



Compositional and Abstraction-Based Approach for Synthesis of Edit Functions for Opacity Enforcement

Sahar Mohajerani , Yiding Ji , and Stéphane Lafortune , *Fellow, IEEE*

Abstract—This article develops a novel compositional and abstraction-based approach to synthesize edit functions for opacity enforcement in modular discrete event systems. Edit functions alter the output of the system by erasing or inserting events in order to obfuscate the outside intruder, whose goal is to infer the secrets of the system from its observation. We synthesize edit functions to solve the opacity enforcement problem in a modular setting, which significantly reduces the computational complexity compared with the monolithic approach. Two abstraction methods called opaque observation equivalence and opaque bisimulation are first employed to abstract the individual components of the modular system and their observers. Subsequently, we propose a method to transform the synthesis of edit functions to the calculation of modular supremal nonblocking supervisors. We show that the edit functions synthesized in this manner correctly solve the opacity enforcement problem.

Index Terms—Abstraction, edit function, finite-state automata, modular systems, opacity.

I. INTRODUCTION

OPACITY characterizes whether the integrity of the secrets of a system can be preserved from the inference of an outside intruder, potentially with malicious purposes. The intruder is modeled as a passive observer with knowledge of the system's structure. A system is called opaque if the intruder is unable to infer the system's secrets from its observation.

Starting with [2] and [3] in the computer science literature, opacity has been extensively studied, especially in the field of discrete event systems (DESs), under multiple frameworks.

For finite state automaton models, various notions of opacity have been studied, e.g., language-based opacity [22], current-state opacity [34], initial-state opacity [36], K-step opacity [49], and infinite-step opacity [33]. Opacity has also been discussed

Manuscript received January 29, 2019; revised May 30, 2019 and July 29, 2019; accepted September 21, 2019. Date of publication October 8, 2019; date of current version July 28, 2020. The work of S. Mohajerani was supported by the Swedish Research Council, 2016-00529. The work of Y. Ji and S. Lafortune was supported in part by the U.S. National Science Foundation under Grant CNS-1421122 and Grant CNS-1738103. Recommended by Associate Editor C. Seatzu. (Corresponding author: Sahar Mohajerani.)

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: mohajera@chalmers.se; jiyiding@umich.edu; stephane@umich.edu).

Digital Object Identifier 10.1109/TAC.2019.2946165

in some other system models, like infinite state systems [6], modular systems [24], and Petri nets [42], [43]. Opacity under a special observer called Orwellian observer is discussed in [30], and opacity under powerful attackers is studied in [14]. A more recent work [51] investigates opacity for networked supervisory control systems. Furthermore, some works investigate opacity in stochastic settings, e.g., [1], [7], [21], [45]. Specifically, Yin *et al.* [52] present a novel approach to tackle infinite-step and K-step opacity in stochastic DES. The survey paper [16] summarizes some recent results on opacity in DES.

When opacity does not hold, it is natural to study its enforcement [10]. One popular approach is supervisory control [8], [9], [35], [41], [48], where some behaviors of the system are disabled before they reveal the secrets. Another widely applied method is sensor activation [5], [50], [53], where the observability of events is dynamically changed.

Recently, a new enforcement method called insertion function has been proposed in [46], which inserts fictitious events into the output of the system to obfuscate the intruder. The authors of [18] extended the method to study opacity enforcement under the assumption that the intruder may or may not know the implementation of insertion functions, while Ji *et al.* [19] discussed opacity enforcement by insertion functions under quantitative constraints. As a following work, Wu *et al.* [47] investigate a more general method called edit functions, which manipulate the output of the system by either inserting or erasing events. Then, Ji *et al.* [17], [20] consider the case when the edit function's implementation is known to the intruder. As a summary and extension, Ji *et al.* [20] characterize opacity enforcement by edit functions as a three-player game and proposes a novel information structure called three-player observer (TPO) to embed edit functions. A special TPO called the all edit structure (AES) is also introduced in [20] to characterize the edit constraints.

In this article, we elaborate the method in [20] to study opacity enforcement in a modular setting. Our motivation is as follows. To generate a TPO, the observer of the system needs to be calculated, which is potentially costly in computation. Furthermore, modern engineering systems usually contain multiple components that are synchronized and subject to malicious inference. In this sense, if we are to apply edit functions to enforce opacity, heavy computation is involved both from determining individual systems and synchronizing them, which may be potentially cumbersome.

To alleviate this issue, this article applies a compositional and abstraction-based method to reduce the size of the modular system before calculating the AES. *Bisimulation* and *observation equivalence* [25] are well-known methods to abstract the state space of an automaton, while they do not preserve opacity properties in general. As a variant, Zhang *et al.* [54] propose several innovative concepts termed opacity-preserving (bi)simulation relations to reduce the state space of the system in opacity verification. A compositional visible bisimulation equivalence method is discussed in [31] for abstraction-based opacity verification.

For abstraction, we introduce *opaque observation equivalence* and *opaque bisimulation*, which consider the secrecy status of states when merging them. In our framework, each individual system is abstracted using opaque observation equivalence. After that, the observer is calculated. Since abstraction reduces the size of the state space, the computational complexity of calculating the observer is lowered potentially. Next, opaque bisimulation is employed to the observer of each abstracted individual system, resulting in the smallest possible automaton for future discussion.

We further leverage some results from supervisor reduction and modular supervisory control theory to reduce the complexity of supervisor synthesis. There is a rich literature on both topics (see, e.g., [23], [28], [37], [39], and [40]). The main idea is to convert the construction of the monolithic AES to a modular supervisory control problem. Specifically, we first transfer each individual TPO (without considering edit constraints) to its automaton form and view the set of interacting automata as the “plant” to be controlled. Then, we put the edit constraint as the specification, also in an automaton form. Afterward, we perform modular supervisory control to synthesize a least restrictive and nonblocking modular supervisor. It is shown that all the traces accepted by the supervisor represent valid edit decisions contained in the monolithic AES. Compared with the conventional monolithic approach for supervisor synthesis [4], our compositional approach is more efficient in computation.

The presentation of this article is organized as follows. Section II gives a brief background introduction about the system model, supervisory control theory, and edit functions. The general idea of this article is presented in Section III as a flowchart. Section IV explains the abstraction methods and synchronization of TPOs. Next, Section V transforms the calculation of the monolithic AES to the calculation of a modular supervisor. Finally, some concluding remarks are given in Section VI.

A preliminary and partial version of this article appears in [26]. The current article improves [26] in the sense that [26] only considers abstraction methods to synthesize edit functions in a monolithic setting, while this article also takes synchronous composition into consideration, and the edit functions are synthesized by a modular approach.

II. MODELING FORMALISM AND BACKGROUND

A. Events, Automata, and Their Composition

In this article, we consider DESs modeled as deterministic or nondeterministic automata.

Definition 1: A (nondeterministic) finite-state automaton is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$, where Σ is a finite set of events, Q is a finite set of states, $\rightarrow \subseteq Q \times \Sigma \times Q$ is the *state transition relation*, and $Q^0 \subseteq Q$ is the set of *initial states*. G is *deterministic* if $|Q^0| = 1$ and if $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies that $y_1 = y_2$.

When state marking is considered, the above definition is extended to $G = \langle \Sigma, Q, \rightarrow, Q^0, Q^m \rangle$, where $Q^m \subseteq Q$ is the set of *marked states*. In this article, we identify marked states using gray shading in the figures.

We assume that the system is partially observed; thus, the concepts of *observable* and *unobservable* events are introduced. Since the exact identity of unobservable events is irrelevant in our later discussion of opacity, they are uniformly represented by a special event τ . The event τ is never included in the alphabet Σ , unless explicitly mentioned. For this reason, $\Sigma_\tau = \Sigma \cup \{\tau\}$ is used to represent the whole set of observable and unobservable events. Hereafter, nondeterministic automata may contain transitions labeled by τ , while *deterministic* automata *never* contain τ transitions. Moreover, $P_\tau : \Sigma_\tau^* \rightarrow \Sigma^*$ is the *projection* that removes from strings in Σ_τ^* all the τ events.

When automata are brought together to interact, lock-step synchronization in the style of [15] is used.

Definition 2: Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^0, Q_1^m \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^0, Q_2^m \rangle$ be two nondeterministic automata. The *synchronous composition* of G_1 and G_2 is defined as

$$G_1 \parallel G_2 := \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow, Q_1^0 \times Q_2^0, Q_1^m \times Q_2^m \rangle \quad (1)$$

where

$$\begin{aligned} (x_1, x_2) &\xrightarrow{\sigma} (y_1, y_2) && \text{if } \sigma \in (\Sigma_1 \cap \Sigma_2) \\ &&& x_1 \xrightarrow{\sigma_1} y_1, \text{ and } x_2 \xrightarrow{\sigma_2} y_2 \\ (x_1, x_2) &\xrightarrow{\sigma} (y_1, x_2) && \text{if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\} \\ &&& \text{and } x_1 \xrightarrow{\sigma_1} y_1 \\ (x_1, x_2) &\xrightarrow{\sigma} (x_1, y_2) && \text{if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\} \\ &&& \text{and } x_2 \xrightarrow{\sigma_2} y_2. \end{aligned}$$

Importantly, synchronous composition only imposes lock-step synchronization on common events from Σ_1 and Σ_2 .

The transition relation of an automaton G is written in infix notation $x \xrightarrow{\sigma} y$, and it is extended to strings in Σ_τ^* by letting $x \xrightarrow{\epsilon} x$ for all $x \in Q$, and $x \xrightarrow{t\sigma} z$ if $x \xrightarrow{t} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{t}$ means that $x \xrightarrow{t} y$ for some $y \in Q$, and $x \rightarrow y$ means that $x \xrightarrow{t} y$ for some $t \in \Sigma_\tau^*$. These notations also apply to state sets, where $X \xrightarrow{t} Y$ for $X, Y \subseteq Q$ means that $x \xrightarrow{t} y$ for some $x \in X$ and $y \in Y$, and to automata, where $G \xrightarrow{t}$ means that $Q^0 \xrightarrow{t}$ (t is defined in G) and $G \xrightarrow{t} x$ means $Q^0 \xrightarrow{t} x$.

For brevity, $p \xrightarrow{s} q$ for $s \in \Sigma_\tau^*$ represents the existence of a string $t \in \Sigma_\tau^*$ such that $P_\tau(t) = s$ and $p \xrightarrow{t} q$. Thus, $q \xrightarrow{u} p$ for $u \in \Sigma_\tau^*$ means a path containing exactly the events in u , while $q \xrightarrow{u} p$ for $u \in \Sigma^*$ means existence of a path between p and q with an arbitrary number of τ events between the observable events in u . Similarly, $p \xrightarrow{\tau} q$ means the existence of a string $t \in \{\tau\}^*$ such that $p \xrightarrow{t} q$.

The language of an automaton G is defined as $\mathcal{L}(G) = \{s \in \Sigma^* \mid G \xrightarrow{s}\}$, and the language generated by G from $q \in Q$ is $\mathcal{L}(G, q) = \{s \in \Sigma^* \mid q \xrightarrow{s}\}$; thus, we do not include event τ in the language of an automaton. Moreover, we also introduce projections P_i for $i = 1, 2$, which are $P_i : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$ for $i = 1, 2$.

For a nondeterministic automaton $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$, the set of *unobservably reached states* of $B \in 2^Q$, is $UR(B) = \bigcup \{C \subseteq Q \mid B \xrightarrow{\tau} C\}$. Its *observer* $\text{det}(G) = \langle \Sigma, X_{\text{obs}}, \rightarrow_{\text{obs}}, X_{\text{obs}}^0 \rangle$ is a deterministic automaton, where $X_{\text{obs}}^0 = UR(Q^0)$ and $X_{\text{obs}} \subseteq 2^Q$, and $X \xrightarrow{\sigma}_{\text{obs}} Y$, where $X, Y \in X_{\text{obs}}$, if and only if $Y = \bigcup \{UR(y) \mid x \xrightarrow{\sigma} y \text{ for some } x \in X \text{ and } y \in Q\}$. By convention, only reachable states from X_{obs}^0 under \rightarrow_{obs} are considered in this article. We also refer to the observer as the (*current-state*) *estimator* of the system, while an observer state is referred to as (*current-state*) *estimate*.

A common automaton operation is the *quotient* modulo, which is an equivalence relation on sets of states.

Definition 3: Let Z be a set. A relation $\sim \subseteq Z \times Z$ is called an *equivalence relation* on Z if it is reflexive, symmetric, and transitive. Given an equivalence relation \sim on Z , the *equivalence class* of $z \in Z$ is $[z] = \{z' \in Z \mid z \sim z'\}$, and $\tilde{Z} = \{[z] \mid z \in Z\}$ is the set of all equivalence classes modulo \sim .

Definition 4: Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *quotient automaton* of G modulo \sim is $\tilde{G} = \langle \Sigma, \tilde{Q}, \rightarrow/\sim, \tilde{Q}^0 \rangle$, where $\rightarrow/\sim = \{([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y\}$ and $\tilde{Q}^0 = \{[x^0] \mid x^0 \in Q^0\}$.

In order to compare automata structurally, we say that an automaton is a *subautomaton* of another automaton if all states and transitions in the first automaton are contained in the second one. Formally, we have the following definition.

Definition 5: Let $G_1 = \langle \Sigma_\tau, Q_1, \rightarrow_1, Q_1^0, Q_1^m \rangle$ and $G_2 = \langle \Sigma_\tau, Q_2, \rightarrow_2, Q_2^0, Q_2^m \rangle$ be two automata. G_1 is a *subautomaton* of G_2 , denoted by $G_1 \sqsubseteq G_2$, if $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, $Q_1^0 \subseteq Q_2^0$, and $Q_1^m \subseteq Q_2^m$.

