. The intruder is an outside observer that knows the system structure and tries to infer the occurrence of the secret by passively observing the output from the system. The system is said to be opaque if for every behavior induced by the secret (termed *secret behavior*), there is another observationally equivalent behavior that is not induced by the secret (termed *non-secret behavior*). The intruder, by observing the output from the system, is never sure if the system is outputting the secret or the non-secret behavior.

Many prior works have considered opacity with various representations of the secret, such as states and languages. These secret representations lead to various opacity notions; e.g., initial-state opacity, current-state opacity, language-based opacity, K-step opacity, and initial-and-final state opacity [6], [9], [12], [15]. When a given opacity notion is violated, researchers have proposed various methods to enforce opacity. One popular approach is to design a minimally restrictive supervisor, which disables behaviors that violate opacity [6], [7], [13]. The work in [2] adopted a similar approach but focused on how to minimally restrict the secret behavior or minimally enlarge the non-secret behavior in order to enforce opacity. In [8], the authors considered a delay mechanism, which enforces K-step opacity by delaying outputting events from the system until the secret expires. Finally, our prior work [16] developed an insertion mechanism that enforces opacity by inserting fictitious events at the output, without interfering with the system.

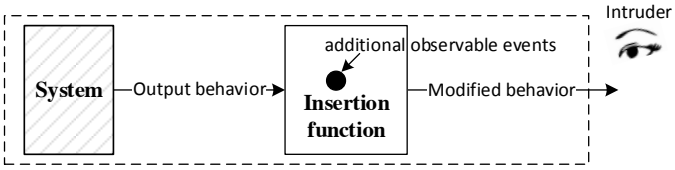


Fig. 1. The insertion mechanism

This paper considers the same insertion mechanism as in [16]. As shown in Figure 1, an insertion function is an interface that inserts fictitious events to the output behavior of the system. In [16], we assumed that the insertion function used by the system is always kept private. With this assumption, we designed insertion functions that only output strings consistent with the non-secret behaviors of the system and thus induce the intruder to think that the secret of the system never occurs. Here, we relax this assumption. While the implementation of the insertion function is supposed to be kept private, a sophisticated intruder may learn the full set of modified behaviors output from the insertion function, compare it with the system model, and reconstruct the correct insertion function. Also, if the intruder knows the system's design optimality criteria, it may follow the optimal synthesis algorithm in [14] and construct the correct insertion function. Hence, we need an insertion function that enforces opacity even when its implementation becomes known.

This work was partially supported by NSF grants CCF-1138860 (Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering) and CNS-1421122, 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.

Y.-C. Wu is with the Department of EECS at the University of Michigan and the Department of EECS at the University of California at Berkeley. S. Lafortune is with the Department of EECS at the University of Michigan. {ycwu;stephane}@umich.edu

To enforce opacity regardless whether or not the intruder knows the implementation of the insertion function, we formally characterize a property called *public-and-private enforceability*, or *PP-enforceability* for short. A PP-enforcing insertion function is guaranteed to enforce opacity when the insertion function is kept private *and* when it becomes known to the intruder. In the former case, the insertion function outputs only behaviors consistent with the non-secret behaviors of the system. In the latter case, the insertion function is designed such that for every secret behavior of the system, there is a non-secret behavior of the system that has the same modified output from the insertion function.

The main contributions of this paper are two-fold. First, we formally characterize the property of PP-enforceability and discuss opacity enforcement in both the private and the public cases. Second, we present an algorithmic procedure that provably synthesizes a PP-enforcing insertion function.

The remaining sections of this paper are organized as follows. Section II introduces the system model and the notion of opacity. Section III formally introduces insertion functions and defines the notion of *public-and-private enforceability*. In Section IV, we present the main result of this paper. We first establish a sufficient condition for a given insertion function to be PP-enforcing, and then presents the INPRIVALIC Algorithm, which synthesizes PP-enforcing insertion functions. A running example that illustrates the INPRIVALIC Algorithm is given. Finally, Section V concludes the paper.

II. Opacity Notions in Automata Models

A. Automata Models

We consider opacity problems in DES modeled as (potentially nondeterministic) automata. An automaton $G = (X, E, f, X_0)$ has a finite set of states, a finite 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 set of system behaviors that is defined by $\mathscr{L}(G, X_0) := \{t \in E^* : (\exists x \in X_0)[f(x,t) \text{ is defined}]\}$. We will write $\mathscr{L}(G)$ if X_0 is clear from the context. 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 $e \in E$. The 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

The settings of an opacity problem in DES are: (1) the system is a partially observable and/or nondeterministic finite-state automaton G; (2) G has a *secret*; (3) the intruder is an observer of G that knows the structure of G. Hence, the intruder, with the knowledge of G, can construct estimates and infer the system's real behavior using online observations. Opacity holds if no intruder's estimate asserts that the real behavior is induced by the secret (termed *secret behavior*). The system is opaque if for every *secret behavior*, there is another observationally equivalent behavior that is not induced by the secret (termed *non-secret behavior*).

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). Because these four notions can be mapped to one and another [15], we derive our results in this paper using only the notion of current-state opacity.

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 $\forall t \in \mathscr{L}_S(G) := \{t \in \mathscr{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap X_S \neq \emptyset\}$, $\exists t' \in \mathscr{L}_{NS}(G) := \{t \in \mathscr{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap (X \setminus X_S) \neq \emptyset\}$ such that $P(t) = P(t')$.

We will write L_S for $\mathscr{L}_S(G)$ and L_{NS} for $\mathscr{L}_{NS}(G)$ when G is obvious from the context.

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 [12]. For simplicity, we will call a forward state estimator an *estimator* and denote it by $\mathscr{E}$ hereafter.

