Automatica 93 (2018) 369–378

Contents lists available at ScienceDirect

Automatica

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

Enforcement of opacity by public and private insertion functions☆

Yiding Ji [a,*], Yi-Chin Wu [a,b,1], Stéphane Lafortune [a]

[a] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA
[b] Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720, USA

ARTICLE INFO

Article history:
Received 13 October 2016
Received in revised form 26 October 2017
Accepted 3 February 2018

Keywords:
Discrete event systems
Privacy
Opacity
Opacity enforcement
Insertion function

ABSTRACT

We study the enforcement of opacity, an information-flow security property, using insertion functions that insert fictitious events at the output of the system. The intruder is characterized as a passive external observer whose malicious goal is to infer system secrets from observed traces of system events. We consider the problems of enforcing opacity under the assumption that the intruder either knows or does not know the structure of the insertion function; we term this requirement as public–private enforceability. The case of private enforceability alone, where the intruder does not know the form of the insertion function, is solved in our prior work. In this paper, we address the stronger requirement of public–private enforceability, that requires opacity be preserved even if the intruder knows or discovers the structure of the insertion function. We formulate the concept of public–private enforceability by defining the notion of public safety. This leads to the notion of public–private enforcing (PP-enforcing) insertion functions. We then identify a necessary and sufficient condition for an insertion function to be PP-enforcing. We further show that if opacity is privately enforceable by the insertion mechanism, then it is also public–private enforceable. Using these results, we present a new algorithm to synthesize PP-enforcing insertion functions by a greedy-maximal strategy. This algorithm is the first of its kind to guarantee opacity when insertion functions are made public or discovered by the intruder.

1. Introduction

Opacity is an information-flow security property that characterizes whether or not "secrets" of a given dynamic system can be inferred by an outside observer termed the *intruder*, because of its potentially malicious intentions. Due to its general formulation that is applicable to many security and privacy issues that arise in networked systems, opacity has received a lot of attention in the literature on security and privacy since it was first introduced in Mazaré (2004). 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 of 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*). When opacity holds, the intruder is never sure if the system's output corresponds to a secret or a non-secret behavior.

Various representations of the system secret have been considered in the study of opacity. These representations have led to the formalization of several notions of opacity for event-driven models of dynamic systems. In the context of automata models, the notions of initial-state opacity, current-state opacity, language-based opacity, K-step opacity and infinite step opacity, have been proposed; see, e.g., Cassez, Dubreil, and Marchand (2012), Lin (2011), Saboori and Hadjicostis (2012b, 2013) and Yin and Lafortune (2017). Opacity for infinite state systems is considered in Chédor, Morvan, Pinchinat, and Marchand (2015) while opacity under so-called Orwellian observers is investigated in Mullins and Yeddes (2014). Opacity for Petri net models has been considered in Bryans, Koutny, and Ryan (2005) and Tong, Li, Seatzu, and Giua (2017b), among others. In addition, several stochastic notions of opacity have been defined and investigated; see, e.g., Bérard, Chatterjee, and Sznajder (2015), Bérard, Mullins, and Sassolas (2015), Chen, Ibrahim, and Kumar (2017), Keroglou and Hadjicostis (2013) and Saboori and Hadjicostis (2014). In Yin, Li, Wang, and Li (2017), an algorithm was proposed for verification of infinite-step opacity in stochastic discrete event system. The recent

☆ 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. The material in this paper was partially presented at the 54th IEEE Conference on Decision and Control, December 15–18, 2015, Osaka, Japan. This paper was recommended for publication in revised form by Associate Editor Christoforos Hadjicostis under the direction of Editor Christos G. Cassandras.

* Corresponding author.
E-mail addresses: jiyiding@umich.edu (Y. Ji), ycwu@umich.edu (Y.-C. Wu), stephane@umich.edu (S. Lafortune).
[1] Current address of the second author: Pure Storage, Mountain View, CA 94041, USA.

https://doi.org/10.1016/j.automatica.2018.03.041
0005-1098/

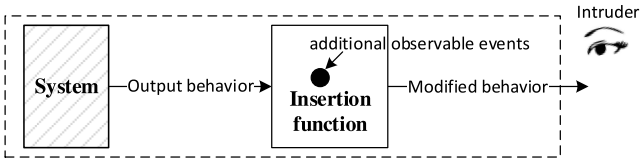


Fig. 1. The insertion mechanism.

survey paper (Jacob, Lesage, & Faure, 2016) may be consulted for a detailed review of the literature on this topic.

When a given notion of opacity is violated, researchers have proposed various methods for its enforcement. One popular approach is to design a minimally restrictive supervisor, which disables behaviors that violate opacity (Darondeau, Marchand, & Ricker, 2015; Dubreil, Darondeau, & Marchand, 2010; Saboori & Hadjicostis, 2012a; Takai & Oka, 2008). The work in Tong, Li, Seatzu, and Giua (2017a) adopts a similar approach but focuses on enforcing opacity with incomparable observations. The approach in Yin and Lafortune (2016) is to embed in a finite structure all feasible supervisors that enforce opacity and use this structure to synthesize one supervisor with desired properties. The work in Zhang, Shu, and Lin (2015) also lies in this category but discusses the problem from the perspective of maximum information release. Several works, such as Cassez et al. (2012), Yin and Lafortune (2015) and Yin and Li (2018), apply a sensor activation framework to enforce opacity by building dynamic observers or most-permissive observers. In Ylies and Hervé (2015), the authors consider a delay mechanism, which enforces K-step opacity or infinite-step opacity by delaying outputting system events until the secret expires.

In contrast to the above approaches for enforcing opacity, we proposed in our prior work (Wu & Lafortune, 2014) an insertion mechanism that enforces opacity by inserting fictitious events at the system's output. Such events are assumed to be indistinguishable from genuine ones from the viewpoint of the intruder. As in Wu and Lafortune (2014), our approach in this paper considers event-driven dynamic systems modeled as automata. Specifically, the system is a partially-observed and/or nondeterministic finite-state automaton, and the secret is modeled as a set of secret states in the automaton's state space. The insertion mechanism, which is depicted in Fig. 1, acts as an interface at the output of the system; hence, it does not interfere with the system, in contrast to the supervisory control based approaches. Insertion functions can be generalized to edit functions that allow event erasure and replacement, as considered in Ji and Lafortune (2017) and Wu, Raman, Rawlings, Lafortune, and Seshia (2017). However, we focus on insertion functions in this paper. The method of insertion functions has also been extended in Ji, Yin, and Lafortune (2018) to study opacity enforcement under quantitative constraints.