B. Supervisory Control Theory

Considering plant G and specification K , *supervisory control theory* provides a method to synthesize a supervisor to restrict the behavior of the plant such that the given specification is always fulfilled. The supervisor S is a function defined from the language of the system G to the set of events; formally, $S : \mathcal{L}(G) \rightarrow 2^\Sigma$. We also partition the set of events as *uncontrollable events* and *controllable events*, i.e., $\Sigma = \Sigma_{uc} \cup \Sigma_c$, where uncontrollable events cannot be disabled by the supervisor. In the figures, the uncontrollable events are marked by an exclamation mark (!). The readers may refer to [4] for the main results of monolithic supervisory control under full observation. Here, we focus on concepts and definitions relevant to this article, and the synthesis procedure in this article is done on deterministic automata. Two requirements for the supervisor are *controllability* and *nonblockingness*, where controllability captures *safety* in the presence of uncontrollable events and nonblockingness focuses on *liveness* of the system.

Definition 6 (see[4]): Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^0, Q_G^m \rangle$ and $K = \langle \Sigma, Q_K, \rightarrow_K, Q_K^0, Q_K^m \rangle$ be two deterministic automata such that $K \sqsubseteq G$. K is *controllable* w.r.t. G if, for all states $x \in Q_K$ and $y \in Q_G$ and for every uncontrollable event $v \in \Sigma_{uc}$ such that $x \xrightarrow{v}_G y$, it also holds that $x \xrightarrow{v}_K y$.

Definition 7 (see[4]): Let G be a deterministic automaton. A state x is called *reachable* in G if $G \rightarrow x$, and *coreachable* in G if $x \rightarrow Q^m$. The automaton G is called *reachable* or *coreachable* if every state in G has this property. G is called *nonblocking* if every reachable state is coreachable.

The upper bound of controllable and nonblocking subautomata is again controllable and nonblocking. This implies the existence of a least restrictive subautomaton of the original system, which is achieved by the maximally permissive and nonblocking supervisor.

Definition 8: Let G be an automaton; the supremal controllable and nonblocking subautomaton of G is called the supremal supervisor, denoted by $\text{sup}\mathcal{C}(G)$, where for all controllable and nonblocking automata K w.r.t. G , $K \sqsubseteq \text{sup}\mathcal{C}(G)$.

Synthesis of $\text{sup}\mathcal{C}(G)$ is done by iteratively removing blocking and uncontrollable states, until a fixed point is reached, and restricting the final automaton to the remaining states and their associated transitions; for more details, see [4], [13], and [44].

In this article, we assume that the modular system has a set of interacting components $\{G_1, \dots, G_n\}$, and there is also a set of supervisors in a modular structure, i.e., $\mathcal{S} = \{S_1, \dots, S_n\}$. Here, supervisor S_i is responsible for controlling G_i . The set of modular supervisors may be synchronized as $\parallel_{i=1}^n S_i$.

C. Opacity and Edit Functions

In this article, we suppose system G has certain secret information, which is characterized by the set of states. Thus, the state space is partitioned into two disjoint subsets: $Q = Q^S \cup Q^{NS}$, where Q^S is the set of *secret states* capturing the secrets of the system, while Q^{NS} is the set of *nonsecret states*. When the system \mathcal{G} is modular, $\mathcal{G} = \{G_1, \dots, G_2\}$, the set of secret states of the system, Q_S , is $Q^S = \{(x_1, \dots, x_n) \mid \exists x_i \in Q_i^S\}$.

Suppose there is an external intruder modeled as the observer of the system, which intends to infer the secrets of the system from its observation. Then, a system is called *opaque* if the intruder is unable to determine unambiguously if the system has entered a secret state or not. Different notions of opacity have been introduced in the literature, and we focus on current-state opacity in this article.

Definition 9: A nondeterministic automaton G with a set of secret states Q^S is *current-state opaque* w.r.t. Q^S if $(\forall s \in \mathcal{L}(G, q^0) : Q^0 \xrightarrow{s} Q^S)$, then $[Q^0 \xrightarrow{s} Q^{NS}]$.

The system is current-state opaque if for any string reaching a secret state, there is string with the same sequence of observable events reaching a nonsecret state. It is known that current-state opacity can be verified by building the standard observer automaton.

Theorem 1: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with set of secret states Q^S . Let $\text{det}(G) = \langle \Sigma, X_{\text{obs}}, \rightarrow_{\text{obs}}, X_{\text{obs}}^0 \rangle$ be the current-state estimator of G . Then,

G is current-state opaque w.r.t. Q^S if and only if $[\det(G) \xrightarrow{s} X$ implies that $X \not\subseteq Q^S]$.

If all states violating current-state opacity are removed from the observer $\det(G)$, then the accessible part of the remaining structure is called the *desired observer*, denoted by $\det_d(G) = \langle \Sigma, X_{\text{obsd}}, \rightarrow_{\text{obsd}}, X_{\text{obsd}}^0 \rangle$. The language generated by the desired observer is referred to as the *safe language*, $L_{\text{safe}} = \mathcal{L}(\det_d(G))$. Accordingly, we also define the *unsafe language*, $L_{\text{unsafe}} = \mathcal{L}(G) \setminus L_{\text{safe}}$.

If a system is not current state opaque, then an interface-based approach called *edit function* [20], [47] may be applied to enforce it. An edit function may insert events into the output of the system or erase events from the output of the system. It is assumed that the intruder fails to distinguish between an inserted event and its genuine counterpart. Let $\Sigma^r = \{\sigma \rightarrow \varepsilon : \sigma \in \Sigma\}$ be the set of “event erasure” events.

Definition 10: A *deterministic edit function* is defined as $f_e : \Sigma^* \times \Sigma \rightarrow \Sigma^*$. Given $s \in \mathcal{L}(G)$, $\sigma \in \Sigma$,

$$f_e(s, \sigma) = \begin{cases} s_I \sigma, & \text{if } s_I \text{ is inserted before } \sigma \\ \varepsilon, & \text{nothing is inserted and } \sigma \text{ is erased} \\ s_I, & \text{if } s_I \text{ is inserted and } \sigma \text{ is erased.} \end{cases}$$

With an abuse of notation, we also define a string-based edit function \hat{f}_e recursively as $\hat{f}_e(\varepsilon) = \varepsilon$, $\hat{f}_e(s\sigma) = \hat{f}_e(s)f_e(s, \sigma)$ for $s \in \Sigma^*$ and $\sigma \in \Sigma$. In the following, to ease the notational burden, we will drop the “” in \hat{f}_e , and it will be clear from the argument(s) of f_e which function we are referring to (incremental single-event one or string-based one).

Two notions termed *public safety* and *private safety* were defined in [20] to characterize the behavior of edit functions. In this article, we consider private safety alone under the assumption that the intruder does not know about the implementation of an edit function.

Definition 11 (Private safety): Given G and its observer $\det(G)$, an edit function f_e is privately safe if $\forall s \in \mathcal{L}(\det(G))$, $f_e(s) \in L_{\text{safe}}$.

Recently, a three-player game structure called *TPO* w.r.t. the system has been defined in [20] to embed edit functions. For the sake of completeness, we recall this definition (more details are available in [20]).

Definition 12 (TPO): Given a system G with its observer $\det(G)$ and desired observer $\det_d(G)$, let $I \subseteq X_{\text{obsd}} \times X_{\text{obs}}$ be the set of information states. A TPO w.r.t. G is a tuple of the form $T = (Q_Y, Q_Z, Q_W, \Sigma, \Sigma^r, \Theta, \rightarrow_{yz}, \rightarrow_{zz}, \rightarrow_{zw}, \rightarrow_{wy}, y_0)$, where, we have the following.

- 1) $Q_Y \subseteq I$ is the set of Y states.
- 2) $Q_Z \subseteq I \times \Sigma$ is the set of Z states. Let $I(z)$ and $E(z)$ denote the information state component and observable event component of a Z state z , respectively, so that $z = (I(z), E(z))$.
- 3) $Q_W \subseteq I \times (\Sigma \cup \Sigma^r)$ is the set of W states. Let $I(w)$ and $A(w)$ denote the information state component and action component of a W state w , respectively, so that $w = (I(w), A(w))$.
- 4) Σ is the set of observable events.
- 5) Σ^r is the set of event-erasure events.

- 6) $\Theta \subseteq \Sigma \cup \{\varepsilon\} \cup \Sigma^r$ is the set of edit decisions at Z states.
 - a) $\rightarrow_{yz} : Q_Y \times \Sigma \times Q_Z$ is the transition function from Y states to Z states. For $y = (x_d, x_f) \in Q_Y$, $e_o \in \Sigma$, we have $y \xrightarrow{e_o}_{yz} z \Rightarrow [x_f \xrightarrow{e_o}_{\text{obs}}] \wedge [I(z) = y] \wedge [E(z) = e_o]$.
 - b) $\rightarrow_{zz} : Q_Z \times \Theta \times Q_Z$ is the transition function from Z states to Z states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$, we have $z \xrightarrow{\theta}_{zz} z' \Rightarrow [\theta \in \Sigma] \wedge [I(z') = (x'_d, x'_f)] \wedge [x_d \xrightarrow{\theta}_{\det_d} x'_d] \wedge [E(z') = e_o]$.
 - c) $\rightarrow_{zw1} : Q_Z \times \Theta \times Q_W$ is the ε insertion transition function from Z states to W states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$ we have $z \xrightarrow{\theta}_{zw1} w \Rightarrow [\theta = \varepsilon] \wedge [I(w) = I(z)] \wedge [A(w) = e_o] \wedge [x_d \xrightarrow{e_o}_{\det_d}] \wedge [x_f \xrightarrow{e_o}_{\text{obs}}]$.
 - d) $\rightarrow_{zw2} : Q_Z \times \Theta \times Q_W$ is the event erasure transition function from Z states to W states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$, we have $z \xrightarrow{\theta}_{zw2} w \Rightarrow [\theta = e_o \rightarrow \varepsilon] \wedge [I(w) = I(z)] \wedge [A(w) = e_o \rightarrow \varepsilon] \wedge [x_f \xrightarrow{e_o}_{\text{obs}}]$.
 - e) $\rightarrow_{wy1} : Q_W \times \Sigma \times Q_Y$ is the transition function from W states whose action component is in Σ to Y states. For $w = ((x_d, x_f), e_o) \in Q_W$, we have $w \xrightarrow{e_o}_{wy1} y \Rightarrow [y = (x'_d, x'_f)] \wedge [x_d \xrightarrow{e_o}_{\det_d} x'_d] \wedge [x_f \xrightarrow{e_o}_{\text{obs}} x'_f]$.
 - f) $\rightarrow_{wy2} : Q_W \times \Sigma \times Q_Y$ is the transition function from W states whose action component is in Σ^r to Y states. For $w = ((x_d, x_f), e_o \rightarrow \varepsilon) \in Q_W$, we have $w \xrightarrow{e_o}_{wy2} y \Rightarrow [y = (x_d, x'_f)] \wedge [x_f \xrightarrow{e_o}_{\text{obs}} x'_f]$.
- 7) $y_0 = (x_{\text{obsd},0}, x_{\text{obs},0}) \in Q_Y$ is the initial state of T , where $x_{\text{obsd},0}$ and $x_{\text{obs},0}$ are initial states of $\det_d(G)$ and $\det(G)$, respectively.

In general, a TPO characterizes a game between a dummy player, the edit function, and the environment (system). The state space of a TPO is partitioned as: Q_Y states (Y states), where the dummy player plays; Q_Z states (Z states), where the edit function plays; and Q_W states (W states), where the environment plays. A Y state contains the intruder’s estimate (left component) as well as the system’s true state estimate (right component). A \rightarrow_{yz} transition is defined out of a Y state, indicating that an observable event may occur and thus is received by the edit function. Then, the TPO transits to a Z state, and the turn of the game is passed to the edit function. Notice that the observable event does not really occur, and this dummy player is only introduced to help determine the decisions of edit functions.

At a Z state, the edit function may choose to insert certain events (including ε) or erase its last observed event. If a non- ε event is inserted, a \rightarrow_{zz} transition leads the TPO to another Z state, which means the edit function still has the turn to insert more events until it decides to stop insertion by inserting ε or by erasing the last observed event. There may be multiple transitions defined out of a Z state, i.e., multiple edit decisions; we write $\Theta(z)$ to denote the set of edit decisions defined at $z \in Q_Z$ in a TPO.

If the edit function inserts nothing (respectively, erases the event it receives from the dummy player), then a \rightarrow_{zw1} [respectively (\rightarrow_{zw2})] transition is defined, and the TPO is at a W state. Then, the environment plays by letting the observable event executed from its preceding Y state occur. Correspondingly, there are also two types of \rightarrow_{wy} transitions, where \rightarrow_{wy1} indicates that the executed observable event will be observed by the intruder, while \rightarrow_{wy2} indicates that the executed observable event will not be observed by the intruder, since it has been erased by the edit function.

When the three players take turns to play, the components of each player's states also get updated. From Definition 12, a \rightarrow_{yz} transition does not change the state estimates for the intruder or the system, since the player at Y states is dummy and the observable events from Y states do not really occur. With a \rightarrow_{zz} transition, only x_d is updated, since x_d is the estimate of the intruder and event insertion only alters the observation of the intruder. For \rightarrow_{zw} transitions, we only require the observable event to be defined at x_d or x_f . Finally, a \rightarrow_{wy1} transition updates both x_d and x_f , while a \rightarrow_{wy2} transition only updates x_f as the intruder does not observe the erased event.

To characterize the information flow in a TPO, the notion of *run* is defined in [20].

Definition 13 (Run): In a TPO T , a run is defined as $\omega = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \dots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \dots \xrightarrow{\theta_1^{m_1-1}} z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \dots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \dots \xrightarrow{\theta_n^{m_n-1}} z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, where y_0 is the initial state of T , $e_i \in \Sigma$, $\theta_i^j \in \Theta(z_i^j)$, $\forall 0 \leq i \leq n, 1 \leq j \leq m_i$ and $n \in \mathbb{N}, m_i \in \mathbb{N}^+$.

We let Ω_T be the set of all runs in a TPO T . For simplicity, similar notations as for automata are defined for TPOs, and thus, $T \xrightarrow{\omega} x$ denotes the existence of a run in a TPO. We also review the concepts of *string generated by a run* and *edit projection* defined in [20].