III. Insertion Mechanism for Opacity Enforcement

To enforce opacity, our prior work proposed an insertion mechanism, which inserts fictitious events to the output of the system. As shown in Figure 1, the insertion function is an interface between the system and outside observers. It receives the system's original output behavior, inserts fictitious events if necessary, and outputs the modified output. Depending on the intruder's knowledge, we design insertion functions according to different specifications. In this section, we introduce and discuss two specifications: *private enforceability* and *public enforceability*. The remainder of this paper will focus on the stronger specification - public enforceability.

A. Insertion Functions and Insertion Automata

Before formally defining an insertion function, we introduce a new event set E_i that contains all the events used by the insertion function. Each event in E_i is an observable event with a *virtual* insertion label, i.e., $E_i := \{e_i : e \in E_o\}$. Notice that from the viewpoint of the intruder, inserted and genuine observable events are indistinguishable. But we introduce set E_i in order to easily distinguish the two types of events in the discussion.

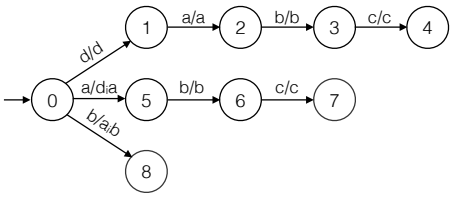


Fig. 2. Insertion automaton *IA*

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 observable string $se_o \in P[\mathscr{L}(G)]$, an insertion function is defined such that $f_I(s,e_o) = s_I e_o$ when string $s_I \in E_i^*$ is inserted before e_o. We also define a *string-based* insertion function f_I^{str} recursively from f_I: $f_I^{str}(\varepsilon) = \varepsilon$ and $f_I^{str}(se_o) = f_I^{str}(s) f_I(s,e_o)$. Given G, the modified language output by the insertion function is $f_I^{str}(P[\mathscr{L}(G)]) = \{\tilde{s} \in (E_i^* E_o)^* : \exists s \in P[\mathscr{L}(G)], f_I^{str}(s) = \tilde{s}\}$.

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 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. Figure 2 shows an insertion automaton that encodes the insertion function f_I defined as: $f_I(\varepsilon,a) = d_i a, f_I(\varepsilon,b) = a_i b$, and $f_I(s,e_o) = e_o$ for $se_o \in P[\mathscr{L}(G)] \setminus \{a,b\}$.

Finally, we define mask $\mathscr{M}_i$ that models the observation of the intruder. Because inserted and genuine observable events are indistinguishable from the viewpoint of the intruder, $\mathscr{M}_i : E_o \cup E_i \to E_o$ is a mask that removes the insertion labels. That is, $\mathscr{M}_i(e) = e$ if $e \in E_o$ and $\mathscr{M}_i(e_i) = e$ if $e_i \in E_i$. Mask $\mathscr{M}_i$ extends to domain $(E_o \cup E_i)^*$ in the usual manner.

B. Private Enforceability

In [16], we characterized the specification of *private enforceability* that insertion functions need to satisfy. Specifically, private enforceability is the combination of two properties: admissibility and private safety.[1] Admissibility is an input property for insertion functions; it requires insertion functions to be defined for all $P[\mathscr{L}(G)]$. This property is required when the system needs to execute its full behaviors or when the system cannot be interfered with. For example, in applications where users query servers, we do not want to exclude any query from a given user (here, the user's behavior defines the system's behavior).

Definition 2 (Admissibility): Consider G, P, L_S and L_{NS}. An insertion function f_I is admissible if: $\forall se_o \in P[\mathscr{L}(G)]$, where $s \in E_o^*, e_o \in E_o$, $\exists s_I \in E_i^*$ s.t. $f_I(s,e_o) = s_I e_o$.
Since no behavior in $P[\mathscr{L}(G)]$ would be excluded by admissible insertion functions, the subset relationship is preserved under admissible insertion functions. Hence, the following proposition holds.

Proposition 1: Consider insertion function f_I that is admissible with respect to $P[\mathscr{L}(G)]$. If $L_1 \subseteq L_2 \subseteq P[\mathscr{L}(G)]$, then $\mathscr{M}_i[f_I^{str}(L_1)] \subseteq \mathscr{M}_i[f_I^{str}(L_2)]$.

On the other hand, private safety is an output property for insertion functions. We term this property "private" safety because it is under the assumption that the intruder has no knowledge of the insertion function at the outset. Consequently, the intruder is expecting to observe behaviors that are consistent with the system structure. Notice that we consider insertion functions that are used to enforce opacity *online*. Hence, every modified output behavior from the insertion function should *always* be consistent with an original non-secret behavior from the system. Because of this "always" requirement, every modified output behavior should be observationally equivalent, under $\mathscr{M}_i$, to a string in the safe language L_{safe}, which is the supremal prefix-closed sublanguage of $P(L_{NS})$ defined as $L_{safe} = P[\mathscr{L}(G)] \setminus (P[\mathscr{L}(G)] \setminus P(L_{NS}))E_o^*$. Hereafter, we call a string $s \in P[\mathscr{L}(G)]$ *safe* if it is in L_{safe} and *unsafe* otherwise.

Definition 3 (Private Safety): Consider G, P, L_S and L_{NS}. An insertion function f_I is privately safe if $\forall s \in P[\mathscr{L}(G)]$, $\mathscr{M}_i[f_I^{str}(s)] \in L_{safe}$; equivalently, $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] \subseteq L_{safe}$.

Finally, when an insertion function is both admissible and privately safe, we say that it is privately enforcing.