In Wu and Lafortune (2014), it is assumed that the insertion function used by the system is always kept private from the intruder. With this assumption, we have shown how to synthesize insertion functions that only output strings consistent with the non-secret behavior of the system and thus prevent the intruder from being certain that a secret behavior has occurred. In this paper, we relax that assumption. While the implementation of the insertion function may be kept private at first, a sophisticated intruder may learn the full set of modified behaviors output by the insertion function, compare it with the system model, and potentially reverse engineer the insertion function. Also, if the intruder knows the system's optimality criteria, it may follow the optimal synthesis algorithm in Wu and Lafortune (2016) and discover the correct insertion function. It may also be the case that the system designers decide to make the insertion function public, as is done in public-key cryptography, for example. *Hence, there is a need to design insertion functions that enforce opacity even when their implementation becomes known to the intruder.* Under the same insertion mechanism as in Fig. 1, to enforce opacity regardless of whether or not the intruder knows the implementation of the insertion function, we formally characterize a property called *public-and-private enforceability*, or *PP-enforceability* for short. A PP-enforcing insertion function is guaranteed to enforce opacity when the insertion function is kept private *and* when it becomes known to the intruder. In the former case, the insertion function outputs only behaviors consistent with non-secret behaviors of the system. In the latter case, the insertion function is designed such that for every secret behavior of the system, there is a non-secret behavior of the system that has the same modified output from the insertion function.

The main contributions of this paper are as follows. First, we formally characterize the properties of public enforceability and of public–private enforceability, in the context of opacity enforcement by insertion functions. We present conditions for PP-enforceability and use them to derive an effective test under which opacity is public–private enforceable. It turns out that if there exists an insertion function that is privately enforcing, then there also exists a (potentially different) insertion function that is PP-enforcing. This result is established by defining a so-called greedy criterion for selecting insertion functions in the All Insertion Structure (AIS) introduced in Wu and Lafortune (2014). These new results lead to an algorithmic procedure, called Algorithm INPRIVALIC-G, that is guaranteed to synthesize a PP-enforcing insertion function if one exists.

The remaining sections of this paper are organized as follows. Section 2 introduces the system model and the notion of opacity. Section 3 formally introduces insertion functions and the notion of *public-and-private enforceability*, along with conditions under which private enforceability and public–private enforceability hold for a given insertion function. Section 4 starts by reviewing the construction procedure of the All Insertion Structure (AIS) from Wu and Lafortune (2016) and then identifies relevant concepts and properties. In Section 5, we first present a sufficient condition for insertion functions to be PP-enforcing, then define the greedy criterion and show that a greedy insertion function is PP-enforcing. Then, in Section 6, the INPRIVALIC-G Algorithm is presented, which synthesizes PP-enforcing insertion functions by using a greedy-maximal insertion criterion within the AIS. Finally, Section 7 concludes the paper.

Preliminary versions of some of the results in sections 3.3 and 5.1 appear in Wu and Lafortune (2015). The results in Sections 4.2, 5.2 and 6 are new and do not appear in Wu and Lafortune (2015). In particular, Algorithm INPRIVALIC-G of Section 6 is guaranteed to output a PP-enforcing insertion function (if one exists) and is a generalized and improved version of Algorithm INPRIVALIC in Wu and Lafortune (2015), which outputs such a function only under certain conditions.

2. Opacity notions for automata models

We consider opacity problems in event-driven dynamic systems. We assume the system's state space is finite. Thus, the dynamic system of interest is modeled as an automaton $G = (X, E, f, X_0)$, where X is the finite set of states, E is the finite set of events, f is the partial state transition function $f : X \times E \to 2^X$, and $X_0 \subseteq X$ is the set of initial states. We allow G to be nondeterministic, which explains why the codomain of f is the power set of X. The transition function is extended to domain $X \times E^*$ in the standard manner (Cassandras & Lafortune, 2008); we still denote the extended function by f. Also, we use the notation $s < u$ to denote that string s is a prefix of string u. 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 $\mathcal{L}(G, X_0) := \{t \in E^* : (\exists x \in X_0)[f(x, t) \text{ is defined}]\}$. We will write $\mathcal{L}(G)$ as short-hand for $\mathcal{L}(G, X_0)$. The system G 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 observable projection 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 $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string.

The setting of the opacity problem considered in this paper is as follows: (1) the system is a partially observable and/or nondeterministic finite-state automaton G, as defined above; (2) G has a *secret* (formally defined below); (3) the intruder is an observer of G that knows the structure of G but only observes the observable projections of the strings in $\mathcal{L}(G)$. Hence, the intruder uses its knowledge of the structure of G and its online observations to infer if the current observed string reveals the secret or not. 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*).

Four notions of opacity studied in the literature are relevant to this paper: current-state opacity (CSO), initial-state opacity (ISO), language-based opacity (LBO), and initial-and-final-state opacity (IFO). In CSO, the secret of the system is modeled by a set of *secret states* denoted by X_S, where $X_S \subset X$; the secret is defined analogously for the other notions. Since it was shown in Wu and Lafortune (2013) that these four notions can be mapped to one and another by polynomial algorithms, we derive our results in this paper using only the notion of current-state opacity. The mappings presented in Wu and Lafortune (2013) can be used to apply our results to the other three notions.

Definition 1 (*Current-State Opacity (CSO)*)**.** Given system $G = (X, E, f, X_0)$, projection P, and the set of secret states X_S, G is CSO if $\forall t \in L_S := \{t \in \mathcal{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap X_S \neq \emptyset\}$, $\exists t' \in L_{NS} := \{t \in \mathcal{L}(G, X_0) : \exists x_0 \in X_0, f(x_0, t) \cap (X \setminus X_S) \neq \emptyset\}$ such that $P(t) = P(t')$.

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

To verify the above-mentioned opacity notions, we build the corresponding *forward state estimator* and check if any estimate contains only 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 $s = P(t) \in P[\mathcal{L}(G)]$ is the intruder's [current-state; initial-state; initial-and-final-state] *estimate* when the intruder observes string s after the system generated string t. Specifically, CSO and LBO can be verified by the standard *observer automaton* defined in section 2.5.2 of Cassandras and Lafortune (2008); ISO and IFO can be verified by the trellis-based initial-state estimator introduced in Saboori and Hadjicostis (2013). For simplicity, we will call a forward state estimator an *estimator* and denote it by $\mathscr{E}$ hereafter. Clearly, CSO holds if no state of the observer automaton is a subset of X_S, i.e., for all $t \in \mathcal{L}(G)$, $\mathscr{E}(P(t)) \cap (X \setminus X_S) \neq \emptyset$.