Definition 14 (String generated by a run): Given a run ω as in Definition 13, the string generated by ω is defined as $l(\omega) = \theta_0^1 \theta_0^2 \dots \theta_0^{m_0-1} \theta_0^{m_0} e_0 \theta_1^1 \dots \theta_1^{m_1-1} \theta_1^{m_1} e_1 \dots e_{n-1} \theta_n^1 \dots \theta_n^{m_n-1} \theta_n^{m_n} e_n$, where $\forall i \leq n, \theta_i^{m_i} e_i = \varepsilon$ if $\theta_i^{m_i} = e_i \rightarrow \varepsilon$.

Definition 15 (Edit projection): Given TPO T and run ω_T as in Definition 13, the edit projection $P_e : \Omega \rightarrow \mathcal{L}(G)$ is defined such that $P_e(\omega_T) = e_0 e_1 \dots e_n$.

In a TPO, $y \in Q_Y$ is a *terminating state* if $\nexists e_o \in \Sigma$, s.t. $y \xrightarrow{e_o}$. And $w \in Q_W$ is a *deadlocking state* if $\nexists e_o \in \Sigma$, s.t. $w \xrightarrow{e_o} y$. Also, $z \in Q_Z$ is a *deadlocking state* if $\nexists \theta \in \Theta$, s.t. $z \xrightarrow{\theta} z'$ or $z \xrightarrow{\theta} w$. We call a TPO T *complete* [20] if there are no deadlocking W or Z states in T and $\forall s \in \mathcal{L}(G), \exists \omega \in \Omega_T$, s.t. $P_e(\omega) = s$.

Definition 16 (Edit function embedded in TPO): Given a TPO T , a deterministic edit function f_e is embedded in T if $\forall s \in \mathcal{L}(G), \exists \omega \in \Omega_T$, s.t. $P_e(\omega) = s$ and $l(\omega) = f_e(s)$.

Next, we construct the *largest TPO* in the sense that all the other TPOs are subautomata of it. Such a notion is well defined by considering *all* admissible transitions at *every* state of the TPO, according to the respective conditions in Definition 12.

Edit functions are designed to erase genuine events or insert fictitious ones to mislead the intruder. In theory, it is possible to

design an edit function that erases all the events of the system, although this is not desirable. To avoid this situation, usually, the user provides some constraints on the edit functions. The constraint that is considered in this article is to limit the number of consecutive erasures.

Definition 17 (Edit constraint): The edit constraint, denoted by Φ , requires that the edit function should not make $n + 1$ consecutive erasures where $n \in \mathbb{N}$.

Finally, we define the AES [20] by considering the edit constraint. A synthesis procedure was also presented in [20] to construct the AES. Notice that the following definition is slightly different from the AES in the preliminary version of this work [26] since edit constraints are not considered in [26].

Definition 18 (AES): Given system G , observer $\text{det}(G)$, and desired estimator $\text{det}_d(G)$, the AES is defined to be the largest complete TPO w.r.t. G , which satisfies the edit constraint.

From results in [20], private safety is achievable when the AES is not empty by construction. Hereafter, we assume that the AES is nonempty in the following discussion; if it is empty, then opacity cannot be enforced by the mechanism of edit functions. It was also proven in [20] that *all* privately safe edit functions satisfying edit constraints are embedded in the AES. Formally speaking, the following result holds.

Theorem 2: Given a system G and its corresponding AES under edit constraint Φ , an edit function f_e is privately safe and satisfies Φ if and only if $f_e \in \text{AES}$.

We end this section by briefly reviewing the pruning process discussed in [20] to construct the AES. The presence of edit constraints may preclude some undesired states from the AES, thus leaving some states without outgoing transitions, i.e., “deadlock” Z or W states. Those states reflect the inability of the edit function to issue a valid edit decision (for insertion or erasure) while still maintaining opacity for all possible future behaviors and, thus, should be removed in the pruning process. Moreover, Y states that have transitions to a deadlock Z state need to be pruned as well, since Y states are the states where the system issues an output event and the edit function is not allowed to prevent their occurrence.

The construction of the AES may also be interpreted as the calculation of a supervisor, where the “plant” is the largest TPO in terms of subautomaton, including all potentially feasible edit decisions without considering edit constraints. The Y states are considered as marked states. The events labeling transitions from Y states to Z states and from W states to Y states are considered as uncontrollable, while the events labeling transitions from Z states to Z states and Z states to W states are viewed as controllable. We also define the proper specification by considering edit constraints, deleting states that violate them, and taking the trim of the resulting structure. The goal is to calculate the least restrictive, controllable, and nonblocking supervisor based on the plant and this specification. Similar processes of pruning game structures akin to TPOs were discussed in prior work, e.g., [18], [20], [46]. We will leverage this approach in the following discussion, but in the framework of *modular supervisory control*.

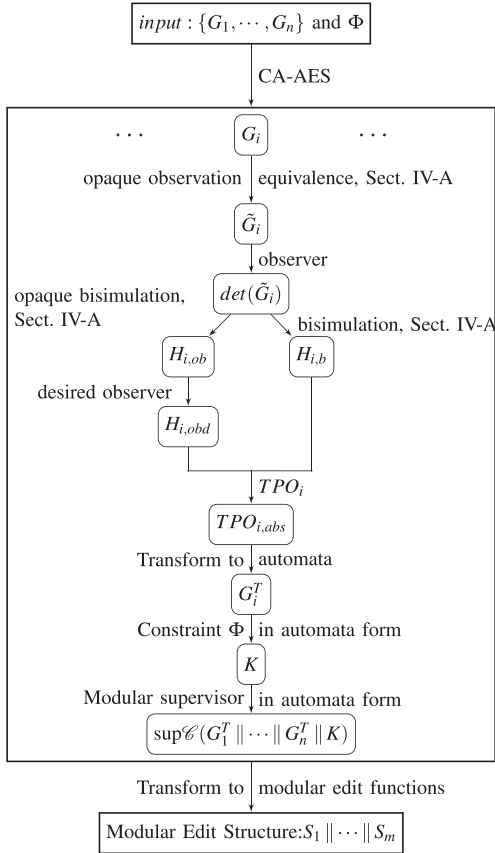


Fig. 1. Steps of Algorithm CA-AES.

III. COMPOSITIONAL ABSTRACTION-BASED METHODOLOGY

This section presents our novel compositional and abstraction-based methodology for synthesizing modular form edit functions based on individual TPOs after abstracting the original system. For simplicity, we call this methodology the composition abstraction all edit structure (CA-AES) algorithm hereafter. The input of the algorithm is a set of nondeterministic automata, $\mathcal{G} = \{G_1, \dots, G_n\}$, and the output is a modular representation of edit functions, which is called *modular edit structure*. The algorithm is summarized in Fig. 1, and its steps are as follows. We will explain how to interpret the modular representation of edit functions later.

- (i) The algorithm first abstracts each individual automaton, G_i , using *opaque observation equivalence*. This results in \tilde{G}_i , which has fewer states and transitions compared to the original automaton.
- (ii) Next, we abstract the observer of \tilde{G}_i , i.e., $\det(\tilde{G}_i)$, by *opaque bisimulation* and *bisimulation*, resulting in two abstracted deterministic automata $H_{i,ob}$ and $H_{i,b}$.
- (iii) Then we calculate the abstracted desired observer of G_i from $H_{i,ob}$, which is denoted by $H_{i,obd}$.
- (iv) Afterward, the largest (abstracted) TPO of each individual component G_i is calculated from the abstracted observer $H_{i,b}$ and the abstracted desired observer $H_{i,obd}$, and it is denoted by TPO_i .

- (v) The final step is to calculate a modular nonblocking and controllable supervisor, then obtain a set of modular edit functions. This is done by transforming the largest TPOs and the edit constraint to a set of automata, i.e., G_i^T and K , respectively. This modular approach is in contrast to calculating monolithic edit functions embedded in the monolithic AES [20].

Specifically, in step (iv), each abstracted TPO w.r.t. the corresponding individual system together with the constraint Φ are transformed to a set of interacting automata. Then, in step (v), a modular supremal controllable and nonblocking supervisor is calculated, thereby fulfilling the edit constraint in the composed structure. Consequently, the modular edit structure is itself a modular supervisor. Regarding step (v), it is possible to leverage existing efficient algorithms on modular supervisory control to calculate the modular edit structure.

In the monolithic approach of calculating the AES, individual systems G_1 through G_n are synchronized first, and then, the observer of the synchronized system is built. Since the computational complexity of calculating the observer is exponential, synchronizing individual components before building the observer significantly increases the complexity, which may be $2^{\prod_{i=1}^n |Q_i|}$ in the worst case, where $|Q_i|$ is the cardinality of Q_i . Moreover, constructing the AES is polynomial in terms of the state space of the observer, which may be potentially intractable when we deal with the synchronized system. In contrast, our compositional and abstraction-based approach reduces computational cost considerably both from abstracting individual systems and conducting computation in a modular way. However, as will be demonstrated later, some edit decisions may be omitted in the modular edit structure output by the CA-AES algorithm.

The presented approach relies heavily on the use of TPOs. We present an example to better understand the structure of such observers.

Example 1: Consider the nondeterministic automaton G_1 with secret states set $Q_1^S = \{q_3\}$, shown in Fig. 2. To generate the TPO of G_1 , first the observer of G_1 needs to be built, which is shown as $\det(G_1)$ in Fig. 2. To generate the desired observer, the state $\{q_3\} \subseteq Q_1^S$ needs to be removed. The desired observer $\det_d(G_1)$ is shown in Fig. 2.

Then, we follow the procedures in [20] to build the TPO w.r.t. $\det(G_1)$ in Fig. 2 (labeled as T'_1). As is discussed, the game on the TPO is initiated from the Y state (q_0, q_0) , where the dummy player executes the observable event γ (the only event defined at q_0 in $\det(G_1)$). Then, the edit function takes the turn to play at the Z state (q_0, q_0, γ) , where it has two choices: insert nothing or erase γ . If γ is erased, then the W state $(q_0, q_0, \gamma \rightarrow \varepsilon)$ is reached, where the environment plays by executing γ . Then, the turn is passed back to the dummy player, and the rest of the structure is interpreted similarly.

The compositional abstraction-based approach is explained in more detail in the following sections. First, in Section IV, we discuss abstractions at the component level and synchronization of individual TPOs, formalizing steps (i)–(iv) of the CA-AES algorithm. Then, in Section V, we discuss the last step of the CA-AES algorithm.

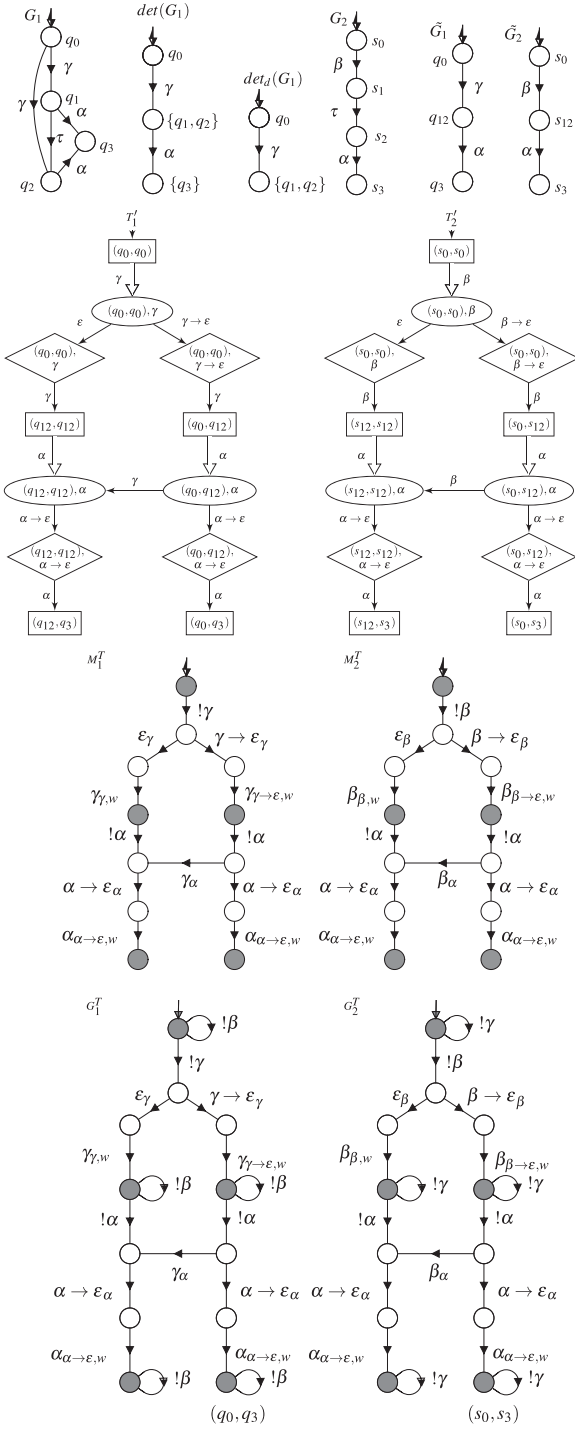


Fig. 2. System $\mathcal{G} = \{G_1, G_2\}$ and its abstraction $\{\tilde{G}_1, \tilde{G}_2\}$. The figure also shows the largest TPOs T_1^T and T_2^T of the abstracted components and their automata transformations, denoted by G_1^T and G_2^T . The uncontrollable events are marked by (!).

IV. SYNCHRONIZATION AND ABSTRACTION OPERATIONS

This section presents results on abstraction and composition that support steps (i)–(iv) of the CA-CAS algorithm. First, Section IV-A describes the methods to abstract nondeterministic automata and their observers. Next, Section IV-B describes the

process of transforming every individual TPO to an automaton form and shows that the automaton representation is a substructure (in the sense of subgraph) of the largest monolithic TPO.

A. Opaque Observation Equivalence

The first strategy used in the CA-AES algorithm to alleviate state-space explosion is abstraction of system components. This subsection contains a collection of abstraction methods that can be used to abstract nondeterministic automata and their observers such that the abstracted observers and the desired observers are bisimilar to their original counterparts. The abstraction methods are based on bisimulation and observation equivalence, which are computationally efficient and can be calculated in polynomial time [12]. We will prove in Theorem 5 that if we build the largest TPO based on the abstracted observer and the desired observer, we obtain the same runs and, consequently, the same edit functions as we do from the largest TPO based on the original observer and desired observer.

Bisimulation is a widely used notion of abstraction that merges states with the same future behavior.