C. Private-and-Public Enforceability (PP-Enforceability)

Privately enforcing insertion functions enforce opacity in a way that the intruder would never observe unsafe behaviors of the system. A naive intruder, with no knowledge of the insertion function at the outset, would believe that the secret has never occurred and would not suspect the existence of an insertion function. However, a drawback is that such an insertion function may fail when the intruder knows the implementation of the insertion function. Below we show such an example.

Example 1: Consider the current-state estimator in Figure 3 where states 7 and 8 contain only secret states. These estimator states represent sets of system states; they are numbered from 0 to 8 for simplicity. Suppose that opacity is enforced by the privately enforcing insertion function shown in Figure 2. If the intruder has no knowledge of f_I, it would never conclude that the secret occurs, as the output from f_I is always safe under $\mathscr{M}_i$ (notice that $L_{safe} = \overline{dabc} + \overline{ab}$). However, if the intruder knows the implementation of f_I, then it would be able to conclude a state estimate of 8 when it observes ab. This is because if ab were the genuine output behavior from the system, it would have been modified to $d_i ab$. The only genuine system's output that would produce ab under mask $\mathscr{M}_i$ is b.

Example 1 shows how an intruder can infer the secret if it knows the implementation of the insertion function. Indeed, there are ways for intruders to learn the implementation of the insertion function. For example, the intruder could use learning algorithms such as in [1] to learn the modified system $\tilde{G}$, which is the input parallel composition of G and

[1]Private enforceability was termed i-enforceability and private safety was termed simply safety in [16]. We adopt the new terminology in order to distinguish the private and the public cases.

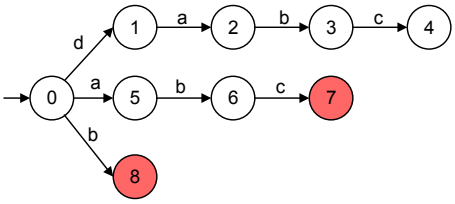


Fig. 3. Current-state estimator $\mathscr{E}$; states 7 and 8 contain only secret states

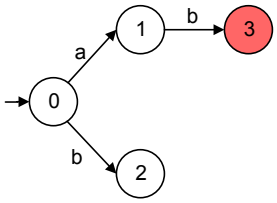


Fig. 4. Current-state estimator $\mathscr{E}$ where state 3 reveals the secret

insertion automaton IA,[2] and may use $\tilde{G}$ and G to infer the correct implementation of IA. Also, if the intruder knows the system's design optimality criteria, it could follow the optimal synthesis algorithm in [14] and may construct the correct insertion function. In either case, we wish to use an insertion function that still enforces opacity when its implementation becomes known.

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

Definition 4 (Public Safety): Consider G, P, L_S and L_{NS}. An insertion function f_I is publicly safe if $\forall \tilde{s} \in f_I^{str}(P[\mathscr{L}(G)]), \exists t \in L_{NS}$ s.t. $\mathscr{M}_i(f_I^{str}[P(t)]) = \mathscr{M}_i(\tilde{s})$; equivalently, $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] \subseteq \mathscr{M}_i(f_I^{str}[P(L_{NS})])$.
We can treat an insertion function as a general "projection" that projects a given string $s \in E_o^*$ to $\tilde{s} \in (E_i^* E_o)^*$. With this consideration, the public safety property becomes an opacity property for G, under projection $\mathscr{M}_i \circ f_I^{str} \circ P$.

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

Definition 5 (PP-Enforceability): Insertion function f_I is PP-enforcing if it is admissible, privately safe and publicly safe.

Example 2: In Example 1, insertion function f_I is privately enforcing but not PP-enforcing. Specifically, for $\tilde{s} = a_i b$, there is no $t \in L_{NS}$ for which $\mathscr{M}_i(f_I^{str}[P(t)]) = ab$. In this example, we define another insertion function: $f_I'(\varepsilon, a) = d_i a, f_I'(\varepsilon, b) = d_i a_i b$, and $f_I'(s, e_o) = e_o, \forall s e_o \in P[\mathscr{L}(G)] \setminus \{a, b\}$. One can verify that f_I' is PP-enforcing. Specifically, f_I' is admissible because it is defined for every $P[\mathscr{L}(G)]$; it is privately safe as $\mathscr{M}_i[f_I'(P[\mathscr{L}(G)])] = \overline{dabc} \subseteq L_{safe}$; also, f_I' is publicly safe since for every $\tilde{s} \in \overline{dabc}$, there is $t \in L_{NS}$ that is observationally equivalent and with no inserted event.

Based on the definitions of private enforceability and public enforceability, we discover a fact that seems surprising at the first look. Intuitively, knowing more about the system's implementation seems to always help in determining the true behavior of the system. We may be tempted to think that if an insertion function is known to the intruder, it should enforce opacity when the intruder does not actually know the insertion function's implementation. However, publicly enforcing insertion functions can fail to enforce opacity when they are not known, i.e., they can fail to be privately enforcing. Below we show such an example.

Example 3: Consider the current-state estimator in Figure 4 where state 3 contains only secret states. We define an insertion function as: $f_I(\varepsilon, b) = a_i b$ and $f_I(s, e_o) = e_o, \forall s e_o \in \overline{ab}$. This insertion function is publicly enforcing since it is admissible and the only unsafe behavior ab is now observationally equivalent to safe behavior b. However, if the intruder does not know the implementation of f_I, it would always believe that the secret has occurred. Hence, the secret will be revealed when the system indeed outputs ab.

This example explains our choice of using PP-enforceability as our specification for insertion functions. We could neither assume the intruder to know the complete implementation of the insertion function nor assume the intruder not to learn the implementation. Thus, insertion functions should enforce opacity regardless of whether the intruder knows the implementation or not.