3. Insertion mechanism for opacity enforcement

To enforce opacity, we proposed in our prior work (Wu & Lafortune, 2014) an insertion mechanism, which inserts additional *fictitious* events to the output of the system. As shown in Fig. 1, the insertion function is an interface between the system and outside observers. It receives the system's output behavior, inserts fictitious events if necessary, and outputs the modified output. The set of allowed events to be inserted is defined to be equal to E_o. We assume that the intruder cannot distinguish between an inserted event and the corresponding genuine one in E_o.[2] We wish to design insertion functions according to different specifications based on the intruder's knowledge: *private*, *public*, and *public–private* enforceability. The remainder of this paper will focus on the strongest specification among the three: public–private enforceability.

3.1. Insertion functions and insertion automata

An insertion function is defined as a (potentially partial) function $f_I : E_o^* \times E_o \to E_o^*$ that outputs a string with inserted events based on the past observed behavior and the current observed event. Given observable string $se_o \in P[\mathcal{L}(G)]$, $f_I(s, e_o) = s_I e_o$ when string $s_I \in E_o^*$ is inserted before e_o. We also define the *string-based* version of f_I, denoted by f_I^{str}, recursively from f_I: $f_I^{str}(\epsilon) = \epsilon$ and $f_I^{str}(se_o) = f_I^{str}(s)f_I(s, e_o)$. Given G, the modified language output by insertion function f_I is denoted by $f_I^{str}(P[\mathcal{L}(G)]) = \{\tilde{s} \in E_o^* : \exists s \in P[\mathcal{L}(G)], f_I^{str}(s) = \tilde{s}\}$. When multiple events are inserted, we assume that they are inserted, hence observed, one by one.[3] Notice that the insertion functions f_I (and corresponding f_I^{str}) considered in this paper are deterministic.

We encode a given insertion function as an input/output (I/O) automaton $IA = (X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$ and call it an *insertion automaton*. The state set X_{ia} of IA could potentially be infinite. The input set is E_o; the output set is a set of *strings* in $E_o^+ = E_o^* E_o$; the transition function f_{ia} defines the dynamics of IA; the output function q_{ia} is defined such that $q_{ia}(x, e_o) = s_I e_o$ where $f_{ia}(x_{ia,0}, s) = x$, if $f_I(s, e_o) = s_I e_o$; and finally $x_{ia,0}$ is the initial state. More details on I/O automata can be found in Cassandras and Lafortune (2008).

3.2. Private enforceability

In Wu and Lafortune (2014), 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.[4] Admissibility is an input property for insertion functions; it requires insertion functions to be defined for all $P[\mathcal{L}(G)]$. This property is required since the system's behavior cannot be interfered with. For example, in applications where users query servers, we do not want to exclude any query from a given user.

Definition 2 (*Admissibility*)**.** Consider G, P, L_S and L_{NS}. An insertion function f_I is admissible if: $\forall se_o \in P[\mathcal{L}(G)]$, where $s \in E_o^*$, $e_o \in E_o$, $\exists s_I \in E_o^*$ s.t. $f_I(s, e_o) = s_I e_o$.

Since no behavior in $P[\mathcal{L}(G)]$ 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 $P[\mathcal{L}(G)]$. If $L_1 \subseteq L_2 \subseteq P[\mathcal{L}(G)]$, then $f_I^{str}(L_1) \subseteq f_I^{str}(L_2)$.*

[2] A trusted observer possibly could do so, using some predetermined mechanism known only to the system and trusted observer.

[3] This requirement is consistent with our prior work (Wu & Lafortune, 2015), although it is not explicitly stated there.

[4] Private enforceability was termed "i-enforceability" and private safety was simply termed "safety" in Wu and Lafortune (2014). We adopt the new terminology in order to distinguish the private and the public cases.

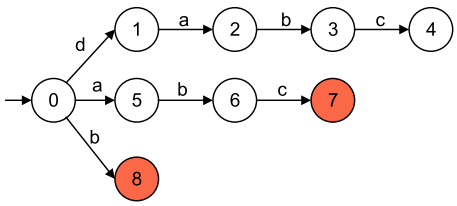


Fig. 2. Current-state estimator $\mathscr{E}$; states 7 and 8 contain only secret states.

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

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

This equation is an application of a result in Kumar and Garg (2012) and a similar expression was also proposed in Dubreil et al. (2010). Hereafter, we call a string $s \in P[\mathcal{L}(G)]$ *safe* if it is in L_{safe} and *unsafe* otherwise, so $L_{unsafe} = P[\mathcal{L}(G)] \setminus L_{safe}$. From the definition of safe language, if a string is unsafe, then all its continuations are unsafe.

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

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

3.3. Private-and-public enforceability

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

Example 1. Consider the current-state estimator in Fig. 2. These estimator states represent sets of system states; they are numbered from 0 to 8 for simplicity. Assume that states 7 and 8 contain only secret states; i.e., these estimator states reveal the secret. Suppose that opacity is enforced by the privately enforcing insertion function where $f_I^{str}(b) = ab, f_I^{str}(a) = da$ and no other insertions are made. If the intruder has no knowledge of f_I, then it would never conclude that the secret is revealed, as the output from f_I is always safe; here, $L_{safe} = \overline{\{dabc, ab\}}$. However, if the intruder knows the implementation of f_I, then it would be able to conclude that the state estimate is state 8 when it observes ab. This is because if ab were the genuine output behavior from the system, then it would have been modified to dab; and the intruder knows that. Hence, the only system output that would produce ab is string b.

Example 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 Angluin (1987), to learn the modified system $\tilde{G}$, which is the parallel composition of G with insertion automaton IA,[5] and then use $\tilde{G}$ and G to reverse engineer IA. Alternatively, if the intruder knows the optimality criteria used by the system's designer, it could follow certain synthesis algorithm and construct the correct insertion function. In either case, we wish to use an insertion function that still enforces opacity when its implementation becomes known. In this manner, the system designers may be able to eventually reveal the structure of f_I, if so desired.

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

Definition 4 (*Public Safety*)**.** Consider G with P, L_S and L_{NS}. An insertion function f_I is publicly safe if $\forall \tilde{s} \in f_I^{str}(P[\mathcal{L}(G)]), \exists t \in L_{safe}$ s.t. $f_I^{str}(t) = \tilde{s}$; equivalently, $f_I^{str}(P[\mathcal{L}(G)]) \subseteq f_I^{str}(L_{safe})$.

In contrast to Definition 4 in Wu and Lafortune (2015), we use L_{safe} instead of $P(L_{NS})$ in the above definition to better capture the on-line operation of the system, where public safety must be preserved for every prefix of a safe string. The idea behind public safety is that no matter what the insertion function outputs, this output could have been obtained from a safe string; hence opacity holds.