Definition 19 (see[25]): Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton. An equivalence relation $\approx \subseteq Q \times Q$ is called a *bisimulation* on G , if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \approx x_2$: if $x_1 \xrightarrow{\sigma} y_1$ for some $\sigma \in \Sigma_\tau$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{\sigma} y_2$, and $y_1 \approx y_2$.

Bisimulation seeks to merge states with the same outgoing transitions to equivalent states *including unobservable events*, i.e., τ events. If the unobservable events are disregarded, a more general abstraction method called *weak bisimulation* or *observation equivalence* naturally comes [25].

Definition 20: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton. An equivalence relation $\sim \subseteq Q \times Q$ is called an *observation equivalence* on G , if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{s} y_1$ for some $s \in \Sigma^*$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{s} y_2$, and $y_1 \sim y_2$.

In order to use observation equivalence for abstraction in the opacity setting, the set of secret states needs to be taken into account. In the following discussion, a restricted version of observation equivalence called *opaque observation equivalence* is employed. This notion was first defined in [27] in the context of verifying opacity.

Definition 21: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with set of secret states $Q^S \subseteq Q$ and set of nonsecret states $Q^{NS} = Q \setminus Q^S$. An equivalence relation $\sim_o \subseteq Q \times Q$ is called an *opaque observation equivalence* on G with respect to Q^S , if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim_o x_2$.

- 1) If $x_1 \xrightarrow{s} y_1$ for some $s \in \Sigma^*$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{s} y_2$, and $y_1 \sim_o y_2$.
- 2) $x_1 \in Q^S$ if and only if $x_2 \in Q^S$.

We also wish to use bisimulation to abstract the observer of a nondeterministic system. Besides opaque observation equivalence, *opaque bisimulation* is also defined.

Definition 22: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with set of secret states $Q^S \subseteq Q$ and set of nonsecret states $Q^{NS} = Q \setminus Q^S$. Let $\det(G) =$

$\langle \Sigma, X_{\text{obs}}, \rightarrow_{\text{obs}}, X_{\text{obs}}^0 \rangle$ be the observer of G . An equivalence relation $\approx_o \subseteq X_{\text{obs}} \times X_{\text{obs}}$ is called an *opaque bisimulation equivalence* on $\det(G)$ with respects to Q^S , if the following holds for all $X_1, X_2 \in X_{\text{obs}}$ such that $X_1 \approx_o X_2$.

- 1) If $X_1 \xrightarrow{s} Y_1$ for some $s \in \Sigma^*$, then there exists $Y_2 \in X_{\text{obs}}$ such that $X_2 \xrightarrow{s} Y_2$, and $Y_1 \approx_o Y_2$.
- 2) $X_1 \subseteq Q^S$ if and only if $X_2 \subseteq Q^S$.

The first step of the CA-AES algorithm is to abstract the system using opaque observation equivalence. It has been shown in [32] that if two automata are bisimilar, then their observers are also bisimilar. In this article, this result is extended such that abstracting a nondeterministic automaton using opaque observation equivalence results in an observer and a desired observer, which are bisimilar to the observer and the desired observer of the original system, respectively.

Proposition 3: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with set of secret states $Q^S \subseteq Q$ and set of non-secret states $Q^{NS} = Q \setminus Q^S$. Let \sim_o be an opaque observation equivalence on G resulting in \tilde{G} and let \approx be a bisimulation. Let $\det_d(G)$ and $\det_d(\tilde{G})$ be the desired observer of G and \tilde{G} . Then, $\det(G) \approx \det(\tilde{G})$ and $\det_d(G) \approx \det_d(\tilde{G})$.

Proof: First, we prove that $\det(G) \approx \det(\tilde{G})$. To prove $\det(G) \approx \det(\tilde{G})$, it is enough to show that $\det(G) \xrightarrow{s} X$ if and only if $\det(\tilde{G}) \xrightarrow{s} \tilde{X}$, which implies language equivalence between $\det(G)$ and $\det(\tilde{G})$, since $\det(G)$ and $\det(\tilde{G})$ are deterministic. This can be shown by induction. Moreover, in the induction, we also show that $x \in X$ if and only if there exists $[x'] \in \tilde{X}$ such that $x \in [x']$. This is used for the second part of the proof, where we show $\det_d(G) \approx \det_d(\tilde{G})$.

It is shown by induction on $n \geq 0$ that $X^0 \xrightarrow{\sigma_1} X^1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} X^n$ in $\det(G)$ if and only if $\tilde{X}^0 \xrightarrow{\sigma_1} \tilde{X}^1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} \tilde{X}^n$ in $\det(\tilde{G})$ such that $x \in X^j$ if and only if $[x'] \in \tilde{X}^k$, where $x \in [x']$, for $1 \leq j \leq n$.

Base case: $n = 0$. Let X^0 be the initial state of $\det(G)$ and \tilde{X}^0 be the initial state of $\det(\tilde{G})$. It is shown that $x \in X^0$ if and only if there exists $[x'] \in \tilde{X}^0$ such that $x \in [x']$.

First, let $x \in X^0$. Then, based on $UR(x^0)$, it follows that there exists $x^0 \in Q^0$ such that $x^0 \xrightarrow{\tau} x$ in G . Since $G \sim_o \tilde{G}$ then based on Definition 21, there exists $[x'^0] \in \tilde{X}^0$ such that $[x'^0] \xrightarrow{\tau} [x']$ in \tilde{G} such that $x^0 \in [x'^0]$ and $x \in [x']$. Then, based on $UR(x^0)$ it follows that $[x'] \in \tilde{X}^0$.

Now, let $[x'] \in \tilde{X}^0$. Then, based on $UR(x^0)$, it follows that there exists $[x'^0] \in Q^0$ such that $[x'^0] \xrightarrow{\tau} [x']$ in \tilde{G} . Since $G \sim_o \tilde{G}$ then based on Definition 21, there exists $x^0 \in X^0$ such that $x^0 \xrightarrow{\tau} x$ in G such that $x^0 \in [x'^0]$ and $x \in [x']$. Then, based on $UR(x^0)$, it follows that $x \in X^0$.

Inductive step: Assume the claim holds for some $n \geq 0$, i.e., $X^0 \xrightarrow{\sigma_1 \sigma_2 \dots \sigma_n} X^n = X$ in $\det(G)$ if and only if $\tilde{X}^0 \xrightarrow{\sigma_1 \sigma_2 \dots \sigma_n} \tilde{X}^n = \tilde{X}$ in $\det(\tilde{G})$, such that $x \in X^k$ if and only if there exists $[x'] \in \tilde{X}^k$ such that $x \in [x']$ for all $0 \leq k < n$.¹ It must be shown that $X = X^n \xrightarrow{\sigma_{n+1}} Y$ in $\det(G)$ if and only if

$\tilde{X} = \tilde{X}^n \xrightarrow{\sigma_{n+1}} \tilde{Y}$ in $\det(\tilde{G})$ such that $x \in X$ if and only if there exists $[x'] \in \tilde{X}$ such that $x \in [x']$.

First, let $X = X^n \xrightarrow{\sigma_{n+1}} Y$ in $\det(G)$ and let $x \in X$. Then, based on $UR(x)$, it holds that $x = x^1 \xrightarrow{\tau} \dots \xrightarrow{\tau} x^r \xrightarrow{\sigma_{n+1}} y$ in G , where $x^j \in X$ for all $1 \leq j \leq r$ and $y \in Y$. Since $G \sim_o \tilde{G}$, it holds that $[x'] = [x^1] \xrightarrow{\tau} \dots \xrightarrow{\tau} [x^r] \xrightarrow{\sigma_{n+1}} [y']$ in \tilde{G} such that $x^j \in [x'^j]$ for all $1 \leq j \leq r$ and $y \in [y']$. Based on $UR(x)$ and inductive assumption, it holds that $\det(\tilde{G}) \xrightarrow{\sigma_1 \sigma_2 \dots \sigma_n} \tilde{X}^n = \tilde{X} \xrightarrow{\sigma_{n+1}} \tilde{Y}$ and $[x'] \in \tilde{X}$.

Now, let $\tilde{X} = \tilde{X}^n \xrightarrow{\sigma_{n+1}} \tilde{Y}$ in $\det(\tilde{G})$ and let $[x] \in \tilde{X}$. Then, based on $UR(x)$, it holds $[x] = [x^1] \xrightarrow{\tau} \dots \xrightarrow{\tau} [x^r] \xrightarrow{\sigma_{n+1}} [y]$ in \tilde{G} , where $[x^i] \in \tilde{X}$ for all $1 \leq i \leq r$ and $[y] \in \tilde{Y}$. Since $G \sim_o \tilde{G}$, it holds that $x' = x'^1 \xrightarrow{\tau} \dots \xrightarrow{\tau} x'^r \xrightarrow{\sigma_{n+1}} y'$ in G such that $x^i \in [x^i]$ for all $1 \leq i \leq r$ and $y' \in [y]$. Based on $UR(x)$ and inductive assumption, it holds that $\det(G) \xrightarrow{\sigma_1 \sigma_2 \dots \sigma_n} X^n = X \xrightarrow{\sigma_{n+1}} Y$ such that $x' \in X$.

Now, we need to show that $\det_d(G) \approx \det_d(\tilde{G})$. It was proven above that $\det(G) \approx \det(\tilde{G})$, which means $\det(G) \xrightarrow{s} X$ if and only if $\det(\tilde{G}) \xrightarrow{s} \tilde{X}$ and $x \in X$ if and only if $[x'] \in \tilde{X}$, where $x \in [x']$. Therefore, it is enough to show that $X \not\subseteq X_{\text{obsd}}$ if and only if $\tilde{X} \not\subseteq \tilde{X}_{\text{obsd}}$.

First, assume $X \subseteq Q^S$, which means for all $x \in X$, it holds that $x \in Q^S$ and $X \not\subseteq X_{\text{obsd}}$. Since for all $x \in X$, it holds that there exist $[x'] \in \tilde{X}$ such that $x \in [x']$, then based on Definition 21, it holds that $[x'] \in \tilde{Q}^S$. Thus, it can be concluded that for all $[x'] \in \tilde{X}$, it holds that $[x'] \in \tilde{Q}^S$. This means that $\tilde{X} \subseteq \tilde{Q}^S$, and consequently, $\tilde{X} \not\subseteq \tilde{X}_{\text{obsd}}$.

Now, assume $\tilde{X} \subseteq \tilde{Q}^S$, which means for all $[x'] \in \tilde{X}$, it holds that $[x'] \in \tilde{Q}^S$ and $\tilde{X} \not\subseteq \tilde{X}_{\text{obsd}}$. If $[x'] \in \tilde{Q}^S$, then for all $x \in [x']$, it holds that $x \in Q^S$. Moreover, it was shown above that $[x'] \in \tilde{X}$ if and only if $x \in X$, where $x \in [x']$. Thus, from $\tilde{X} \subseteq \tilde{Q}^S$, it follows that $X \subseteq Q^S$, which means that $X \not\subseteq X_{\text{obsd}}$.

Thus, it can be concluded that $\det_d(G) \approx \det_d(\tilde{G})$. \blacksquare

Opaque observation equivalence seeks to merge states of a nondeterministic automaton, which are “equivalent,” before constructing the observer. After calculating the observer, it is possible to further abstract the observer using opaque bisimulation. This guarantees that the smallest abstracted observer generates the same language as the original observer. In the following, Proposition 4 shows that if opaque bisimulation is used to abstract the observer, then the abstracted desired observer is also bisimilar to the original desired observer.

Proposition 4: Let $G = \langle \Sigma_\tau, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with set of secret states $Q^S \subseteq Q$ and set of nonsecret states $Q^{NS} = Q \setminus Q^S$. Let \approx_o be an opaque bisimulation on $\det(G)$ resulting in \tilde{G} . Let $\det_d(G)$ and H_d be the desired observers of $\det(G)$ and $\det(\tilde{G})$, respectively. Then, $\det_d(G) \approx H_d$, where \approx is a bisimulation relation.

Proof: Since $\det(G) \approx_o \tilde{G}$ based on Definition 22, it holds that $\det(G) \xrightarrow{s} X$ if and only if $\tilde{G} \xrightarrow{s} [X']$ and $X \in [X']$. Thus, it is enough show that $X \not\subseteq X_{\text{obs}, \det_d(G)}$ if and only if $[X'] \not\subseteq X_{\text{obs}, H_d}$, where $X \in [X']$.

¹Since the base case of the induction is proven for $n = 0$, $X^0 \xrightarrow{\sigma}$, the inductive step is considered true for $0 \leq k < n$.

First, assume $X \subseteq Q^S$, so $X \notin X_{\text{obs}, \det_d(G)}$. Then, since $X \in [X']$, based on Definition 22, it holds that for all $X' \in [X']$, $X' \subseteq Q^S$. This means $[X'] \subseteq Q^S$, and consequently, $[X'] \notin X_{\text{obs}, H_d}$.

Then, assume $[X'] \subseteq Q^S$, so $[X'] \notin X_{\text{obs}, H_d}$. Since $X \in [X']$, based on Definition 22, $X \subseteq Q^S$ holds, i.e., $X \notin X_{\text{obs}, \det_d(G)}$. ■

We now present the main results of this subsection.

Theorem 5: Let G be a nondeterministic automaton with secret states $Q^S \subseteq Q$ and nonsecret states $Q^{NS} = Q \setminus Q^S$. Let $\det(G)$ and $\det_d(G)$ be the observer and the desired observer of G , respectively. Let \sim_o be an opaque observation equivalence on G such that $\tilde{G} \sim_o G$. Let $H_{ob} \approx_o \det(\tilde{G})$ and $H_b \approx \det(\tilde{G})$, where \approx_o and \approx are opaque bisimulation and bisimulation, respectively. Let H_{obd} be the desired observer of H_{ob} . Let T be the largest TPO w.r.t. $\det(G)$ and $\det_d(G)$, also let T' be the largest TPO w.r.t. H_{obd} and H_b . Then, $T \xrightarrow{\omega} q$ if and only if $T' \xrightarrow{\omega} \tilde{q}$.