IV. Synthesis of PP-Enforcing Insertion Functions

Our prior work has constructed the All-Insertion Structure (AIS) that embeds all privately enforcing insertion functions [14], [16]. In this section, we will rely on the structure of the AIS and synthesize an insertion function that is PP-enforcing. Specifically, this section starts with establishing a sufficient condition for privately enforcing insertion functions to be PP-enforcing. Based on this sufficient condition, we build in Section IV-B a substructure of the AIS that embeds only PP-enforcing insertion functions and then use the resulting substructure to synthesize a PP-enforcing insertion.

A. Sufficient Condition for PP-Enforcing Insertion Functions

This section will characterize a sufficient condition for privately enforcing insertion functions to be PP-enforcing. We first derive in Proposition 2 a necessary and sufficient condition for an admissible insertion function to be PP-enforcing, based on the definitions in Section III.

Proposition 2: An admissible insertion function f_I is PP-enforcing if and only if:
(i) $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] = \mathscr{M}_i(f_I^{str}[P(L_{NS})])$
(ii) $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] \subseteq L_{safe}$.

[2] We refer the interested readers to [16] for the formal definition of input parallel composition.

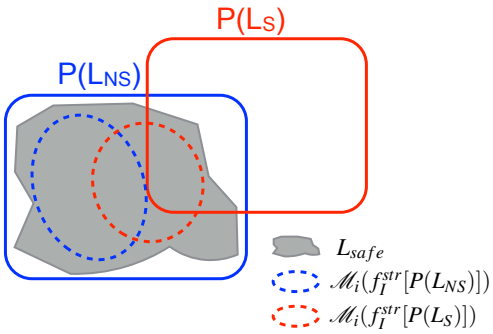


Fig. 5. Venn Diagram that shows the relationship between $P(L_S), P(L_{NS}), \mathscr{M}_i(f_I^{str}[P(L_S)]), \mathscr{M}_i(f_I^{str}[P(L_{NS})])$, and L_{safe} for a given privately enforcing f_I

Proof: In (i), $\mathscr{M}_i(f_I^{str}[P(L_{NS})]) \subseteq \mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])]$ holds because of Proposition 1; also, $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] \subseteq \mathscr{M}_i(f_I^{str}[P(L_{NS})])$ by the definition of public safety. If f_I is privately-safe, (ii) also holds by definition. ■

To better understand how a privately enforcing f_I fails to be PP-enforcing, we also draw in Figure 5 the Venn diagram of the secret and the non-secret languages before and after being modified by a privately enforcing f_I. As shown in the figure, f_I maps all strings in $P[\mathscr{L}(G)]$ to L_{safe}. In general, $\mathscr{M}_i(f_I^{str}[P(L_S)])$ may not be a subset of $\mathscr{M}_i(f_I^{str}[P(L_{NS})])$. In this case, the intruder, when knowing the implementation of f_I, could determine the occurrence of the secret when it observes strings in $\mathscr{M}_i(f_I^{str}[P(L_S)]) \setminus \mathscr{M}_i(f_I^{str}[P(L_{NS})])$. If, on the other hand, $\mathscr{M}_i(f_I^{str}[P(L_S)])$ is contained in $\mathscr{M}_i(f_I^{str}[P(L_{NS})])$, then $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] = \mathscr{M}_i(f_I^{str}[P(L_{NS})])$ and thus f_I is PP-enforcing. A special case where $\mathscr{M}_i(f_I^{str}[P(L_S)])$ is guaranteed to be contained in $\mathscr{M}_i(f_I^{str}[P(L_{NS})])$ is when $\mathscr{M}_i(f_I^{str}[P(L_{NS})])$ is the entire set of L_{safe}. Based on this special case, Lemma 1 and Theorem 1 below show sufficient conditions for a privately enforcing f_I to be PP-enforcing.

Lemma 1: Consider privately enforcing insertion function f_I. If $\mathscr{M}_i[f_I^{str}(L_{safe})] = L_{safe}$, then f_I is also publicly enforcing; that is, f_I is PP-enforcing.

Proof: Because a privately enforcing insertion function f_I is admissible, we can prove this Lemma using Proposition 2. We will show that items (i) and (ii) hold if $\mathscr{M}_i[f_I^{str}(L_{safe})] = L_{safe}$. First, item (ii) holds as f_I is privately safe. We then show that item (i) holds to complete the proof. Since f_I is admissible and $L_{safe} \subseteq L_{NS}$, by Proposition 1, $\mathscr{M}_i[f_I^{str}(L_{safe})] \subseteq \mathscr{M}_i(f_I^{str}[P(L_{NS})])$. If $\mathscr{M}_i[f_I^{str}(L_{safe})] = L_{safe}$, then $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] \subseteq L_{safe} = \mathscr{M}_i[f_I^{str}(L_{safe})] \subseteq \mathscr{M}_i(f_I^{str}[P(L_{NS})])$. Because $P(L_{NS}) \subseteq P[\mathscr{L}(G)]$ and f_I is admissible, the other direction of the set inclusion $\mathscr{M}_i(f_I^{str}[P(L_{NS})]) \subseteq \mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])]$ holds. ■

We now replace L_{safe} with a subset $L \subseteq L_{safe}$ and follow the same argument in the proof for Lemma 1 to derive a more general sufficient condition in Theorem 1.

Theorem 1: Consider privately enforcing insertion function f_I. If there is $L \subseteq L_{safe}$ such that $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] = L$ and $\mathscr{M}_i[f_I^{str}(L)] = L$, then f_I is also publicly enforcing; i.e., f_I is PP-enforcing.