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

Definition 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 $\tilde{s} = ab$, there is no $t \in L_{safe}$ for which $f_I^{str}[P(t)] = ab$. Let us define another insertion function: $f_I'(\epsilon, a) = da, f_I'(\epsilon, b) = dab$, and $f_I'(s, e_o) = e_o, \forall s e_o \in P[\mathcal{L}(G)] \setminus \{a, b\}$. One can verify that f_I' is PP-enforcing. Specifically, f_I' is admissible because it is defined for every $P[\mathcal{L}(G)]$; it is privately safe as $f_I'(P[\mathcal{L}(G)]) = \overline{\{dabc\}} \subseteq L_{safe}$; also, f_I' is publicly safe since for every $\tilde{s} \in \overline{\{dabc\}}$, there exists $t \in L_{safe}$ that is observationally equivalent and is unmodified by f_I', which is sufficient to ensure the condition in Definition 4.

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

[5] This type of parallel composition of a regular automaton with an I/O one is sometimes called "input parallel composition"; we refer the reader to Wu and Lafortune (2014) for its formal definition.

Example 3. Consider the current-state estimator with strings $\overline{\{ab, b\}}$, where string ab is secret. Consider the insertion function f_I: $f_I(\epsilon, b) = ab$ and $f_I(s, e_o) = e_o, \forall s e_o \in \overline{\{ab\}}$. 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.

The issue in the preceding example is that a publicly safe insertion function is free to map strings to anything, as long as the condition in Definition 4 holds. It is not required that the output string be safe. This explains our choice of using PP-enforceability as our specification for insertion functions. We do not wish to make any assumptions about the intruder's knowledge, either at the outset or as it keeps observing the system. Thus, insertion functions should enforce opacity regardless of what the intruder knows about the implementation of insertion function, including nothing. Hence, by also requiring private safety, PP-enforceability ensures that only safe strings will be output.

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

4. All insertion structure and analysis

We originally developed in Wu and Lafortune (2014) a procedure to synthesize privately enforcing insertion functions based on a special discrete structure called the *All Insertion Structure* (AIS). In this section, we start by reviewing the process of building the AIS, but following the procedure in Wu and Lafortune (2016), which is more efficient than the one in Wu and Lafortune (2014, 2015).

4.1. Construction of the AIS

The review of the construction procedure of the AIS herein is necessary in order to explain how we employ this structure for the purposes of this paper and also to define relevant notations. The AIS is a game-like bipartite structure between the system and the insertion function, with so-called Y states and Z states. When the system plays, it outputs an observable event e_0 that is defined at the current Y-state y of the AIS, and it leads to a Z-state $z = (y, e_0)$ in the AIS. On the other hand, when the insertion function plays, certain insertion decisions are made at Z-state z corresponding to strings that can be inserted before the last observed event e_0. As shown in Wu and Lafortune (2014), the AIS embeds in its transition structure *all* privately enforcing insertion functions.

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

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

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

Definition 6 (*Verifier Parallel Composition* $\|_v$)**.** The verifier parallel composition is a special kind of parallel composition between automata $\mathscr{E}^d$ and $\mathscr{E}^f$. Two kinds of transition functions, $\delta_{vs} : (M_d \times M) \times E_o \to (M_d \times M)$ and $\delta_{vd} : (M_d \times M) \times E_o \to (M_d \times M)$, are defined for synchronization:

$$V := (M_v, E_o, \delta_{vd}, \delta_{vs}, m_{v0}) = \mathscr{E}^d \|_v \mathscr{E}^f = Ac(M_d \times M, E_0, \delta_{vd}, \delta_{vs}, (m_0, m_0))$$

where the transition functions are defined as

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

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

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

In step (2) of the AIS construction, we "unfold" all deterministic insertion decisions from the i-verifier resulting in a game structure between the "system player" G and the "insertion function player"; we call this structure the *unfolded verifier*. This unfolding procedure is given in Algorithm 1 in Wu and Lafortune (2016). The essence of the construction is to: (i) include all possible system plays, i.e., newly-generated observable events, at a given Y-state, and (ii) include all insertions that are possible *before* that observable event at a given Z-state, based on existing paths of inserted transitions in the i-verifier.

In order to synthesize admissible insertion functions, in step (3) of the AIS construction, we follow Algorithm 2 in Wu and Lafortune (2016) to prune away all the inadmissible insertion decisions (i.e., those that lead to deadlock at Z-states, since the insertion function should always play) from the unfolded i-verifier and call the final bipartite structure the AIS. This iterative pruning and associated trimming is described in Algorithm 2 in Wu and Lafortune (2016). As explained in Wu and Lafortune (2016), it can be interpreted as a supremal controllable sublanguage calculation. Notice that there may be multiple paths of inserted events between two states m_v and m'_v in V and this is captured by the function $Ins(m_v, m'_v) = \{s_I \in E_o^* : \delta_{vd}(m_v, s_I) = m'_v\}$ in section IV. A of Wu and Lafortune (2016). (In contrast with Wu & Lafortune, 2016, we do not use the notation E_i in this paper since it is the same as E_o.) Notice that $Ins(m_v, m'_v)$ may be an infinite set if there is a cycle of inserted events in the path from m_v to m'_v. In this paper, we make the assumption that such cycles are redundant (from the viewpoint of event insertion) and extract only the finite set of cycle-free paths from m_v to m'_v, i.e., cycles of inserted events are replaced by ϵ.

The function Ins is used in line 5 of Algorithm 2 in Wu and Lafortune (2016) to label transitions from Z-states to Y-states in the AIS as sets of admissible strings that can be inserted when such transitions are taken. For the sake of simplicity of notation, we denote hereafter these sets by $L(z, y)$ for a given transition between state z and state y. It can be shown from the construction of V and of the AIS that any two $L(z, y_i)$ and $L(z, y_j)$ are disjoint for any two distinct successors y_i and y_j of z. Moreover, these sets are all finite since cycles of inserted events have been removed as mentioned above. As defined, the AIS does not pre-specify which

string in an $L(z, y)$ set is to be selected and thus all the possible insertion choices are encoded in it. The reader is referred to Wu and Lafortune (2014, 2016) for further details. As shown in Wu and Lafortune (2014), *opacity is privately enforceable if and only if the AIS is not empty.*

For the sake of completeness, we formally define this bipartite transition system. Let $I = M_d \times M$ denote the set of all information states.

