Using Subobservers to Synthesize Opacity-Enforcing Supervisors

Richard Hugh Moulton1 · Behnam Behinaein Hamgini1 · Zahra Abedi Khouzani1 · Rômulo Meira-Góes2 · Fei Wang3 · Karen Rudie1,4

Received: 11 February 2020 / Accepted: 2 July 2022 / Published online: 19 September 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
In discrete-event system control, the worst-case time complexity for computing a system’s observer is exponential in the number of that system’s states. This results in practical difficulties since some problems require calculating multiple observers for a changing system, e.g., synthesizing an opacity-enforcing supervisor. Although calculating these observers in an iterative manner allows us to synthesize an opacity-enforcing supervisor and although methods have been proposed to reduce the computational demands, room exists for a practical and intuitive solution. Here we extend the subautomaton relationship to the notion of a subobserver and demonstrate its use in reducing the computations required for iterated observer calculations. We then demonstrate the subobserver relationship’s power by simplifying state-of-the-art synthesis approaches for opacity-enforcing supervisors under realistic assumptions.

Keywords Discrete-event systems · Supervisory control · Opacity

1 Introduction

Discrete-event systems (DES) research has recently focused on opacity, a system property that ensures that secret information cannot be distinguished from non-secret information. The literature includes methods for verifying if a system is opaque, namely, whether a hostile agent could—based on its observations—determine whether or not the system is in a secret state, or whether or not the system has generated a secret event sequence. A natural question to ask if a system is not opaque is "how can we make this system opaque?" One approach is to alter which events are visible to an adversary, however this is rarely in
the power of the system designer. An alternative approach is to use supervisory control to disable event occurrences so that the controlled system is opaque (Jacob et al. 2016).

Researchers have examined opacity-enforcing supervisors under various assumptions (Badouel et al. 2007; Ben-Kalefa and Lin 2011; Dubrei et al. 2010; Takai and Oka 2008; Tong et al. 2018; Yin and Lafortune 2015; 2016). The naïve method of producing such a supervisor starts by constructing a plant automaton representing the system and an observer automaton capturing the adversary’s view of the system based on which events the adversary can observe. The standard method for constructing an observer automaton relies on converting a nondeterministic finite automaton to a deterministic finite automaton (NFA-to-DFA conversion) (Hopcroft and Ullman 1979), which is known to require asymptotic computing time that is exponential in the number of states of the plant. Although supervisory control can then restrict plant behaviour to enforce opacity, a control-aware adversary will then be able to revise its plant estimates. As a result, repeated plant evolution necessitates repeated NFA-to-DFA operations. In this paper we present a method for synthesizing an opacity-enforcing supervisor that relies on only one NFA-to-DFA operation on the states of the uncontrolled plant.

Although we motivate our approach with the problem of plant evolution due to supervisory control, it is more generally a method for tracking the joint behaviour of a plant and an observer through multiple steps of plant evolution, without computing an observer at each intermediary stage. In addition supervisory control, a plant may evolve over time for other reasons as well. For example, a discrete-event process may be a naturally time-varying dynamic discrete-event system (Grigorov and Rudie 2006), may require online control (Chung et al. 1992), or may be best modelled by a time-varying automaton because more becomes known about the process over time. Alternatively, a discrete-event process may be controlled by a number of decentralized agents with different views of the plant, (Lin and Wonham 1988a), where controls enacted by one agent may allow another agent to update its own estimates of the plant.

The power of our approach is that it leads to efficient and intuitive computations. In the remainder of this paper we review previous approaches to the opacity control problem, we introduce the subobserver property and show that it can be used to incrementally refine the joint behaviour of a plant and an observer, and, finally, we demonstrate how this approach can be used to efficiently synthesize an opacity-enforcing supervisor.

2 Problem definition and related works

To lay the theoretical foundation for our contribution, we introduce the formalisms of DES, define the opacity control problem, and review the literature for related concepts and methods.

2.1 Discrete-event systems

DES are used to model processes that are discrete, asynchronous and potentially nondeterministic (Ramadge and Wonham 1987). The discrete nature of these systems is captured with an automaton: at any given time the system is in a specific state and the system moves between states via event-based transitions.