Notice that the condition in Theorem 1 is sufficient but not necessary. That is, there exists a PP-enforcing insertion function f_I such that no $L \subseteq L_{safe}$ satisfies $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] = L$ and $\mathscr{M}_i[f_I^{str}(L)] = L$. Below we show such an f_I.

Example 4: Consider automaton G with observable events $E_o = \{a,b,c,d\}$. The observable behavior is $P[\mathscr{L}(G)] = \{\overline{abc}, \overline{dabc}, b, c\}$, where $L_{safe} = \{\overline{abc}, \overline{dabc}, c\}$. Define f_I so that $f_I(\varepsilon, a) = d_i, f_I(\varepsilon, b) = a_i, f_I(\varepsilon, c) = a_i b_i$ and $f_I(s, e_o) = e_o$ otherwise. Because $\mathscr{M}_i(f_I^{str}[\mathscr{L}(G)]) = \{\overline{abc}, \overline{dabc}\} \subseteq L_{safe}$, f_I is privately enforcing. One can also check that f_I is publicly enforcing. However, the only set $L \subseteq L_{safe}$ satisfying $\mathscr{M}_i[f_I^{str}(L)] = L$ is $\{\overline{dabc}\}$, which is not equal to $\mathscr{M}_i(f_I^{str}[\mathscr{L}(G)]) = \{\overline{abc}, \overline{dabc}\}$. Hence, f_I is a PP-enforcing insertion function f_I such that no $L \subseteq L_{safe}$ satisfies $\mathscr{M}_i(f_I^{str}[\mathscr{L}(G)]) = L$ and $\mathscr{M}_i[f_I^{str}(L)] = L$. The condition in Theorem 1 is not necessary.

B. The INPRIVALIC Algorithm

Based on Theorem 1, we now develop an algorithm that synthesizes a PP-enforcing insertion function. Recall the AIS built in [16] that embeds all privately enforcing insertion functions. The INPRIVALIC Algorithm uses the AIS to identify language $L_{pp} \subseteq L_{safe}$ that can be output with no insertion, and then builds an insertion structure IS_{pp} that embeds insertion functions that never insert events into strings in L_{pp} and modifies all other strings to L_{pp}. As a result, insertion functions synthesized from IS_{pp} are guaranteed to be PP-enforcing by Theorem 1. IS_{pp} embeds only PP-enforcing insertion functions that *greedily* output the original strings if possible. Because this algorithm synthesizes INsertion functions with PRIVAte-and-pubLIC-enforceability property, we call it the INPRIVALIC Algorithm. Algorithm 1 presents the 9 steps of the INPRIVALIC Algorithm.

Algorithm 1: INPRIVALIC Algroithm

input : $G = (X, E, f, X_0)$, P, and $X_S \subseteq X$
output: A PP-enforcing insertion automaton

1 $(\mathscr{E}^d, \mathscr{E}^f, \mathscr{E})$ = CONSTRUCT-ESTIMATORS(G, P, X_S)
2 $V = \mathscr{E}^d ||_d \mathscr{E}^f$
3 V_u = UNFOLD-VERIFIER$(V, \mathscr{E})$
4 AIS = PRUNE-INADMISSIBLE(V_u)
5 $\mathscr{E}^d_{pp}$ = BUILD-L_{pp}(AIS, $\mathscr{E}^d$)
6 $V_{pp} = \mathscr{E}^d_{pp} ||_d \mathscr{E}^f$
7 $V_{u,pp}$ = UNFOLD-PP-VERIFIER$(V_{pp}, \mathscr{E}, \mathscr{E}^d_{pp})$
8 IS_{pp} = PRUNE-INADMISSIBLE$(V_{u,pp})$
9 return SYNTHESIZE(IS_{pp})

Steps 1 to 4 construct the AIS. These steps are improved from those in [16] and the same as in [14]. We recall them below to make this paper self-contained. Specifically, in step 1, we build from the system's state estimator $\mathscr{E}$ two special estimators: $\mathscr{E}^d$ that recognizes language L_{safe} and $\mathscr{E}^f$ that includes all possible insertions. Then, in step 2, we compose the two estimators to find all the insertions that modify $P[\mathscr{L}(G)]$ to L_{safe}. Step 3 unfolds the insertions identified in step 2 with respect to every system's output event, in a

two-player game structure. Finally, in step 4, we prune away insertions that are not admissible; the resulting AIS embeds all privately enforcing insertion functions.

With the AIS being built, steps 5 to 8 then build IS_{pp}, which embeds "greedy" PP-enforcing insertion functions that output the original strings if possible. In step 5, we construct $\mathscr{E}^d_{pp}$ that recognizes L_{pp}, which is a sublanguage of L_{safe} that can be output without insertion while not affecting private enforceability. With $\mathscr{E}^d_{pp}$, steps 6 to 8 follow similar procedures to steps 2 to 4. Specifically, step 6, like step 2, composes $\mathscr{E}^d_{pp}$ and $\mathscr{E}^f$ to identify insertions that modify $P[\mathscr{L}(G)]$ to L_{pp}. Step 7 unfolds the insertions in step 6 but in a way that never inserts events into strings in L_{pp}. Then, step 8 prunes away inadmissible insertions. The structure built in step 8 is denoted by IS_{pp} because it is an *Insertion Structure that embeds PP-enforcing insertion functions*. Finally, step 9 synthesizes an insertion automaton from IS_{pp}. Because IS_{pp} embeds only PP-enforcing insertion functions, the insertion automaton synthesized from the INPRIVALIC Algorithm is guaranteed to be PP-enforcing.

Procedures for each step are formally presented below. Because of the space limitation, we will not explain the details of these procedures but instead use examples to demonstrate the algorithms. We use a system that has the estimator in Figure 3 as our running example, and demonstrate all the steps of the INPRIVALIC Algorithm in Example 5.