Definition 7 (*All Insertion Structure*)**.** The All Insertion Structure w.r.t. current-state estimator $\mathscr{E}$ is the tuple: $AIS = (Y, Z, E_o, 2^{E_o^*}, f_{AIS,yz}, f_{AIS,zy}, \Gamma, y_0)$, where

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

Example 4. Here we show an example to illustrate the whole construction process of the AIS. The current state estimator $\mathscr{E}$ is the same as in Example 1 and is shown in Fig. 2. In this example, states 7 and 8 are secret states, so we delete them as well as transitions leading to them and then obtain the desired estimator $\mathscr{E}^d$ in Fig. 3. Next, we add self-loops for events $\{a, b, c, d\}$ at each state of $\mathscr{E}$ and obtain the feasible estimator $\mathscr{E}^f$ in Fig. 4. After that, we do the verifier parallel composition between $\mathscr{E}^d$ and $\mathscr{E}^f$ and obtain verifier V in Fig. 5. Notice that dashed transitions that are not followed by any solid transition are not shown in the figure. Those transitions do not indicate valid insertions and play no role in building the unfolded verifier. By the insertion mechanism, events are inserted before the occurrence of the next observable event, thus every δ_{vd} transition should be followed by a δ_{vs} transition somewhere in the verifier. Then we construct the unfolded verifier in Fig. 6, where the rectangular states are Y states and the oval states are Z states. As is seen in the figure, Z state $((6, 6), c)$ is a deadlock state and should be pruned away in the next step of building the AIS. Following Algorithm 2 in Wu and Lafortune (2016), the shaded path in V_u is pruned away. Finally, we obtain the AIS in Fig. 7.[6] The game starts at the initial Y-state $(0, 0)$ where the system plays; initially the system can output a, b, or d. If the system outputs b, the game then reaches Z-state $((0, 0), b)$, where the insertion function plays. The transition a between states $((0, 0), b)$ and $(6, 8)$ stands for insertion of event a and all the other transitions from Z states to Y states can be interpreted similarly. The insertion function can choose to insert a or da, leading the system to state $(6, 8)$ or $(3, 8)$, respectively.

4.2. Analysis of AIS

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

[6] For simplicity, since all sets labeling transitions from Z-states to Y-states in this example are singletons, "{" and "}"have been omitted in these labels. The same comment applies to later figures.

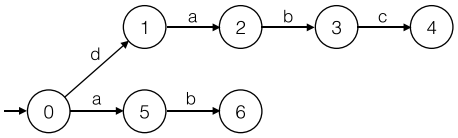


Fig. 3. Desired estimator $\mathscr{E}^d$.

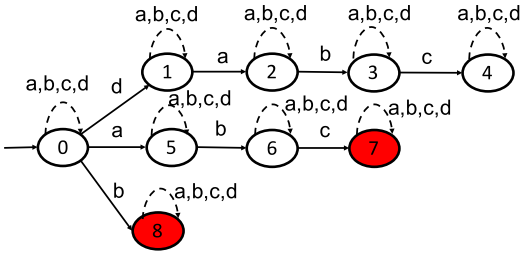


Fig. 4. Feasible estimator $\mathscr{E}^f$.

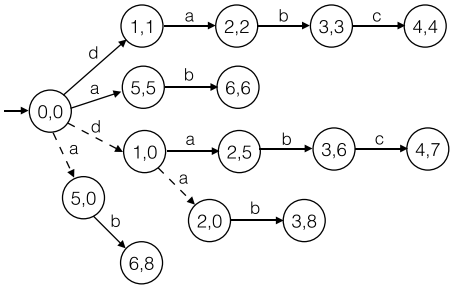


Fig. 5. Verifier V without dangling δ_{vd} transitions.

Definition 8 (*Run*)**.** A run ω in the AIS is a sequence of alternating states, observable events and insertion decisions.

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

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

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

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

Definition 9 (*String Generated by a Run*)**.** The string generated by run $\omega \in \Omega$ is defined as: $S(\omega) = s_0 e_0 s_1 e_1 \ldots s_{n-1} e_{n-1}$, given $\omega = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \cdots y_{n-1} \xrightarrow{e_{n-1}} z_{n-1} \xrightarrow{s_{n-1}} y_n \rangle$.

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

Definition 10 (*Partition of Safe Language*)**.** Safe language L_{safe} is partitioned as:

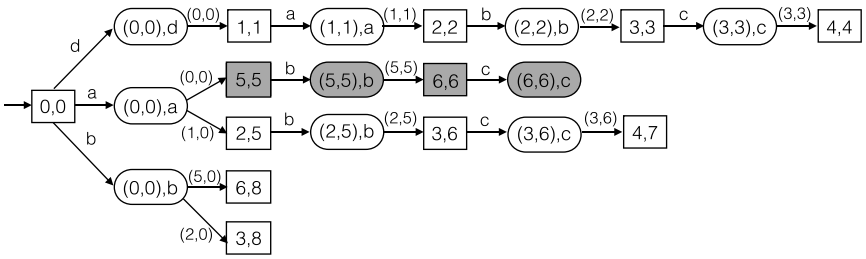


Fig. 6. Unfolded verifier V_u.

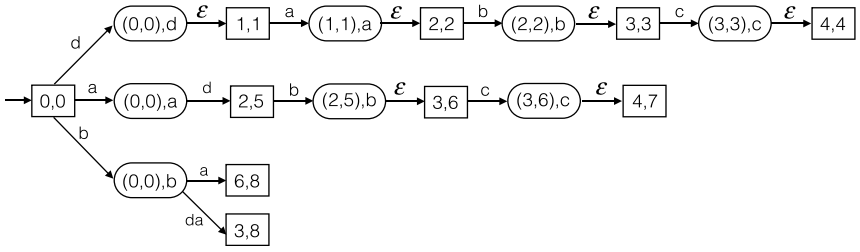


Fig. 7. AIS in Example 4.

(1) $L^1_{safe} = \overline{\tilde{L}_{safe}}$ where $\tilde{L}_{safe} = \{s \in L_{safe} : \nexists u \in L_{unsafe}, \text{ s.t., } s < u\}$.
(2) $L^2_{safe} = L_{safe} \setminus L^1_{safe}$.

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

Proposition 2. *$L^1_{safe} \neq \emptyset$ if private safety is enforceable.*

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

5. PP-enforcing insertion functions

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

5.1. A sufficient condition for PP-enforcing insertion functions

Based on the definitions, a privately safe f_I maps all strings in $P[\mathcal{L}(G)]$ to a subset of L_{safe}. However, in general, $f^{str}_I[P(L_S)]$ may not be a subset of $f^{str}_I(L_{safe})$. In this case, the intruder, when knowing the implementation of f_I, could determine the occurrence of the secret when it observes strings in $f^{str}_I[P(L_S)] \setminus f^{str}_I(L_{safe})$. If, on the other hand, $f^{str}_I[P(L_S)]$ is contained in $f^{str}_I(L_{safe})$, then $f^{str}_I(P[\mathcal{L}(G)]) = f^{str}_I(L_{safe})$ and thus f_I is PP-enforcing. A special case where $f^{str}_I[P(L_S)]$ is guaranteed to be contained in $f^{str}_I(L_{safe})$ is when $f^{str}_I(L_{safe})$ is the entire set L_{safe}. Based on this special case, Lemma 1 and Theorem 1 show sufficient conditions for a privately enforcing f_I to be PP-enforcing.