Mathematically, a DFA $G$ is a 5-tuple $(Q, \Sigma, \delta_G, q_0, Q_m)$ where $Q$ is the finite set of states in the system, $\Sigma$ is the finite alphabet of events that can occur, $\delta_G : Q \times \Sigma \rightarrow Q$ is a partial function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states. From a control theory perspective the automaton $G$ represents the plant, or underlying process, to be controlled. The
alphabet $\Sigma$ represents the events that can occur within the plant and can be divided into disjoint sets by considering whether individual events are controllable or uncontrollable ($\Sigma_c$ and $\Sigma_{uc}$ respectively) and whether they are observable ($\Sigma_o$ and $\Sigma_{uo}$ respectively). The transition function $\delta_G$ leads inductively from single events to the language $L(G)$, which is the plant’s full range of behaviour. Because $\delta_G$ is usually a partial function, we use the notation $\delta_G(q, \sigma)$ to denote that a specific transition $\delta_G(q, \sigma)$ is defined. We will also use the notation $(q', \sigma, q)_G$ to denote that $\delta_G(q', \sigma) = q$. The set of marked states may represent the completion of a process or, in our case, a set of distinctive states that we wish to keep indistinguishable to an adversary. In this paper we call these “secret states.”

A particular DFA representation of a language induces a binary relation called the equiresponse relation (Definition 1, from Lin and Wonham (1988b)) on $\Sigma^*$, the Kleene closure of $\Sigma$. The equiresponse relation for a DFA associates strings with the states they lead to, so that two strings are related if they lead to the same state. Because the equiresponse relation is a right congruence on $\Sigma^*$, it is also an equivalence relation and, like all equivalence relations, defines a partition. We note that different DFA representations of a language will induce different equiresponse relations and there is no canonical equiresponse relation.

**Definition 1 (Equiresponse relation)** Given an automaton $G = (Q, \Sigma, \delta_G, q_0, Q_m)$, the equiresponse relation of $G$ is a right congruence $eq(G)$ on $\Sigma^*$ and defined by

$$s \equiv s' \mod eq(G) \iff \delta_G(q_0, s) = \delta_G(q_0, s')$$

and if $\delta_G(q_0, s)$ is undefined then $\delta_G(q_0, s) = \delta_G(q_0, s')$ if and only if $\delta_G(q_0, s')$ is undefined as well.

### 2.1.1 Observing discrete-event systems

When an automaton contains unobservable events, an observer automaton can be used to capture an agent’s beliefs about the system. This observer automaton can be computed by transforming the original automaton, first by substituting the empty string $\epsilon$ for all transitions whose events are not in $\Sigma_o$ (Cassandras and Lafortune 2008, p. 76), and then performing NFA-to-DFA conversion (Cassandras and Lafortune 2008, pp. 87-90). Note that we do not use the standard marking associated with NFA-to-DFA conversion (whereby a subset labelling a state in the DFA is marked if any element in the subset is marked in the original NFA); rather, we mark a state in the DFA only if all elements of the subset label are marked states in the original NFA. As will be discussed in Section 5.1, this way of marking states in the observer automaton will distinguish states in which an adversary can be sure that the original system is in a secret state.

Throughout this paper we denote the transformation of an automaton to an observer automaton with respect to the alphabet $\Sigma_o$ by $T_{\Sigma_o}(\cdot)$; we denote the associated change in languages generated by these automata as the projection $P_{\Sigma_o}(\cdot)$ (Definition 2, from (Cassandras and Lafortune 2008, p. 57)).

**Definition 2 (Projection of strings)** For an alphabet $\Sigma$ and another alphabet $\Sigma_o \subseteq \Sigma$, we define the projection $P_{\Sigma_o} : \Sigma^* \rightarrow \Sigma_o^*$ recursively as follows:
The projection operation implicitly defines an equivalence relation over \( \Sigma^* \), where two strings \( s \) and \( s' \) are related if they share a projection, \( P_{\Sigma_o}(s) = P_{\Sigma_o}(s') \). Each cell in the associated partition contains those strings whose projections are equal. Different projections will define different partitions on \( \Sigma^* \) and we say that one partition refines another partition if it continues to distinguish between all elements that the latter partition distinguishes between (Definition 3) (Rosen 2019, Ch. 9).