Example 5: In this example, we illustrate the INPRIVALIC Algorithm by using the estimator $\mathscr{E}$ in Figure 3. In step 1, since $\mathscr{E}$ is already built, we only need to build $\mathscr{E}^d$ and $\mathscr{E}^f$ from $\mathscr{E}$. Specifically, we build $\mathscr{E}^d$ in Figure 6(a) by removing states 7 and 8 that contain only secret states; it recognizes language L_{safe}. Also, we obtain $\mathscr{E}^f$ in Figure 6(b) by adding self loops for a_i, b_i, c_i, d_i (represented as dashed transitions in the figure) at every state. In step 2, we *dashed parallel compose* $\mathscr{E}^d$ and $\mathscr{E}^f$ and obtain $V := \mathscr{E}^d || _d \mathscr{E}^f$ in Figure 6(c). Dashed parallel composition is a special composition that synchronizes solid transitions in $\mathscr{E}^d$ and all transitions in $\mathscr{E}^f$ when the two transitions are labeled by the same event, subject to the virtual insertion label. Hence, every string in V is a modified string with insertion that has the same observable projection to a string in L_{safe} (here, $L_{safe} = \overline{dabc} + \overline{ab}$) under mask $\mathscr{M}_i$. That is, V identifies privately safe insertions. In step 3, we "unfold" the privately safe insertions in V for every system output, and build a game structure denoted by V_u. As can be seen in Figure 6(d), there are two types of states in V_u: squared states where the system plays and elliptical states where the insertion function plays. These states are information states that contain sufficient information to enumerate actions available at those states. Here, the actions at the squared states are events to be output from the system; the actions at the elliptical states are privately safe insertion strings. The game starts with the initial state $(0,0)$ of V where the system plays; initially the system can output d, a or b. If the system outputs a, the game then reaches state $((0,0),a)$, where the insertion function plays. To determine what insertion actions can be made at state $((0,0),a)$, we search on the graph of V. Specifically, to output a from $(0,0)$, we can either go to $(1,0)$ and inserts d_i, or stay at $(0,0)$ and inserts ε.[3] In the former case, we reach state $(2,5)$ in V with $d_i a$; in the latter case, we reach state $(5,5)$ with a. In either case, the reached state is the system's information state for the next turn. After we complete constructing the game structure V_u in this manner, we then prune inadmissible insertions. Observe that, in this V_u, state $((6,6),c)$ is a blocking state for the insertion function. This means that the insertion function could not react when the system outputs c and thus is not admissible. Hence, in step 4, we prune away inadmissible insertions by iteratively pruning such blocking states. The resulting structure is the AIS, which is the structure in Figure 6(d) without the shaded states.

With the AIS being built, we now proceed to steps 5 to 9. Step 5 builds $\mathscr{E}^d_{pp}$ that identifies L_{pp} by breadth-first searching on the AIS. The search continues as long as the insertion function can choose not to insert. For the AIS in Figure 6(d), the search expands the top branch in the AIS, resulting in $L_{pp} = \overline{\{dabc\}}$ and $\mathscr{E}^d_{pp}$ in Figure 6(e). Now, in step 6, we dashed parallel compose again and obtain $V_{pp} := \mathscr{E}^d_{pp} || _d \mathscr{E}^f$ in Figure 6(f) to identify insertions that modify the system's output strings to L_{pp}. Step 7 is similar to step 3 that unfolds all the insertions in V_{pp} but in a way that does not insert to L_{pp}. This explains why all insertion actions in the top branch of $V_{u,pp}$ in Figure 6(g) are labeled by ε. Finally, we complete IS_{pp} by pruning inadmissible insertions, as was done in step 4. Here, no state in $V_{u,pp}$ needs to be pruned, IS_{pp} is the same as $V_{u,pp}$ in Figure 6(g). To synthesize a PP-enforcing insertion function, we choose all outgoing edges at squared states and *one* insertion string at elliptical states. The resulting insertion function f_I is encoded as the insertion automaton IA in Figure 6(h), which inserts $d_i a_i$ before b and d_i before a. This is the same PP-enforcing insertion function considered in Example 2.

Algorithm 2: CONSTRUCT-ESTIMATORS

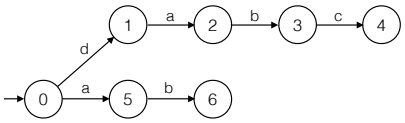
input : $G = (X, E, f, X_0)$, P, and $X_S \subseteq X$
output: $\mathscr{E}^d, \mathscr{E}^f, \mathscr{E}$

1 $\mathscr{E} := Obs(G) = (M, E_o, \delta, m_0)$ w.r.t P
2 Construct $\mathscr{E}^d$ by pruning every state $m \in M$ where $m \subseteq X_S$ and taking the accessible part
3 $\mathscr{E}^f := (M, E_o \cup E_i, \delta \cup \delta_{sl}, m_0)$ where $\delta_{sl}(m, e_i) = m, \forall m \in M, e_i \in E_i$
4 return $\mathscr{E}^d, \mathscr{E}^f, \mathscr{E}$

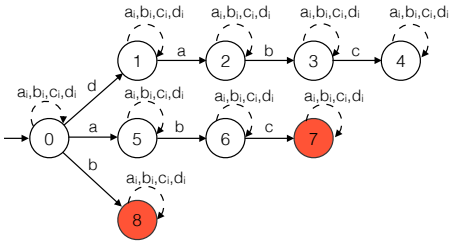
Theorem 2: Every insertion function f_I synthesized using the INPRIVALIC Algorithm is PP-enforcing.