Lemma 1. *Consider privately enforcing insertion function f_I. If $f^{str}_I(L_{safe}) = L_{safe}$, then f_I is also publicly enforcing; that is, f_I is PP-enforcing.*

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

We now replace L_{safe} with a subset $L \subseteq L_{safe}$ and follow the argument in the proof of Lemma 1 to derive a more general condition in Theorem 1 (proof omitted since similar to that of Lemma 1).

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

The condition in Theorem 1 is sufficient and the following example shows a case when the theorem does not hold. Thus it remains to be seen whether a PP-enforcing insertion function can always be synthesized from the AIS.

Example 5. Consider system G with observable event set $E_o = \{a, b, c, d\}$ and observable language $P[\mathcal{L}(G)] = \overline{\{dabc, abc, bc, c\}}$, where $L_{safe} = \overline{\{dabc, abc, c\}}$. Define f_I so that $f_I(\epsilon, a) = da$, $f_I(\epsilon, b) = ab$, $f_I(\epsilon, c) = abc$ and $f_I(s, e_o) = e_o$ otherwise. Because $f^{str}_I[\mathcal{L}(G)] = \overline{\{abc, dabc\}} \subseteq L_{safe}$, f_I is privately enforcing. One can also check that f_I is publicly enforcing. However, the only set $L \subseteq L_{safe}$ satisfying $f^{str}_I(L) = L$ is $\overline{\{dabc\}}$, which is not equal to $f^{str}_I[\mathcal{L}(G)] = \overline{\{abc, dabc\}}$. Hence, f_I is a PP-enforcing insertion function such that no $L \subseteq L_{safe}$ satisfies $f^{str}_I[\mathcal{L}(G)] = L$ and $f^{str}_I(L) = L$.

5.2. Greedy PP-enforcing insertion functions

In this section, we introduce the notion of a *greedy-maximal* PP-enforcing insertion function and then leverage the results in Section 4.2 together with Theorem 1.

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

Definition 11 (*Greedy-Maximal Criterion*)**.**

(1) At any $z \in Z_1 \cup Z_2$ in the AIS, choose ϵ insertion.
(2) At any $z \in Z_3$ in the AIS, choose for insertion choice any string $s_{max} \in \arg\max[\,|s_i|,\ s_i \in \Gamma(z)]$ where $|\cdot|$ denotes the length of the string.

Any insertion function that satisfies the greedy-maximal criterion at every Z-state that it visits in the AIS is called a *greedy-maximal insertion function*, denoted as f_{greedy}. By this criterion, $f_{greedy}(L^1_{safe}) = L^1_{safe}$ since ϵ is chosen at every Z state. Moreover, $f_{greedy}(L^2_{safe} \cup L_{unsafe}) \subseteq L^1_{safe}$, a fact established below in the proof of Theorem 2. In order to prove that theorem, we first give definition of a particular projection P_e.

Definition 12 (*Projection P_e*)**.** Given a run $\omega = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \cdots y_{n-1} \xrightarrow{e_{n-1}} z_n \xrightarrow{s_{n-1}} y_n \rangle$ where y_0 is the initial state of the AIS, the edit projection P_e returns the string $P_e(\omega) = s = e_0 e_1 \ldots e_{n-1}$.

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

Theorem 2. *A greedy-maximal insertion function is PP-enforcing.*

Proof. Consider greedy-maximal insertion function f_{greedy}. First, by Proposition 2, $L^1_{safe} \neq \emptyset$. We also know that $\forall s \in L^1_{safe}, f_{greedy}(s) = s$, i.e., $f_{greedy}(L^1_{safe}) = L^1_{safe}$ by our greedy criterion.

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

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

Corollary 1. *Opacity is PP-enforceable if and only if it is privately enforceable.*

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

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

This result is a direct improvement of our preliminary work (Wu & Lafortune, 2015) in the sense that PP-enforcing insertion function always exists as long as privately safe insertion function exists. Let us revisit Example 5: it is clear that f_l is not greedy-maximal since $f_l(\epsilon, c) \neq dabc$. If we set $f_l(\epsilon, c) = dabc$, then we obtain a greedy-maximal insertion function that is PP-enforcing.

6. The INPRIVALIC-G Algorithm

In this section, we develop a new algorithm that synthesizes a PP-enforcing insertion function by leveraging Theorem 2. We first build the AIS, which embeds all privately enforcing insertion functions. The strategy of the proposed algorithm is to identify L^1_{safe} and modify all other strings to strings in L^1_{safe} by using the greedy-maximal criterion. As a result, any insertion function synthesized in that manner is guaranteed to be PP-enforcing by Theorem 2.

Because this algorithm synthesizes INsertion functions with PRIVAte-and-pubLIC-enforceability property using Greedy-maximal criterion, we call it the INPRIVALIC-G Algorithm.

Algorithm 1 INPRIVALIC-G ALGORITHM

Input: $G = (X, E, f, X_0)$, projection $P, X_s \subseteq X$
Output: A PP-enforcing IA
1: Build $\mathscr{E}$, $\mathscr{E}^d$, $\mathscr{E}^f$
2: $V = \mathscr{E}^d \|_v \mathscr{E}^f$
3: Construct All Insertion Structure (AIS) by algorithms in Wu and Lafortune (2016)
4: Synthesize a greedy insertion function from AIS

Hereafter, we denote a greedy-maximal insertion function by f_{greedy}. The INPRIVALIC-G Algorithm is not meant to synthesize *any* PP-enforcing insertion function, but it is guaranteed to find one (unless the AIS is empty).

We discuss the steps of the algorithm, as a way of summarizing the methodology developed in this paper. Steps 1 to 3 construct the AIS. These steps were already discussed earlier in Section 4.1 and will not be repeated here. After that, step 4 synthesizes an insertion automaton from the AIS using the greedy-maximal criterion. The main idea is that at each Z-state in the AIS, a greedy-maximal insertion choice is selected according to Definition 11 and this process proceeds until: (1) a terminating Y is reached; or (2) a previously visited Y state is visited again. It is implemented in Algorithm 2, which builds the reachable part of the AIS for the selections made, until a complete IA is obtained.

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

Example 6. Let automaton G with observable events $E_o = \{a, b, c, d, e\}$ have the state estimator shown in Fig. 8, where estimator state 7 reveals the secret. We use this example to illustrate all the steps of the INPRIVALIC-G Algorithm. Following the algorithm, we build the AIS and synthesize a PP-enforcing insertion function encoded by an I/O automation.