Formal proof of Theorem 5 can be found in [55]. Theorem 5 proves that the largest TPO obtained from the abstracted system (using opaque observation equivalence and opaque bisimulation) has the same set of runs with that obtained from the original system. This result is essential for the correctness of the CA-AES algorithm.

Remark 1: The abstractions in the worst-case scenario fail to merge any states. However, as pointed out in this article, the complexity of the abstraction methods is polynomial, while the complexity of calculating the observer is exponential in the number of states. Thus, if the abstraction results in merging even few states, it can potentially reduce the complexity of calculating the observer significantly. Therefore, it is worth applying the abstraction algorithm before calculating the observers.

Example 2: Consider the nondeterministic system $\mathcal{G} = \{G_1, G_2\}$, shown in Fig. 2, with secret states sets $Q_1^S = \{q_3\}$ and $Q_2^S = \{s_3\}$, where all the events are observable except event τ . In G_1 , states q_1 and q_2 are opaque observation equivalent as they both have the same secrecy status and equivalent states can be reached from both, $q_1 \xrightarrow{\alpha} q_3$ and $q_2 \xrightarrow{\alpha} q_3$, and $q_1 \xrightarrow{\tau} q_2$ and $q_2 \xrightarrow{\varepsilon} q_1$. Merging q_1 and q_2 results in the abstracted automaton \tilde{G}_1 shown in Fig. 2. Moreover, states s_1 and s_2 are also opaque observation equivalent, and merging them results in automaton \tilde{G}_2 shown Fig. 2. After abstracting the automata, the system becomes a deterministic system. Moreover, the observers as of \tilde{G}_1 and \tilde{G}_2 are bisimilar to $\det(G_1)$ and $\det(G_2)$, respectively. The same is also true for the desired observer of \tilde{G}_1 and \tilde{G}_2 . Fig. 2 shows the largest TPOs of \tilde{G}_1 and \tilde{G}_2 , respectively.

B. Synchronous Composition of TPOs

The second strategy used in the CA-AES algorithm to reduce computation complexity is synchronous composition of individual systems. In this article, the main advantage of our compositional approach is to build the largest TPO of each component individually, instead of synchronizing individual components and then building the largest monolithic TPO. Before synchronization, we first transfer each individual TPO to an automaton using Definition 23. Next, the individual automata are

transformed to a set of interacting automata based on Definition 24. It is shown in Theorem 8 that the set of modular TPOs form a subsystem of their monolithic counterpart, in the sense that some runs are omitted after synchronization. Before Theorem 8, Lemmas 6 and 7 establish that synchronization of individual observers (respectively, desired observers) is isomorphic to the observer (respectively, desired observers) of the synchronized system.

Definition 23: Let $T = \langle Q_Y, Q_Z, Q_W, \Sigma, \Sigma^\varepsilon, \Theta, \rightarrow_{yz}, \rightarrow_{zz}, \rightarrow_{zw}, \rightarrow_{wy}, y_0 \rangle$ be a TPO. Automaton $M^T = \langle \Sigma_{M^T}, Q, \rightarrow, Q^0, Q^m \rangle$ is the monolithic transformed deterministic automaton of T where we have the following.

- 1) $\Sigma_{M^T} = \Sigma \cup [\bigcup_{p \in Q_Z} \Theta_{E(p)}] \cup [\bigcup_{p \in Q_W} \Sigma_{A(p), w}]$.
- 2) $Q = Q_Y \cup Q_Z \cup Q_W$.
- 3) $\rightarrow = \{ (p, \alpha, q) \mid p \in Q_Y \wedge p \xrightarrow{\alpha}_{yz} q \} \cup \{ (p, \sigma, q) \mid p \in Q_Z \wedge \sigma = \alpha_{E(p)} \wedge p \xrightarrow{\alpha}_{zz, zw_1, zw_2} q \} \cup \{ (p, \sigma, q) \mid p \in Q_W \wedge \sigma = \alpha_{A(p), w} \wedge p \xrightarrow{\alpha}_{wy_1, wy_2} q \}$.
- 4) $Q^0 = y_0$.
- 5) $Q^m = Q_Y$.

The events labeling outgoing transitions mapped from original Y states in T , i.e., $\{ (p, \alpha, q) \mid p \in Q_Y \wedge p \xrightarrow{\alpha}_{yz} q \}$ and outgoing transitions mapped from original W states in T , $\{ (p, \sigma, q) \mid p \in Q_W \wedge \sigma = \alpha_{A(p), w} \wedge p \xrightarrow{\alpha}_{wy_1, wy_2} q \}$ are considered as uncontrollable while the other events are controllable.

In Definition 23, Σ_α represents that α is added to all the events of Σ . To transform a TPO to an automaton, each state of the TPO is considered as an automaton state, $Q_i = Q_Y^i \cup Q_Z^i \cup Q_W^i$ in Definition 23. Moreover, the information about the states needs to be considered to distinguish some transitions in the transformed automaton and to have a correct synchronization of TPOs, since the information about state types (Y, W, Z) is lost in the transformation. To this end, in the transformed automaton, the events labeling the transitions from z to z states, from z to w states and from w to y states, need to have the information of the predecessor states reflected in them. Thus, the events in the transformed automaton have the observable event components of Z states, $\bigcup_{p \in Q_Z^i} \Theta^i_{E(p)}$, and the action components of W states, $\bigcup_{p \in Q_W^i} \Sigma^i_{A(p), w}$. The initial state of the transformed automaton is the initial state of the TPO, and the marked states are the original Y states.

Example 3: Consider the two TPOs T'_1 and T'_2 shown in Fig. 2. To transform the TPOs T_1 and T_2 to their monolithic automata, renaming $\rho : \{ \alpha, \beta, \gamma, \varepsilon, \gamma \rightarrow \varepsilon, \gamma \rightarrow \varepsilon, w, \gamma_\alpha, \varepsilon_\beta, \beta \rightarrow \varepsilon_\beta, \beta_{\beta \rightarrow \varepsilon, w}, \beta_{\varepsilon, w}, \beta_\alpha, \alpha \rightarrow \varepsilon_\alpha, \alpha_{\alpha \rightarrow \varepsilon, w} \} \rightarrow \{ \alpha, \beta, \gamma, \varepsilon, \alpha \rightarrow \varepsilon, \beta \rightarrow \varepsilon, \gamma \rightarrow \varepsilon \}$ is introduced. Next, all Y states in T'_1 and T'_2 are considered as marked. Fig. 2 shows M^T_1 and M^T_2 , which are the monolithic transformed automata from T'_1 and T'_2 , respectively. All the events that come from transitions defined out of Y states and W states in T'_1 and T'_2 are considered as uncontrollable in G^T_1 and G^T_2 ; thus, α, β , and γ are uncontrollable.

This article describes the compositional approach for modular systems. In order to have a correct interaction between the transformed automata, the transformation of the TPOs needs to be done in the modular setting.

TABLE I
LINK BETWEEN THE EVENTS OF A TPO, ITS TRANSFORMED AUTOMATON,
AND THE CORRESPONDING RENAMING

TPO T	Automaton G^T	Renaming ρ
$y \xrightarrow{\alpha} yz z$	$y \xrightarrow{\alpha} z$	$\rho(\alpha) = \alpha$
$z \xrightarrow{\alpha} z z z', E(z) = e_o$	$z \xrightarrow{\alpha_{e_o}} z'$	$\rho(\alpha_{e_o}) = \alpha$
$z \xrightarrow{\alpha} z w_1 w, E(z) = e_o$	$z \xrightarrow{\alpha_{e_o}} w$	$\rho(\alpha_{e_o}) = \alpha$
$z \xrightarrow{\alpha} z w_2 w, E(z) = e_o$	$z \xrightarrow{\alpha_{e_o}} w$	$\rho(\alpha_{e_o}) = \alpha$
$w \xrightarrow{\alpha} w y_1 y, A(w) = e_o$	$w \xrightarrow{\alpha_{e_o, w}} y$	$\rho(\alpha_{e_o, w}) = \alpha$
$w \xrightarrow{\alpha} w y_2 y, A(w) = e_o \rightarrow \varepsilon$	$w \xrightarrow{\alpha_{e_o \rightarrow \varepsilon, w}} y$	$\rho(\alpha_{e_o \rightarrow \varepsilon, w}) = \alpha$

Definition 24: Let $\mathcal{T} = \{T_1, \dots, T_n\}$ be a TPO system such that $T_i = \langle Q_Y^i, Q_Z^i, Q_W^i, \Sigma^i, \Sigma_i^\varepsilon, \Theta^i, \rightarrow_{yz}^i, \rightarrow_{zz}^i, \rightarrow_{zw}^i, \rightarrow_{wy}^i, y_0^i \rangle$. Let $M_i^T = \langle \Sigma_{MT}^i, Q_M^i, \rightarrow_M^i, Q_M^0, Q_M^m \rangle$ be the monolithic transformed deterministic automaton of T_i , based on Definition 23. The transformed automaton system of \mathcal{T} is $\mathcal{G}^T = \{G_1^T, \dots, G_n^T\}$, where $G_i^T = \langle \Sigma_{G^T}^i, Q_i, \rightarrow^i, Q_i^0, Q_i^m \rangle$ and

- 1) $\Sigma_{G^T}^i = \Sigma_{MT}^i \cup \left[\bigcup_{\alpha \in (\Sigma^j \setminus \Sigma^i); j \neq i} ((\Sigma^i \cap \Sigma^j)_\alpha \cup (\Sigma^i \cap \Sigma^j)_{\alpha, w} \cup (\Sigma^i \cap \Sigma^j)_{\alpha \rightarrow \varepsilon, w}) \right]$;
- 2) $Q_i = Q_M^i$;
- 3) $\rightarrow_M^i \cup \{(p, \alpha, p) \mid p \in Q_y^i \text{ and } \alpha \in \bigcup_{j \neq i} (\Sigma^j \setminus \Sigma^i)\}$;
- 4) $Q_i^0 = Q_M^0$;
- 5) $Q_i^m = Q_M^m$.

Since some shared events in the transformed automaton become local after incorporating the extra state information, they need to be added in the alphabet of the transformed automaton, $\bigcup_{\alpha \in (\Sigma^j \setminus \Sigma^i); j \neq i} ((\Sigma^i \cap \Sigma^j)_\alpha \cup (\Sigma^i \cap \Sigma^j)_{\alpha, w} \cup (\Sigma^i \cap \Sigma^j)_{\alpha \rightarrow \varepsilon, w})$ in Definition 24. Moreover, the events not defined from Y states of certain TPO T but defined from Y states of some other TPO T' are added as self-loops at the corresponding states in the transformed automaton of T , $\{(p, \alpha, p) \mid p \in Q_y^i \text{ and } \alpha \in \bigcup_{j \neq i} (\Sigma^j \setminus \Sigma^i)\}$. To create a map between the events of a TPO and its transformed automaton, *renaming* of events is necessary. Note that when the transformation of a single automaton is considered, Definitions 23 and 24 produce the same results. Thus, in the following, wherever a transformed automaton is discussed, we refer to Definition 24.

Renaming ρ simply removes the extra information from the events of the transformed automaton and maps them back to the original events in the TPO. To be more specific, ρ is a map such that $\rho(\alpha_\sigma) = \alpha$ and $\rho(\alpha) = \alpha$. Table I shows how the events in a transformed automaton are linked to the original events of a TPO, while the third column shows how renaming works. Specifically, in the case where events label \rightarrow_{yz} transitions, renaming does not change events names.

Example 4: Consider the abstracted system $\tilde{\mathcal{G}} = \{\tilde{G}_1, \tilde{G}_2\}$, shown in Fig. 2. The sets of secret states are $\tilde{Q}_1^S = \{q_3\}$, $\tilde{Q}_2^S = \{s_3\}$, where all the events are observable. T_1' and T_2' are the largest TPOs of \tilde{G}_1 and \tilde{G}_2 , respectively. In Example 3, the monolithic transformed automata of T_1' and T_2' were generated. The TPO system $\{T_1', T_2'\}$ is transformed to automata system $\mathcal{G} = \{G_1^T, G_2^T\}$, shown Fig. 2, by adding self-loops at the marked states. Event β is not in the alphabet of T_1' so it appears as a self-loop at all marked states in G_1^T , which correspond to Y

states in T_1' . Similarly, γ is added as a self-loop at marked states in G_2^T since γ is not in the alphabet of T_2' .

In the following, Theorem 8 proves that if the synchronization of transformed individual TPOs contains a transition, then the largest monolithic TPO w.r.t. the synchronized system also contains an equivalent transition. However, the inverse is not necessarily true as there are some behaviors in the monolithic TPO that are omitted in the modular structure. Before Theorem 8, Lemma 6 [38] and Lemma 7 establish that the modular observer and desired observer are isomorphic to their monolithic counterparts.

Lemma 6 (see[38]): Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^0 \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^0 \rangle$ be two nondeterministic automata. Then, $\det(G_1 \parallel G_2)$ is isomorphic to $\det(G_1) \parallel \det(G_2)$.

Lemma 7: Let $G_1 = \langle Q_1, \Sigma_1, \rightarrow_1, Q_1^0 \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \rightarrow_2, Q_2^0 \rangle$ be two nondeterministic automata with sets of secret states Q_1^S and Q_2^S , respectively. Then, $\det_d(G_1) \parallel \det_d(G_2)$ is isomorphic to $\det_d(G_1 \parallel G_2)$.

Proof: From $\det(G_1 \parallel G_2)$ is isomorphic to $\det(G_1) \parallel \det(G_2)$, it follows that $\det(G_1 \parallel G_2) \xrightarrow{s} X$ if and only if $\det(G_1) \parallel \det(G_2) \xrightarrow{s} (X_1, X_2)$ and $(x_1, x_2) \in X$ if and only if $(x_1, x_2) \in X_1 \times X_2$. Now, we need to show that $X \notin X_{\text{obsd}}$ if and only if $(X_1, X_2) \notin X_{1, \text{obsd}} \times X_{2, \text{obsd}}$.

First, assume $X \notin X_{\text{obsd}}$, which means $X \subseteq Q^S$. This further means that for all $(x_1, x_2) \in X$, either $x_1 \in Q_1^S$ or $x_2 \in Q_2^S$, which implies either $X_1 \notin X_{1, \text{obsd}}$ or $X_2 \notin X_{2, \text{obsd}}$. Thus, $(X_1, X_2) \notin X_{1, \text{obsd}} \times X_{2, \text{obsd}}$.