Proof: Every f_I synthesized from IS_{pp} is privately enforcing because IS_{pp} embeds a subset of the insertion functions in the AIS. By construction, every f_I embedded in IS_{pp} does not insert to $L_{pp} \subseteq L_{safe}$ and modifies all other system's output strings to L_{pp}. That is, every f_I synthesized

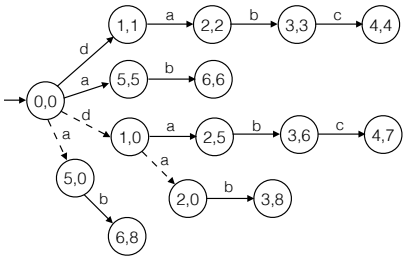
[3]Each insertion action is encoded using a state in V to represent a set of insertion strings. In this example, each insertion action represents only one insertion string. We refer the interested readers to [16] for examples and details about how we can use states in V to encode sets of insertion strings.



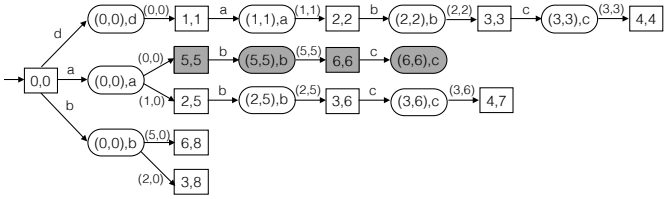
(a) Step 1: $\mathscr{E}^d$



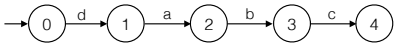
(b) Step 1: $\mathscr{E}^f$



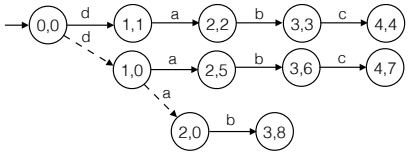
(c) Step 2: V



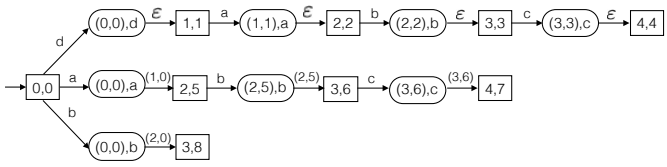
(d) Steps 3 and 4: V_u and the AIS (the structure without the shaded states)



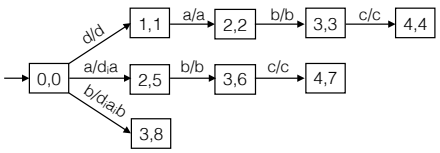
(e) Step 5: $\mathscr{E}^d_{pp}$



(f) Step 6: V_{pp}



(g) Steps 7 and 8: IS_{pp}



(h) Step 9: PP-enforcing insertion function

Fig. 6. The INPRIVALIC Algorithm on the running example

Algorithm 3: UNFOLD-VERIFIER

input : $V = (M^v, E_o \cup E_i, \delta^v, m_0^v)$ and $\mathscr{E} = (M, E_o, \delta, m_0)$
output: $V_u = (Y \cup Z, E_o \cup M^v, f_{yz} \cup f_{zy}, y_0)$

$y_0 := m_0^v$, $Y := \{y_0\}$
for $y := (m^d, m^f) \in Y$ *that has 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 $z := (y,e) \in Z$ *that has not been examined* **do**
  for $m' \in M^v$ **do**
    if $\exists s_I \in E_i^*$ *s.t.* $m' = \delta^v(m, s_I)$ *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 until all accessible part has been built

Algorithm 4: PRUNE-INADMISSIBLE

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

Mark all the Y states in V_u
Let E_o be uncontrollable and M_v be controllable
$V_u^{trim} := Trim(V_u)$
Obtain $[\mathscr{L}_m(V_u^{trim})]^{\uparrow C}$ w.r.t $\mathscr{L}(V_u)$ by following the standard $\uparrow C$ algorithm in [5]
Return the automaton representation of $[\mathscr{L}_m(V_u^{trim})]^{\uparrow C}$

Algorithm 5: BUILD-L_{pp}

input : AIS$= (Y \cup Z, E_o \cup M_v, f_{AIS,yz} \cup f_{AIS,zy}, y_0)$ and $\mathscr{E}^d = (M^d, E_o, \delta^d, m_0^d)$
output: $\mathscr{E}^d_{pp}$

$Q := \{y_0\}$, $\delta^d_{pp} := \emptyset$
for $y = (m^d, m^f) \in Q$ *that has not been examined* **do**
  for $e \in E_o$ *s.t.* $f_{AIS,yz}(y,e)$ *is defined* **do**
    $z := f_{AIS,yz}(y,e)$
    if $f_{zy}(z,y)$ *is defined* **then**
      $Q := Q \cup \{f_{zy}(z,y)\}$
      $\delta^d_{pp}(m^d, e) := \delta^d(m^d, e)$
$\mathscr{E}^d_{pp} := (M^d, E_o, \delta^d_{pp}, m_0^d)$
Return $Trim(\mathscr{E}^d_{pp})$

from IS_{pp} is guaranteed to satisfy $\mathscr{M}_i[f_I^{str}(L_{pp})] = L_{pp}$ and $\mathscr{M}_i[f_I^{str}(P[\mathscr{L}(G)])] = L_{pp}$, where $L_{pp} \subseteq L_{safe}$. By Theorem 1, f_I is PP-enforcing. ■

We now discuss the computational complexity of the INPRIVALIC Algorithm. Consider a system with estimator $\mathscr{E}$. According to [14], the AIS has at most $(|E_o|+1)|X_{\mathscr{E}}|^2$ number of states, where $|X_{\mathscr{E}}|$ is the number of states in $\mathscr{E}$. Since IS_{pp} is a substructure of the AIS, it also has