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

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

In step 3, we unfold the insertions in V for every system output, and build the game structure V_u. Since there is no inadmissible insertion in V_u, no state will be pruned away and the AIS is immediately obtained in Fig. 9. There are two types of states in the

Algorithm 2 Synthesize a greedy insertion function

Input: $AIS = (Y, Z, E_o, 2^{E_o^*}, f_{AIS,yz}, f_{AIS,zy}, \Gamma, y_0)$
Output: IA=$(X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$

$x_{ia,0} := y_o, X_{ia} := \{x_{ia,0}\}$
for $x_{ia} \in X_{ia}$ that has not been examined **do**
  for $e \in E_o$ s.t. $f_{AIS,yz}(x_{ia}, e)$ is defined and where $z = f_{AIS,yz}(x_{ia}, e) = (x_{ia}, e)$ **do**
    if $\epsilon \in \Gamma(z)$ **then**
      $x'_{ia} := f_{AIS,zy}(z, L(z, x'_{ia}))$ where $\epsilon \in L(z, x'_{ia})$
      $f_{ia}(x_{ia}, e) = x'_{ia}$
      $q_{ia}(x_{ia}, e) = e$
    else
      pick one $s_{max} \in \arg\max[|s_i|, s_i \in \Gamma(z)]$
      $x'_{ia} := f_{AIS,zy}(z, L(z, x'_{ia}))$ where $s_{max} \in L(z, x'_{ia})$
      $f_{ia}(x_{ia}, e) = x'_{ia}$
      $q_{ia}(x_{ia}, e) = s_{max}e$
    end if
    $X_{ia} := X_{ia} \cup \{x'_{ia}\}$
  end for
end for
return IA

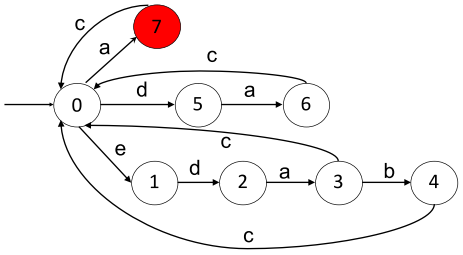


Fig. 8. $\mathscr{E}$ with secret-revealing state 7.

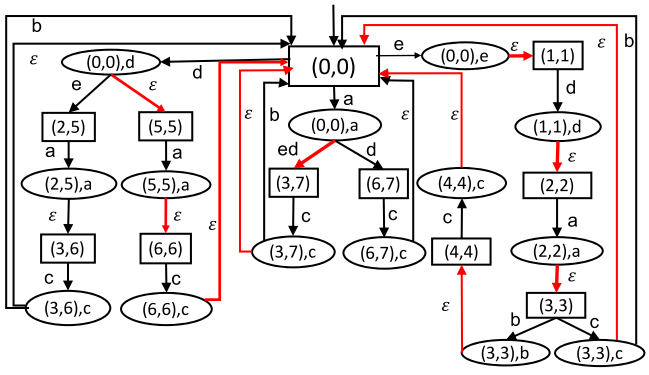


Fig. 9. All insertion structure with greedy-maximal criterion. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

AIS: square states where the system plays and round states where the insertion function plays.

With the AIS built, we proceed to the synthesis part. By the greedy-maximal criterion, at state $((0, 0), a)$, ed should be inserted and at state $((3, 7), c)$, ϵ should be inserted. Similarly for the other Z-states: we insert ϵ if it is defined. In Fig. 9 we use bold red lines to indicate the greedy-maximal criterion in the AIS. Finally, the insertion automaton in Fig. 10 encodes the constructed PP-enforcing insertion function.

We conclude with a brief discussion of the computational complexity of the INPRIVALIC-G Algorithm. Consider a system with estimator $\mathscr{E}$; as shown in Wu and Lafortune (2014), the AIS has at most $(|E_o| + 1)|X_{\mathscr{E}}|^2$ states, where $|X_{\mathscr{E}}|$ is the number of states in $\mathscr{E}$. The time complexity for building the AIS is of $O(|X_{\mathscr{E}}|^6)$ according to Wu and Lafortune (2016). Finally, the greedy-maximal synthesis step is done by performing a breadth-first search on the AIS, which requires time complexity linear in its size. In all, the computational complexity of the INPRIVALIC-G Algorithm is therefore of $O(|X_{\mathscr{E}}|^6)$. In the worst case, $|X_{\mathscr{E}}|$ may be $2^{|X|}$ and the complexity is exponential in terms of $|X|$. We refer the reader to Wu and Lafortune (2014) for numerical tests on the construction of the AIS using an explicit representation, and to Wu et al. (2017) for a symbolic implementation of the AIS construction using binary decision diagrams, which achieves greater scalability.

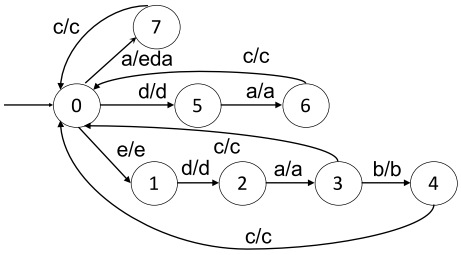


Fig. 10. A PP-enforcing insertion function encoded as an I/O automaton.

Remark 1. The INPRIVALIC-G Algorithm is sound and complete, unlike the INPRIVALIC Algorithm in Wu and Lafortune (2015), which was provably sound only.

7. Conclusion

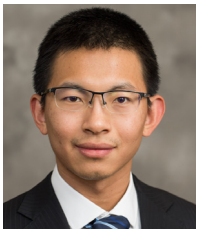
This paper extends prior works on opacity enforcement by insertion functions to the case where the insertion function may become known to the intruder. To handle this situation, we defined the notion of public–private (PP) opacity and investigated its enforcement by so-called PP-enforcing insertion functions.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

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

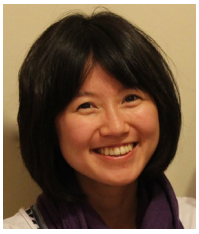
References

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, *75*(2), 87–106.
Bérard, B., Chatterjee, K., & Sznajder, N. (2015). Probabilistic opacity for Markov decision processes. *Information Processing Letters*, *115*(1), 52–59.
Bérard, B., Mullins, J., & Sassolas, M. (2015). Quantifying opacity. *Mathematical Structures in Computer Science*, *25*(Special issue 2), 361–403.
Bryans, J. W., Koutny, M., & Ryan, P. Y. A. (2005). Modelling opacity using Petri nets. *Electronic Notes in Theoretical Computer Science*, *121*, 101–115.
Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems –2nd Edition*. Springer.
Cassez, F., Dubreil, J., & Marchand, H. (2012). Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, *40*(1), 88–115.