Now, assume $(X_1, X_2) \notin X_{1, \text{obsd}} \times X_{2, \text{obsd}}$. This means either $X_1 \notin X_{1, \text{obsd}}$ or $X_2 \notin X_{2, \text{obsd}}$, which implies either $X_1 \subseteq Q_1^S$ or $X_2 \subseteq Q_2^S$. Hence, for all $(x_1, x_2) \in (X_1, X_2) = X$ either $x_1 \in Q_1^S$ or $x_2 \in Q_2^S$, which implies $X \subseteq Q^S$. Thus, $X \notin X_{\text{obsd}}$. ■

Theorem 8: Let $G_1 = \langle Q_1, \Sigma_1, \rightarrow_1, Q_1^0 \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \rightarrow_2, Q_2^0 \rangle$ be two nondeterministic automata with sets of secret states Q_1^S and Q_2^S , respectively. Let $T_1 = \langle Q_Y^1, Q_Z^1, Q_W^1, \Sigma_1, \Sigma_1^\varepsilon, \Theta_1, \rightarrow_{yz}^1, \rightarrow_{zz}^1, \rightarrow_{zw}^1, \rightarrow_{wy}^1, y_0^1 \rangle$ and $T_2 = \langle Q_Y^2, Q_Z^2, Q_W^2, \Sigma_2, \Sigma_2^\varepsilon, \Theta_2, \rightarrow_{yz}^2, \rightarrow_{zz}^2, \rightarrow_{zw}^2, \rightarrow_{wy}^2, y_0^2 \rangle$ be the largest TPOs w.r.t. G_1 and G_2 , respectively. Let $G_1^T = \langle \Sigma_{G_1^T}, Q_1, \rightarrow^1, Q_1^0, Q_1^m \rangle$ and $G_2^T = \langle \Sigma_{G_2^T}, Q_2, \rightarrow^2, Q_2^0, Q_2^m \rangle$ be the transformed automata of T_1 and T_2 , respectively. Let T be the largest monolithic TPO w.r.t. $G_1 \parallel G_2$. Then, let $\rho : (\Sigma_{G_1^T} \cup \Sigma_{G_2^T}) \rightarrow (\Sigma_1 \cup \Sigma_1^\varepsilon \cup \Theta_1) \cup (\Sigma_2 \cup \Sigma_2^\varepsilon \cup \Theta_2)$ be a renaming. We have $[G_1^T \parallel G_2^T \xrightarrow{s} (q_1, q_2)] \Rightarrow [T \xrightarrow{\rho(s)} q]$.

Formal proof of Theorem 8 can be found in [55]. Theorem 8 shows that synchronization of individual transformed TPOs is a subsystem of the largest monolithic TPO. Specifically, if there is a string s in $G_1^T \parallel G_2^T$, then there always exists a corresponding path $\rho(s)$ in T . The synchronized automaton form TPOs may not always be equal to the monolithic TPO, since some \rightarrow_{zz} transitions may not appear in the synchronized system. This happens when a state in the synchronization of TPOs is a combination of an original Z and an original Y state from individual TPOs, while the observable event component of the original Z state is a local event.

However, as there is no difference between local and shared events in the monolithic approach of obtaining TPOs, the

largest monolithic TPO contains all possible transitions of edit decisions.

The proof of Theorem 8 also illustrates that for every state in the largest monolithic TPO, there exists a corresponding state in the synchronized individual TPOs in automaton form.

Remark 2: Although the statement of Theorem 8 concentrates on the case of two individual systems, the result can be generalized to more than two individual systems. ■

Remark 3: Notice that Theorem 8 illustrates that some transitions are “missing” in the synchronized automaton compared with the largest monolithic TPO T . It further implies that more transitions will be missing if we synchronize more individual TPOs (in automaton form). Actually, we may locate those missing transitions and add them back to the synchronized automaton $\prod_{i=1}^n G_i^T$.

Specifically, consider states (q_1, q_2, \dots, q_n) and $(q'_1, q'_2, \dots, q'_n)$ in $\prod_{i=1}^n G_i^T$ such that $q_i, q'_i \in Q_Y^i \cup Q_Z^i$ for all i , i.e., every component in those states is either a Y state or a Z state from an individual transformed automaton. Then, we add transition $(q_1, q_2, \dots, q_n) \xrightarrow{\sigma} (q'_1, q'_2, \dots, q'_n)$ if there exists a set of indexes $\mathbb{I} \in 2^{\{1,2,\dots,n\}}$, such that for all $i \in \mathbb{I}$, $q_i = (x_{i,d}, x_{i,f}), q'_i = (x'_{i,d}, x_{i,f}) \in Q_Y^i$ and $x_{i,d} \xrightarrow{\sigma} x'_{i,d}$ in $\det_d(G_i)$; while for all $i \notin \mathbb{I}$, $q_i = q'_i$. Intuitively, the added transition implies that event σ may be inserted in the largest monolithic TPO w.r.t. $\prod_{i=1}^n G_i$. However, due to the fact that there are no transitions defined from a Y state to another Y state in TPOs, those transitions are missing in $\prod_{i=1}^n G_i^T$, which implies the synchronized system $\prod_{i=1}^n G_i^T$ may only contain a subset of edit decisions in the largest monolithic TPO.

However, the above-mentioned operation may not be preferred in practice, since it involves explicitly synchronizing individual TPOs in their automaton form. This is usually not feasible in modular approaches and should be avoided in our CA-AES algorithm as well. ■

Finally, the results of this section are formally recapped in Theorem 9, which illustrates that the synchronization of transformed (automaton form) TPOs w.r.t. individual abstracted systems contain a subset of the transitions of the largest monolithic TPO. The proof follows directly from Theorems 5 and 8.

Theorem 9: Let G_1 and G_2 be two nondeterministic automaton with sets of secret states $Q_i^S \subseteq Q_i$ and sets of nonsecret states $Q_i^{NS} = Q \setminus Q_i^S$ for $i = 1, 2$. Let $\det(G_i)$ and $\det_d(G_i)$ be the observer and the desired observer of G_i , respectively. Let \sim_o be an opaque observation equivalence on G_i such that $\tilde{G}_i \sim_o G_i$ for $i = 1, 2$. Let $H_{i,ob} \approx_o \det(\tilde{G}_i)$ and $H_{i,b} \approx \det(\tilde{G}_i)$ for $i \in \{1, 2\}$, where \approx_o and \approx are opaque bisimulation and bisimulation, respectively. Let T be the largest TPO w.r.t. $G_1 \parallel G_2$ and let $T_i^t = \langle Q_Y^i, Q_Z^i, Q_W^i, \Sigma_i, \Sigma_i^e, \Theta_i, \rightarrow_{yz}^i, \rightarrow_{zz}^i, \rightarrow_{zw}^i, \rightarrow_{wy}^i, y_0^i \rangle$ be the largest TPO w.r.t. $\det_d(H_{i,ob})$ and $H_{i,b}$ for $i \in \{1, 2\}$. Let $G_i^T = \langle \Sigma_{G_i^T}, Q_i, \rightarrow^i, Q_i^0, Q_i^m \rangle$ for $i = 1, 2$ be the transformed automata of T_i^t and let $\rho: (\Sigma_{G_1^T} \cup \Sigma_{G_2^T}) \rightarrow (\Sigma_1 \cup \Sigma_1^e \cup \Theta_1) \cup (\Sigma_2 \cup \Sigma_2^e \cup \Theta_2)$ be a renaming. We have $[G_1^T \parallel G_2^T \xrightarrow{\rho(s)} (q_1, q_2)] \stackrel{\tau}{\cong} [T \xrightarrow{\rho(s)} q]$.

Example 5: Consider the system $\mathcal{G} = \{G_1, G_2\}$ shown in Fig. 2. In the first step of the compositional approach, the system

is abstracted by applying opaque observation equivalence (see Example 2). The abstracted system $\tilde{\mathcal{G}} = \{\tilde{G}_1, \tilde{G}_2\}$ is shown in Fig. 2. Next, the TPO of individual components are built. As explained in Example 4, the TPOs of \tilde{G}_1 and \tilde{G}_2 are T_1^t and T_2^t , respectively, shown in Fig. 2. Moreover, Fig. 2 also shows G_1^T and G_2^T , the transformed automata of T_1^t and T_2^t , respectively. The largest monolithic TPO w.r.t. $G_1 \parallel G_2$ is denoted by T and is shown in Fig. 3.

In this particular example, it can be verified that the original and abstracted TPOs are identical (this need not be true in general); therefore, T in Fig. 3 also represents the largest TPO w.r.t. $\tilde{G}_1 \parallel \tilde{G}_2$. In T , we have states $A = \{(q_0, s_0)\}$, $B = \{(q_0, s_1), (q_0, s_2)\}$, $C = \{(q_1, s_0), (q_2, s_0)\}$, $D = \{(q_1, s_1), (q_2, s_1), (q_2, s_2), (q_1, s_2)\}$, and $E = \{(q_3, s_3)\}$.

After synchronizing G_1^T with G_2^T , we find that there are some transitions in Fig. 3, which do not correspond to any transition in $G_1^T \parallel G_2^T$. For example, no transition in $G_1^T \parallel G_2^T$ corresponds to the zz transition of β from state (A, B, γ) to (B, B, γ) in Fig. 3.

V. FROM AES CALCULATION TO SUPERVISOR SYNTHESIS

So far, we have shown that in our compositional and abstraction-based approach, individual components can be abstracted, and each largest TPO w.r.t. an abstracted component can be calculated individually. Then, we transfer those TPOs to their automaton forms. After that, we have also shown in Theorem 9 that the synchronization of the transformed TPOs results in a subsystem of the largest monolithic TPO up to the renaming of events.

Recall that the AES is obtained after pruning deadlocking states from the largest TPO. Here the modular structure of the transformed TPO is kept and the calculation of a “modular edit structure” can be done by mapping this problem to a modular nonblocking supervisory control problem under full observation.

As was discussed at the end of Section II-C, we pursue an approach to convert the pruning process (from the largest TPO to the AES) to a supervisory control problem. In this setting, the plant is a collection of automata transformed from individual largest TPOs obtained at the end of step (iv) of CA-AES algorithm. The specification is the automaton form of the edit constraint. The constraint of having up to $n + 1$ consecutive erasures can be modeled by a specification automaton with n states, where transitions are labeled by the decision events and all states are marked except the last state, which is a blocking state. After n consecutive event erasures, the next transition of event erasure $\alpha \rightarrow \varepsilon_\alpha$ leads the specification forward to a blocking state. If the next event is a nonerasure event, it leads the specification back to the initial state, thus resetting the sequence of erasures. Since we have a *modular* representation of the plant, we are able to leverage computationally efficient compositional techniques for modular nonblocking supervisory control problems.

Definition 25: Let $\mathcal{T} = \{T_1, \dots, T_n\}$ be a TPO system, where $T_i = \langle Q_Y^i, Q_Z^i, Q_W^i, \Sigma_i, \Sigma_i^e, \Theta_i, \rightarrow_{yz}^i, \rightarrow_{zz}^i, \rightarrow_{zw}^i,$

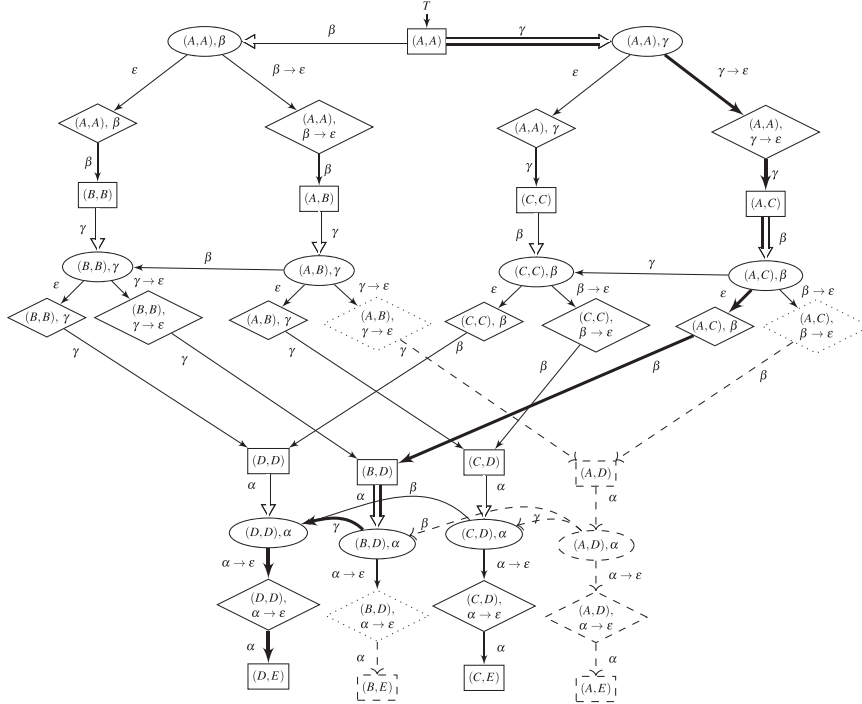


Fig. 3. Monolithic largest TPO w.r.t. to $G_1 \parallel G_2$ (also same as that w.r.t. $\tilde{G}_1 \parallel \tilde{G}_2$ in this particular case) in Example 5.

$\rightarrow_{wy}^i, y_0^i$), and let Φ be the edit constraint on \mathcal{T} such that there are not $n + 1$ consecutive event erasures.

Then, $K = \langle \Sigma_K, Q_K, \rightarrow_K, Q_K^0, Q_K^K \rangle$ is the automaton form of Φ , where we have the following.