Algorithm 6: UNFOLD-PP-VERIFIER

input : $V_{pp} = (M^v_{pp}, E_o \cup E_i, \delta^v_{pp}, m^v_{0,pp})$,
$\mathscr{E} = (M, E_o, \delta, m_0)$, $\mathscr{E}^d_{pp} = (M^d_{pp}, E_o, \delta^d_{pp}, m^d_{0,pp},)$
output: $V_{u,pp} = (Y \cup Z, E_o \cup M^v \cup \{\varepsilon\}, f_{yz} \cup f_{zy}, y_0)$

1 $y_0 := m^v_{0,pp}$, $Y := \{y_0\}$, and $Y_{pp} := \{y_0\}$
2 **for** $y := (m^d, m^f) \in Y$ *that has not been examined* **do**
  for $e \in E_o$ **do**
    if $\delta(m^f, e)$ *is defined* **then**
      $f_{yz}(y,e) := (y,e)$
      $Z := Z \cup \{f_{yz}(y,e)\}$
3 **for** $z := (y,e) = ((m^d, m^f), e) \in Z$ *that has not been examined* **do**
  if $y \in Y_{pp}$ *and* $\delta^d_{pp}(y,e)$ *is defined* **then**
    $f_{zy}(z,\varepsilon) := \delta^d_{pp}(y,e)$
    $Y := Y \cup \{f_{zy}(z,\varepsilon)\}$
    $Y_{pp} := Y_{pp} \cup \{f_{zy}(z,\varepsilon)\}$
  else
    for $m' \in M^v_{pp}$ **do**
      if $\exists s_I \in E^*_i$ *s.t.* $m' = \delta^v_{pp}(m, s_I)$ *and* $\delta^v_{pp}(m', e)$ *is defined* **then**
        $f_{zy}(z,m') := \delta^v_{pp}(m', e)$
        $Y := Y \cup \{f_{zy}(z,m')\}$
4 Go back to step 2 until all accessible part has been built

Algorithm 7: SYNTHESIZE

input : $\mathrm{IS}_{pp} = (Y \cup Z, E_o \cup M^v \cup \{\varepsilon\}, f_{IS,yz} \cup f_{IS,zy}, y_0)$
output: $IA = (X_{ia}, E_o, E^*_i E_o, f_{ia}, q_{ia}, x_{ia,0})$

1 $x_{ia,0} := y_0$, $X_{ia} := \{x_{ia,0}\}$
2 **for** $x_{ia} \in X_{ia}$ *that has not been examined* **do**
  for $e \in E_o$ *s.t.* $f_{IS,yz}(x_{ia}, e)$ *is defined* **do**
    $z := f_{IS,yz}(x_{ia}, e) = (x_{ia}, e)$
    if $f_{IS,zy}(z,\varepsilon)$ *is defined* **then**
      $x'_{ia} := f_{IS,zy}(z,\varepsilon)$
      $f_{ia}(x_{ia}, e) = x'_{ia}$
      $q_{ia}(x_{ia}, e) = e$
    else
      Select $m' \in M_{v,mo}$ s.t. $f_{IS,zy}(z,m')$ is defined
      Select string $s_I \in Ins(x_{ia}, m')$
      $x'_{ia} := f_{IS,zy}(z,m')$
      $f_{ia}(x_{ia}, e) = x'_{ia}$
      $q_{ia}(x_{ia}, e) = s_I e$
    $X_{ia} := X_{ia} \cup \{x'_{ia}\}$

at most $(|E_o|+1)|X_{\mathscr{E}}|^2$ states. The time complexity for building the AIS is $O(|X_{\mathscr{E}}|^6)$ according to [14]. To build IS_{pp}, step 5 requires a breadth-first search on the AIS, which is $O(|X_{\mathscr{E}}|^2)$, and steps 6-8 are the same as steps 2-4 but on smaller automata, which is $O(|X_{\mathscr{E}}|^6)$. Hence, the time complexity for building IS_{pp} is also $O(|X_{\mathscr{E}}|^6)$. Finally, step 9 is done by performing a breadth-first search on IS_{pp}, which requires time complexity $O(|X_{\mathscr{E}}|^6)$. In all, the computational complexity of the INPRIVALIC Algorithm is $O(|X_{\mathscr{E}}|^6)$.

Remark 1: The algorithm proposed here only synthesizes a special class of PP-enforcing insertion functions. While every synthesized insertion function is guaranteed to be PP-enforcing, there are PP-enforcing insertion functions that cannot be synthesized by the INPRIVALIC Algorithm; e.g., the insertion function in Example 4. However, this does not necessarily imply that the INPRIVALIC Algorithm is incomplete. The INPRIVALIC Algorithm is sound; but whether it is also complete remains an open problem.

V. Conclusion

This paper extends our prior work and considers the synthesis of insertion functions that enforce opacity when the implementation of the insertion function is kept private as well as when it becomes known to the public. We have characterized the property of public-and-private enforceability that insertion functions need to satisfy in order to enforce opacity in both scenarios. We have established sufficient conditions for an insertion function to be PP-enforcing. Based on this condition, we have developed an algorithm that synthesizes a PP-enforcing insertion function.

References

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

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

[8] Y. Falcone and H. Marchand. Runtime enforcement of K-step opacity. In *Proc. of the 52nd IEEE Conference on Decision and Control*, 2013.

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

[10] L. Mazaré. Using unification for opacity properties. *Proc. of the 4th IFIP WG1*, 7:165–176, 2003.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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