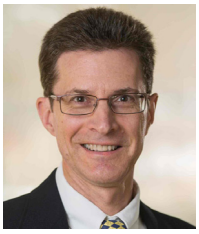
Chédor, S., Morvan, C., Pinchinat, S., & Marchand, H. (2015). Diagnosis and opacity problems for infinite state systems modeled by recursive tile systems. *Discrete Event Dynamic Systems: Theory and Applications*, *25*(1–2), 271–294.
Chen, J., Ibrahim, M., & Kumar, R. (2017). Quantification of secrecy in partially observed stochastic discrete event systems. *IEEE Transactions on Automation Science and Engineering*, *14*(1), 185–195.
Darondeau, P., Marchand, H., & Ricker, L. (2015). Enforcing opacity of regular predicates on modal transition systems. *Discrete Event Dynamic Systems: Theory and Applications*, *25*(1–2), 251–270.
Dubreil, J., Darondeau, P., & Marchand, H. (2010). Supervisory control for opacity. *IEEE Transactions on Automatic Control*, *55*(5), 1089–1100.
Jacob, R., Lesage, J.-J., & Faure, J.-M. (2016). Overview of discrete event systems opacity: models, validation, and quantification. *Annual Reviews in Control*, *41*, 135–146.
Ji, Y., & Lafortune, S. (2017). Enforcing opacity by publicly known edit functions. In *Proceedings of the 56th IEEE conference on decision and control* (pp. 4866–4871).
Ji, Y., Yin, X., & Lafortune, S. (2018). Opacity enforcement by insertion functions under energy constraints. In *Proceedings of the 14th IFAC international workshop on discrete event systems*.
Keroglou, C., & Hadjicostis, C. N. (2013). Initial state opacity in stochastic DES. In *Proceedings of the 18th IEEE conference on emerging technologies & factory automation* (pp. 1–8).
Kumar, R., & Garg, V. K. (2012). *Modeling and control of logical discrete event systems, Vol. 300*. Springer Science & Business Media.
Lin, F. (2011). Opacity of discrete event systems and its applications. *Automatica*, *47*(3), 496–503.
Mazaré, L. (2004). Using unification for opacity properties. *Proceedings of the 4th IFIP WG1*, *7*, 165–176.
Mullins, J., & Yeddes, M. (2014). Opacity with orwellian observers and intransitive non-interference. *IFAC Proceedings Volumes*, *47*(2), 344–349.
Saboori, A., & Hadjicostis, C. N. (2012a). Opacity-enforcing supervisory strategies via state estimator constructions. *IEEE Transactions on Automatic Control*, *57*(5), 1155–1165.
Saboori, A., & Hadjicostis, C. N. (2012b). Verification of infinite-step opacity and complexity considerations. *IEEE Transactions on Automatic Control*, *57*(5), 1265–1269.
Saboori, A., & Hadjicostis, C. N. (2013). Verification of initial-state opacity in security applications of discrete event systems. *Information Sciences*, *246*, 115–132.
Saboori, A., & Hadjicostis, C. N. (2014). Current-state opacity formulations in probabilistic finite automata. *IEEE Transactions on Automatic Control*, *59*(1), 120–133.
Takai, S., & Oka, Y. (2008). A formula for the supremal controllable and opaque sublanguage arising in supervisory control. *SICE Journal of Control, Measurement, and System Integration*, *1*(4), 307–311.
Tong, Y., Li, Z., Seatzu, C., & Giua, A. (2017a). Current-state opacity enforcement in discrete event systems under incomparable observations. *Discrete Event Dynamic Systems: Theory and Applications*, 1–22.
Tong, Y., Li, Z., Seatzu, C., & Giua, A. (2017b). Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, *62*(6), 2823–2837.
Wu, Y.-C., & Lafortune, S. (2013). Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems: Theory and Applications*, *23*(3), 307–339.
Wu, Y.-C., & Lafortune, S. (2014). Synthesis of insertion functions for enforcement of opacity security properties. *Automatica*, *50*(5), 1336–1348.
Wu, Y.-C., & Lafortune, S. (2015). Synthesis of opacity-enforcing insertion functions that can be publicly known. In *Proceedings of the 54th IEEE conference on decision and control* (pp. 3506–3513).
Wu, Y.-C., & Lafortune, S. (2016). Synthesis of optimal insertion functions for opacity enforcement. *IEEE Transactions on Automatic Control*, *61*(3), 571–584.
Wu, Y.-C., Raman, V., Rawlings, B. C., Lafortune, S., & Seshia, S. A. (2017). Synthesis of obfuscation policies to ensure privacy and utility. *Journal of Automated Reasoning*.
Yin, X., & Lafortune, S. (2015). A general approach for solving dynamic sensor activation problems for a class of properties. In *Proceedings of the 54th IEEE conference on decision and control* (pp. 3610–3615).
Yin, X., & Lafortune, S. (2016). A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, *61*(8), 2140–2154.
Yin, X., & Lafortune, S. (2017). A new approach for the verification of infinite-step and K-step opacity using two-way observers. *Automatica*, *80*, 162–171.
Yin, X., & Li, S. (2018). Synthesis of dynamic masks for infinite-step opacity. In *Proceedings of the 14th IFAC international workshop on discrete event systems*.
Yin, X., Li, Z., Wang, W., & Li, S. (2017). Infinite-step opacity of stochastic discrete-event systems. In *Proceedings of the 11th Asian control conference* (pp. 102–107).
Ylies, F., & Hervé, M. (2015). Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems: Theory and Applications*, *25*(4), 531–570.
Zhang, B., Shu, S., & Lin, F. (2015). Maximum information release while ensuring opacity in discrete event systems. *IEEE Transactions on Automation Science and Engineering*, *12*(3), 1067–1079.



Yiding Ji received the B.Eng. degree from Tianjin University, Tianjin, China in 2014, the M.S. degree from the University of Michigan, Ann Arbor, USA in 2016, all in electrical engineering. He is now a Ph.D. candidate in the Department of Electrical Engineering and Computer Science, the University of Michigan. His research interests include supervisory control of discrete-event systems, security and secrecy in cyber-physical systems and formal methods.



Yi-Chin Wu was a postdoctoral researcher at the TerraSwarm Research Center and affiliated with the University of Michigan and the University of California, Berkeley, when this work was performed. She currently works at Pure Storage Inc. She received her B.S. degree from National Taiwan University, Taipei, Taiwan, in 2008, and her Ph.D. 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. Lafortune is a Fellow of the IEEE (1999) and of IFAC (2017). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper coauthored with S.-L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). Lafortune's research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He co-authored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems — Second Edition* (Springer, 2008). Lafortune is Editor-in-Chief of the Journal of Discrete Event Dynamic Systems: Theory and Applications.