- 1) $Q_K = \{x_1, \dots, x_n\}$.
- 2) $\rightarrow_K = \bigcup_{1 \leq i \leq n-1} \{(x_i, \alpha \rightarrow \varepsilon_\alpha, x_{i+1}) \mid p \xrightarrow{\alpha \rightarrow \varepsilon_\alpha} zw\}$
 q and $E(p) = \alpha\} \cup \bigcup_{1 \leq i \leq n-2} \{(x_{i+1}, \varepsilon_\alpha, x_1) \mid p \xrightarrow{\varepsilon_\alpha} zw\}$
 q and $E(p) = \alpha\} \cup \bigcup_{1 \leq i \leq n-2} \{(x_{i+1}, \alpha_\sigma, x_1) \mid p \xrightarrow{\alpha_\sigma} zz\}$
 q and $E(p) = \sigma\} \cup \{(x_1, \varepsilon_\alpha, x_1) \mid p \xrightarrow{\varepsilon_\alpha} zw\}$ and $E(p) = \alpha\}$
 $\cup \{(x_1, \alpha_\sigma, x_1) \mid p \xrightarrow{\alpha_\sigma} zz\}$ and $E(p) = \sigma\}$.
- 3) $Q_K^0 = x_1$.
- 4) $Q_K^K = \{x_1, \dots, x_{n-1}\}$.

Example 6: Consider the transformed system $\mathcal{G}^T = \{G_1^T, G_2^T\}$ shown in Fig. 2. Assume the constraint Φ only allows one erasure. The specification automaton of this constraint is shown in Fig. 4 as K . Automaton K has three states. As there is no constraint on event insertion, the events related to event insertion just form self-loops at the initial state of K . On the other hand, by executing $\alpha \rightarrow \varepsilon_\alpha$, $\beta \rightarrow \varepsilon_\beta$, or $\gamma \rightarrow \varepsilon_\gamma$ the specification transits from x_1 to x_2 . Next, at x_2 if the edit decision is to certain events, then the system goes back to the initial state x_1 , thus allowing more event erasures since there are no consecutive erasures. However, if another event erasure occurs from x_2 , the system goes to the blocking state x_3 .

The following theorem establishes that the TPO of the system under constraint Φ and the transformed system synchronized with the specification K have the same runs up to a renaming of the events.

Theorem 10: Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with the set of secret states $Q^S \subseteq Q$. Let $T' = \langle Q'_Y, Q'_Z, Q'_W, \Sigma, \Sigma^\varepsilon, \Theta, \rightarrow'_{yz}, \rightarrow'_{zz}, \rightarrow'_{zw}, \rightarrow'_{wy}, y'_0 \rangle$ be

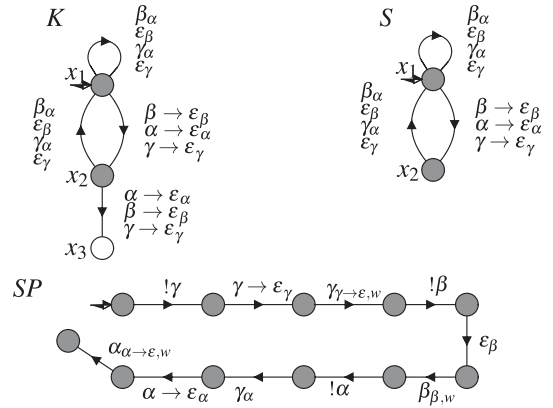


Fig. 4. Automaton K is the automaton form of the constraint Φ in Example 6; S and SP are the supervisor and the selected path in Example 7, respectively.

the largest TPO of G under the edit constraint Φ which prohibits $n + 1$ consecutive event erasures.

Let $T = \langle Q_Y, Q_Z, Q_W, \Sigma, \Sigma^\varepsilon, \Theta, \rightarrow_{yz}, \rightarrow_{zz}, \rightarrow_{zw}, \rightarrow_{wy}, y_0 \rangle$ be the largest TPO w.r.t. G when no constraint is considered and let $G^T = \langle \Sigma_{G^T}, Q, \rightarrow, Q^0 \rangle$ be the automaton transformation of T . Let K be the specification automaton of Φ . Then, $G^T \parallel K \xrightarrow{s}$ if and only if $T' \xrightarrow{\rho(s)}$.

Proof: Clearly, $T' \subseteq T$ and $\Sigma_K \subseteq \Sigma_{G^T}$. First, let $G^T \parallel K \xrightarrow{s} (q_G, q_K) \xrightarrow{\sigma} (p_G, p_K)$, let $P_K : \Sigma_{G^T} \rightarrow \Sigma_K$, and let $P_E : \Sigma_{G^T} \rightarrow \bigcup_{\sigma \in \Sigma} \{\sigma \rightarrow \varepsilon_\sigma \in \Sigma_K\}$ be a map that removes all the events except event erasures from Σ_K . From $G^T \parallel K \xrightarrow{s} (q_G, q_K) \xrightarrow{\sigma} (p_G, p_K)$ and $\Sigma_K \subseteq \Sigma_{G^T}$ and Definition 2, it holds

that $G^T \xrightarrow{s} q_G \xrightarrow{\sigma} p_G$ and $K \xrightarrow{P_K(s)} q_K \xrightarrow{P_K(\sigma)} p_K$. Now, consider three cases.

- 1) $q_K = x_i$, $p_K = x_{i+1}$ and $p_K, q_K \in Q_m^K$. Then, $|P_E(s)| < n$. This implies that σ is an erasure event, but there are not n consecutive erasures in s , so $T' \xrightarrow{\rho(s)} q_T \xrightarrow{\rho(\sigma)} p_T$.
- 2) $q_K = x_i$, $p_K = x_{i+1}$, $q_K \in Q_m^K$ but $p_K \notin Q_m^K$. Then, $|P_E(s)| = n$ and $K \xrightarrow{P_K(s)} q_K \xrightarrow{\sigma} p_K$, which implies $G^T \parallel K \xrightarrow{s\sigma} (p_G, p_K)$ and (p_G, p_K) is a blocking state. This further indicates there are n consecutive erasures in s , so $T' \xrightarrow{\rho(s)} q_T \not\rightarrow$.
- 3) $q_K = x_i$, $p_K = x_1$ and $p_K, q_K \in Q_m^K$. This implies σ is a nonerasure event, so $T' \xrightarrow{\rho(s)} q_T \xrightarrow{\rho(\sigma)} p_T$.

Now, assume $T' \xrightarrow{\rho(s)} q_T$. This means $T \xrightarrow{\rho(s)} q_T$, which implies $G^T \xrightarrow{s} q_T$. Consider two cases. 1. $|P_E(s)| < n$. Then, $K \xrightarrow{P_K(s)} q_K$, which implies $G^T \parallel K \xrightarrow{s} (q_G, q_K)$. 2. $|P_E(s)| = n$. This means $T' \xrightarrow{\rho(s)} q_T \not\rightarrow$ and $K \xrightarrow{P_K(s)} q_K \not\rightarrow$, which implies $G^T \parallel K \xrightarrow{s} (q_G, q_K) \not\rightarrow$. ■

The following theorem shows that equivalent states are removed in solving the supervisory control problem to obtain an automaton satisfying the edit constraint and in the pruning process to obtain the AES from the largest TPO.

Theorem 11: Let $G = \langle \Sigma, Q, \rightarrow, Q^0 \rangle$ be a nondeterministic automaton with the set of secret states $Q^S \subseteq Q$, and let Φ be the edit constraint which prohibits $n + 1$ consecutive event erasures. Let $T = \langle Q_Y, Q_Z, Q_W, \Sigma, \Sigma^e, \Theta, \rightarrow_{yz}, \rightarrow_{zz}, \rightarrow_{zw}, \rightarrow_{wy}, y_0 \rangle$ be the largest TPO w.r.t. G without considering the edit constraint and let $G^T = \langle \Sigma_{G^T}, Q, \rightarrow, Q^0 \rangle$ be the transformed automaton of T . Let $\rho : \Sigma_{G^T} \rightarrow (\Sigma_1 \cup \Sigma_1^e \cup \Theta_1)$ be a renaming and let AES be the all edit structure obtained from T . Let S be the supremal controllable and nonblocking subautomaton of G^T after considering the specification introduced by Φ . Then, $S \xrightarrow{\rho^{-1}(s)} q$ if and only if $AES \xrightarrow{\rho(s)} q$.

Formal proof of Theorem 11 can be found in [55]. Theorem 11 proves that when it comes to imposing the edit constraint, the pruning process from the largest TPO to the AES removes equivalent states with the synthesis procedure of a supremal supervisor. Hence, no information is lost when we apply the supervisory control approach to enforce the edit constraints and obtain edit functions. This result is essential to show that the transformation of the TPO to an equivalent automaton and the constraint Φ to specification K is correctly done in Definitions 23 and 25, respectively. The next step is to consider the modular representation of the system. In that case, we will use the transformation in Definition 24, which results in a set of automata transformations of the individual TPOs, with necessary self-loops to capture the synchronization among the components. Finally, we combine the results about abstraction and decomposition, which results in Theorem 12.

Theorem 12: Let $\mathcal{G} = \{G_1, \dots, G_n\}$ be a modular nondeterministic system with sets of secret states Q_i^S . Let AES be the all edit structure of \mathcal{G} under constraint Φ . Let $\det(G_i)$ and $\det_d(G_i)$ be the observer and the desired observer of G_i , respectively. Let \sim_o be an opaque observation equivalence on

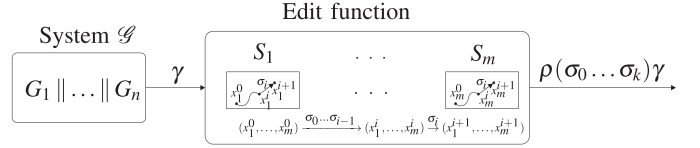


Fig. 5. Process of selecting an edit decision from the modular AES.

G_i such that $\tilde{G}_i \sim_o G_i$ for $i = 1, \dots, n$. Let $H_{i,ob} \approx_o \det(\tilde{G}_i)$ and $H_{i,b} \approx \det(\tilde{G}_i)$ for $i = 1, \dots, n$, where \approx_o and \approx are opaque bisimulation and bisimulation, respectively. Let T'_i be the largest TPO of $\det_d(H_{i,ob})$ and $H_{i,b}$ for $i = 1, \dots, n$ with the event set $\Sigma_{i,T}$. Let G_i^T be the transformed automaton of T'_i and K be the automaton specification. Let $P_e : \Omega \rightarrow \mathcal{L}(\mathcal{G})$ be an edit projection and $l(\omega)$ be a string generated by run (see Definitions 14 and 15). Let \mathcal{S} be the least restrictive controllable and nonblocking supervisor calculated from $\{G_1^T, \dots, G_n^T, K\}$ and let $\rho : (\Sigma_{G_1^T} \cup \dots \cup \Sigma_{G_n^T}) \rightarrow (\Sigma_{1,T} \cup \dots \cup \Sigma_{n,T})$ be the renaming map. Then, $[\forall s \in \mathcal{L}(\mathcal{G}), \exists t \in \mathcal{L}(\mathcal{S}) : P_e(\rho(t)) = s] \Rightarrow [l(\rho(t)) = f_e(s) \text{ where } f_e \in AES]$.

Proof: The proof follows directly from Theorems 10 and 11, in combination with Theorem 9. ■

Theorem 12 essentially shows the proof for all the steps shown in Fig. 1. The theorem shows that the CA-AES algorithm correctly synthesizes edit functions for opacity enforcement in a modular form; therefore, the algorithm is sound. It also reveals that the problem of calculating the modular representation of the AES can be transformed to synthesizing modular supervisors. The advantage of such a transformation is that we may leverage various existing approaches for calculating a modular supremal nonblocking supervisor in the literature; see, e.g., [11], [28], [29], and [37]. Therefore, we can obtain a modular representation of the AES, which is noticeably efficient to compute. Then, we may synchronize individual components in the modular edit structure, which results in a subsystem of the monolithic AES. However, as was pointed out in Section IV, some edit decisions are omitted after the synchronization. In practice, it is usually desired to retain the modular structure and extract an edit function from it, much in the same way as a set of modular supervisors control a plant. The extraction process is explained next.

Each step of extracting a valid edit decision is described in Fig. 5. Here, the edit function is an interface between the system's output and the outside environment. Assume that the system outputs event γ ; then, the edit function makes an edit decision for that event, and the edited string will be output to the external observers.

Specifically, this process contains the following steps.

- 1) When γ is received by the edit function, all the components of the modular edit structure are in states that correspond to Y states of the AES.
- 2) At these states, event γ is executed, and states of all the components in the modular edit structure are updated simultaneously. After the execution of γ , each component of the modular edit structure is at a state that corresponds to a Z state of the AES.

- 3) Then, assume that there are multiple transitions defined out of such a current state, we need to select one common transition, which corresponds to a specific edit decision and can be viewed as making a control decision from the current state.

Note that as the selected transition needs to be accepted by all the components, thus it may happen spontaneously in all the components of the system. The solution of the modular supervisory control problem guarantees the existence of such a common transition out of the current Z states.

The CA-AES algorithm returns the edited string $\rho(\sigma_0 \dots \sigma_k)$ for event γ when every component of the modular edit structure reaches a new state corresponding to a Y state of the AES. At that point, the modular edit structure is ready to process the next event output by the system, and the above steps repeat. Meanwhile, the algorithm keeps track of the states of the modular edit structure as its components evolve. Based on Theorem 11, the edited string $\rho(\sigma_0 \dots \sigma_k)$ is accepted by the monolithic AES. This finally confirms that the extracted edit decision from the modular edit structure corresponds to a valid edit decision in the monolithic AES. The above process is illustrated in the following example.

Example 7: Consider the nondeterministic system $\mathcal{G} = \{G_1, G_2\}$ shown in Fig. 2. As it was shown in Example 2, the system can be abstracted using opaque observation equivalence. After abstraction, the system becomes deterministic, which means that there is no need to calculate the observers of G_1 and G_2 . The largest TPOs of \tilde{G}_1 and \tilde{G}_2 are T'_1 and T'_2 , respectively, shown in Fig. 2. Next, the TPOs are transformed to automata G_1^T and G_2^T shown in Fig. 2, as explained in Example 4.

Assume that the user adds an edit constraint such that only one consecutive erasure is allowed as in Example 6. The specification automaton of this constraint is K , shown in Fig. 4. Due to this constraint, the Y states (A, D) and (B, E) are considered undesired states in T , shown in Fig. 3, and they should not be reached when we synthesize edit functions. Since (A, D) is not allowed, its successor states (A, D, α) , $(A, D, \alpha \rightarrow \varepsilon)$, and (A, E) become unreachable from the initial state (A, A) . Those three states together with (A, D) and (B, E) are drawn in dashed lines in Fig. 3 and are to be removed in the next step. Furthermore, states $(B, D, \alpha \rightarrow \varepsilon)$, $(A, C, \beta \rightarrow \varepsilon)$, and $(A, B, \gamma \rightarrow \varepsilon)$ become deadlocking after (A, D) and (B, E) are removed. They are drawn in dotted lines in Fig. 3 and are also to be removed.

Following the compositional approach with supervisor reduction presented in [29] and [40], we calculate a least restrictive and nonblocking supervisor for the transformed automaton, which is shown in Fig. 4 as automaton S . All the paths accepted by this supervisor represent valid edit decisions. Consider the accepted path SP shown in Fig. 4, it corresponds to an edit function's decisions for string $\gamma\beta\alpha$ such that $f_e(\gamma\beta\alpha) = \gamma(\gamma \rightarrow \varepsilon)\gamma\beta\varepsilon\beta\alpha\gamma(\alpha \rightarrow \varepsilon)\alpha$, which is shown by thick lines in T , Fig. 3. As is seen, $\rho(SP) = f_e(\gamma\beta\alpha)$. Specifically, when event γ is output by the system, SP returns $\gamma(\gamma \rightarrow \varepsilon_\gamma)(\gamma_{\gamma \rightarrow \varepsilon, w})$, which means erasing the γ . Next, event β is output by the system, and it is unchanged according to SP . Finally, α is output by the system and SP returns $\alpha(\gamma_\alpha)(\alpha \rightarrow \varepsilon_\alpha)(\alpha_{\alpha \rightarrow \varepsilon, w})$, which means erasing α and inserting γ . Similarly, we may consider

other paths accepted by the supervisor in Fig. 4 to track edit decisions on other strings; then, we have a complete picture of how an edit function works.

Remark 4: From the result in Theorem 8, our modular algorithm CA-AES results in fewer edit decisions compared with the monolithic approach in [20], due to the synchronization process in Section IV. This indicates that our method may not be complete in the sense that even if the CA-AES algorithm does not return any modular form edit functions, the monolithic approach may still return valid edit functions. This may be viewed as the tradeoff of reducing computational complexity by the modular method. ■

VI. CONCLUSION

This article investigated a compositional and abstraction-based approach to synthesize edit functions for opacity enforcement in a modular setting, given a set of individual systems. The edit functions modify the system's output by inserting and erasing events, under the constraint of limited number of event erasures. The TPO and AES proposed in our prior work were employed here; these discrete structures embed edit functions and reflect the constraints. The monolithic approach first synchronizes all individual systems and then calculates the monolithic AES to obtain edit functions. In contrast, the compositional approach first exploits the modular structure and builds individual TPOs. Then, it incorporates the edit constraint and calculates the modular edit structure in a nonblocking modular supervisory control manner to obtain edit functions. In addition, we also applied abstraction methods to reduce the state space of the system before opacity enforcement. We showed that the abstraction processes preserve opacity. Combining system composition and abstraction, we proposed an efficient approach to enforce opacity for complex systems containing multiple components.

REFERENCES

- [1] B. Bérard, K. Chatterjee, and N. Sznajder, "Probabilistic opacity for Markov decision processes," *Inf. Process. Lett.*, vol. 115, no. 1, pp. 52–59, 2015.
- [2] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *Int. J. Inf. Secur.*, vol. 7, no. 6, pp. 421–435, 2008.
- [3] J. W. Bryans, M. Koutny, and P. Y. A. Ryan, "Modelling opacity using Petri nets," *Electron. Notes Theor. Comput. Sci.*, vol. 121, pp. 101–115, 2005.
- [4] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Berlin, Germany: Springer, 2008.
- [5] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods Syst. Des.*, vol. 40, no. 1, pp. 88–115, 2012.
- [6] S. Chédor, C. Morvan, S. Pinchinat, and H. Marchand, "Diagnosis and opacity problems for infinite state systems modeled by recursive tile systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 25, nos. 1/2, pp. 271–294, 2015.
- [7] J. Chen, M. Ibrahim, and R. Kumar, "Quantification of secrecy in partially observed stochastic discrete event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 185–195, Jan. 2017.
- [8] P. Darondeau, H. Marchand, and L. Ricker, "Enforcing opacity of regular predicates on modal transition systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 25, nos. 1/2, pp. 251–270, 2015.

- [9] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory control for opacity," *IEEE Trans. Autom. Control*, vol. 55, no. 5, pp. 1089–1100, May 2010.
- [10] Y. Falcone and H. Marchand, "Enforcement and validation (at runtime) of various notions of opacity," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 25, no. 4, pp. 531–570, 2015.
- [11] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Trans. Autom. Control*, vol. 53, no. 6, pp. 1449–1461, Jul. 2008.
- [12] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Sci. Comput. Program.*, vol. 13, nos. 2/3, pp. 219–236, 1990.
- [13] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 17, no. 4, pp. 475–504, 2007.
- [14] L. Hélouët, H. Marchand, and L. Ricker, "Opacity with powerful attackers," in *Proc. 14th IFAC Int. Workshop Discrete Event Syst.*, 2018, pp. 475–482.
- [15] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.
- [16] R. Jacob, J.-J. Lesage, and J.-M. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annu. Rev. Control*, vol. 41, pp. 135–146, 2016.
- [17] Y. Ji and S. Lafortune, "Enforcing opacity by publicly known edit functions," in *Proc. 56th IEEE Conf. Decis. Control*, 2017, pp. 4866–4871.
- [18] Y. Ji, Y.-C. Wu, and S. Lafortune, "Enforcement of opacity by public and private insertion functions," *Automatica*, vol. 93, pp. 369–378, 2018.
- [19] Y. Ji, X. Yin, and S. Lafortune, "Enforcing opacity by insertion functions under multiple energy constraints," *Automatica*, vol. 108, 2019, Art. no. 108476.
- [20] Y. Ji, X. Yin, and S. Lafortune, "Opacity enforcement using nondeterministic publicly-known edit functions," *IEEE Trans. Autom. Control*, vol. 64, no. 10, pp. 4369–4376, Oct. 2019, doi: [10.1109/TAC.2019.2897553](https://doi.org/10.1109/TAC.2019.2897553).
- [21] C. Keroglou and C. N. Hadjicostis, "Probabilistic system opacity in discrete event systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 28, no. 2, pp. 289–314, 2018.
- [22] F. Lin, "Opacity of discrete event systems and its applications," *Automatica*, vol. 47, no. 3, pp. 496–503, 2011.
- [23] R. Malik and R. Leduc, "Compositional nonblocking verification using generalized nonblocking abstractions," *IEEE Trans. Autom. Control*, vol. 58, no. 8, pp. 1891–1903, Aug. 2013.
- [24] T. Masopust and X. Yin, "Complexity of detectability, opacity and A-diagnosability for modular discrete event systems," *Automatica*, vol. 101, pp. 290–295, 2019.
- [25] J. Milner, *Communication and Concurrency*, vol. 84, New York, NY, USA: Prentice-Hall, 1989.
- [26] S. Mohajerani, Y. Ji, and S. Lafortune, "Efficient synthesis of edit functions for opacity enforcement using bisimulation-based abstractions," in *Proc. 57th IEEE Conf. Decis. Control*, 2018, pp. 3573–3578.
- [27] S. Mohajerani and S. Lafortune, "Transforming opacity verification to non-blocking verification in modular systems," *IEEE Trans. Autom. Control*, p. 1, 2019, doi: [10.1109/TAC.2019.2934708](https://doi.org/10.1109/TAC.2019.2934708).
- [28] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional synthesis of modular nonblocking supervisors," *IEEE Trans. Autom. Control*, vol. 59, no. 1, pp. 150–162, Jan. 2014.
- [29] S. Mohajerani, R. Malik, and M. Fabian, "Compositional synthesis of supervisors in the form of state machines and state maps," *Automatica*, vol. 76, pp. 277–281, 2017.
- [30] J. Mullins and M. Yeddes, "Opacity with orwellian observers and intransitive non-interference," in *Proc. 12th IFAC Int. Workshop Discrete Event Syst.*, 2014, pp. 344–349.
- [31] M. Noori-Hosseini, B. Lennartson, and C. Hadjicostis, "Compositional visible bisimulation abstraction applied to opacity verification," in *Proc. 14th IFAC Int. Workshop Discrete Event Syst.*, 2018, pp. 434–441.
- [32] J. JMM Rutten, "Automata and coinduction (an exercise in coalgebra)," in *Proc. Int. Conf. Concurrency Theory*, 1998, pp. 194–218.
- [33] A. Saboori and C. N. Hadjicostis, "Verification of infinite-step opacity and complexity considerations," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1265–1269, May 2012.
- [34] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *Proc. 46th IEEE Conf. Decis. Control*, 2007, pp. 5056–5061.
- [35] A. Saboori and C. N. Hadjicostis, "Opacity-enforcing supervisory strategies via state estimator constructions," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1155–1165, May 2012.
- [36] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of discrete event systems," *Inf. Sci.*, vol. 246, pp. 115–132, 2013.
- [37] K. W. Schmidt and J. E. R. Cury, "Efficient abstractions for the supervisory control of modular discrete event systems," *IEEE Trans. Autom. Control*, vol. 57, no. 12, pp. 3224–3229, Dec. 2012.
- [38] C. Seatzu, M. Silva, and J. H. van Schuppen, *Control of Discrete-Event Systems: Automata and Petri Net Perspectives*. Berlin, Germany: Springer, 2012.
- [39] R. Su, J. H. van Schuppen, and J. E. Rooda, "Model abstraction of non-deterministic finite-state automata in supervisor synthesis," *IEEE Trans. Autom. Control*, vol. 55, no. 11, pp. 2527–2541, Nov. 2010.
- [40] R. Su and W. M. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 14, no. 1, pp. 31–53, 2004.
- [41] S. Takai and Y. Oka, "A formula for the supremal controllable and opaque sublanguage arising in supervisory control," *SICE J. Control, Meas., Syst. Integr.*, vol. 1, no. 4, pp. 307–311, 2008.
- [42] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Decidability of opacity verification problems in labeled petri net systems," *Automatica*, vol. 80, pp. 48–53, 2017.
- [43] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Verification of state-based opacity using Petri nets," *IEEE Trans. Autom. Control*, vol. 62, no. 6, pp. 2823–2837, Jun. 2017.
- [44] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Berlin, Germany: Springer, 2019.
- [45] B. Wu and H. Lin, "Privacy verification and enforcement via belief abstraction," *IEEE Control Syst. Lett.*, vol. 2, no. 4, pp. 815–820, Oct. 2018.
- [46] Y.-C. Wu and S. Lafortune, "Synthesis of insertion functions for enforcement of opacity security properties," *Automatica*, vol. 50, no. 5, pp. 1336–1348, 2014.
- [47] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune, and S. A. Seshia, "Synthesis of obfuscation policies to ensure privacy and utility," *J. Autom. Reason.*, vol. 60, no. 1, pp. 107–131, 2018.
- [48] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 8, pp. 2140–2154, Aug. 2016.
- [49] X. Yin and S. Lafortune, "A new approach for the verification of infinite-step and K-step opacity using two-way observers," *Automatica*, vol. 80, pp. 162–171, 2017.
- [50] X. Yin and S. Lafortune, "A general approach for optimizing dynamic sensor activations for discrete event systems," *Automatica*, vol. 105, pp. 376–383, 2019.
- [51] X. Yin and S. Li, "Verification of opacity in networked supervisory control systems with insecure control channels," in *Proc. 57th IEEE Conf. Decis. Control*, 2018, pp. 4851–4856.
- [52] X. Yin, Z. Li, W. Wang, and S. Li, "Infinite-step opacity and K-step opacity of stochastic discrete-event systems," *Automatica*, vol. 99, pp. 266–274, 2019.
- [53] B. Zhang, S. Shu, and F. Lin, "Maximum information release while ensuring opacity in discrete event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 1067–1079, Jul. 2015.
- [54] K. Zhang, X. Yin, and M. Zamani, "Opacity of nondeterministic transition systems: A (bi) simulation relation approach," *IEEE Trans. Autom. Control*, p. 1, 2019, doi: [10.1109/TAC.2019.2908726](https://doi.org/10.1109/TAC.2019.2908726).
- [55] S. Mohajerani, Y. Ji, and S. Lafortune, "Compositional and abstraction-based approach for synthesis of edit functions for opacity enforcement," 2019, *arXiv:1910.00417*. [Online]. Available: <https://arxiv.org/abs/1910.00417>



Sahar Mohajerani received the B.Sc. degree in electrical engineering from the Khaje Nasir Toosi University of Technology, Tehran, Iran, in 2005, and the M.S. degree in systems, control, and mechatronics and the Ph.D. degree in automation from the Chalmers University of Technology, Gothenburg, Sweden, in 2009 and 2015, respectively.

She worked with Volvo Cars Corporation on function verification. She is currently a Postdoctoral Fellow with the University of Michigan, Ann Arbor, MI, USA. Her research interests include formal methods for verification and synthesis of large discrete event systems and cyber-physical systems.



Yiding Ji received the Bachelor of Engineering degree in electrical engineering from Tianjin University, Tianjin, China, in 2014, and the Master of Science and Ph.D. degrees in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2016 and 2019, respectively.

His research interests include theory of discrete event systems, security and privacy in cyber and cyber-physical systems, formal methods, and game theory.

Dr. Ji is a member of the IEEE Control Systems Society Technical Committee on Discrete Event Systems.



Stéphane Lafortune (F'99) received the B.Eng. degree from the École Polytechnique de Montréal, Montréal, QC, Canada, in 1980, the M.Eng. degree from McGill University, Montréal, in 1982, and the Ph.D. degree from the University of California at Berkeley, CA, USA, in 1986, all in electrical engineering.

Since September 1986, he has been with the University of Michigan, Ann Arbor, MI, USA, where he is currently a Professor of Electrical Engineering and Computer Science. His

research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems.

Dr. Lafortune received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the 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 coauthored with G. Barrett). He became a Fellow of the International Federation of Automatic Control in 2017.