**Definition 3 (Refinement of a partition)** Let \( \rho_1 \) and \( \rho_2 \) be binary relations on a set \( X \). We say that \( \rho_1 \) refines \( \rho_2 \), or that \( \rho_2 \) is coarser than \( \rho_1 \), denoted \( \rho_1 \leq \rho_2 \) if

\[
(\forall x, y \in X) \ (x, y) \in \rho_1 \implies (x, y) \in \rho_2.
\]

### 2.1.2 Controlling discrete-event systems

In addition to observing DES processes, we can also control them. In DES the controller that modifies which events can occur in a plant \( G \) is called the supervisor and is formally a pair \( S = (S, \phi) \). Here, \( S = (X, \Sigma, \xi, x_0, X_m) \) is a DFA that shares an alphabet with \( G \) but has a different state space, transition function and set of marked states (Ramadge and Wonham 1987). The second component of the pair, \( \phi : X \rightarrow \Gamma \), is a state feedback map that maps supervisor states, \( x \in X \), to control patterns \( \gamma \in \Gamma \),

\[
\gamma := \phi(x) \in \{0, 1\}^\Sigma.
\]

These control patterns indicate whether the supervisor will enable an event, denoted by 1, or disable it, denoted by 0, with the requirement that no uncontrollable event can be disabled. The controlled DES process is then represented by the supervisor composed with the plant and is denoted by

\[
S/G = Ac(X \times Q, \Sigma, \xi \times \delta, (x_0, q_0), X_m \times Q_m)
\]

where \( Ac \) is a function that restricts an automaton to its accessible portion: those states that can be reached from the initial state via a string in the automaton’s language (Ramadge and Wonham 1987). A control pattern can also be defined using the supervisor’s transition function, where an event \( \sigma \) is enabled in \( \phi(x) \) if \( \xi(x, \sigma) \) and it is disabled otherwise. A great amount of work exists in the literature for determining whether a system is controllable, whether a system is observable, and how to synthesize supervisors (Cassandras and Lafortune 2008; Wonham and Cai 2019).

### 2.2 The notion of opacity

When we talk about the opacity property of a system, we use the plain language meaning of an opaque system. That is, we say that a system is opaque if an observer of the system is unable to
unambiguously determine some characteristic of that system. Mazaré formalized this notion of opacity for computer science and showed that under this definition it is decidable whether or not a particular system property is opaque (Mazaré 2004). Formulating this notion of opacity for transition systems, Bryans et al. used epistemic logic and possible world models to frame opacity as the inability of an observer to determine the truth of a particular predicate related to the system (Bryans et al. 2008).

Building on this work, Saboori and Hadjicostis characterized a number of state-based opacities for DES including initial state, $k$-step, and infinite step opacities (Saboori and Hadjicostis 2008; 2011; 2012). Alternatively, Lin considered opacity to be a broader concept than Bryans et al. and defined opacity in terms of languages, distinguishing between strong and weak opacity in terms of the degree to which strings in one language are confused by an observer with strings in another language (Lin 2011).

More recently, Wu and Lafortune unified four common kinds of opacity: language-based, initial-state, current-state, and initial-and-final-state. They considered a problem setup where adversaries had full knowledge of the system’s structure but only partial observations of its behaviour. Their key result was a set of polynomial-time algorithms for transforming each of these four kinds of opacity into the other under the authors’ problem formulation. The one exception to this is that language-based opacity cannot be transformed into initial-state opacity if either the secret or non-secret language is not prefix-closed (Wu and Lafortune 2013). Without loss of generality, therefore, we consider in this paper the problem of enforcing current-state opacity, which we define in line with Wu and Lafortune (2013).

**Definition 4 (Current-state opacity)** Given a plant $G = (Q, \Sigma, \delta, q_0, Q_m)$, a projection $P_{\Sigma_s}$, and a disjoint sets of secret states $Q_S \subseteq Q$ and non-secret states $Q_{NS} \subset Q$, $Q_S \cap Q_{NS} = \emptyset$, $G$ is current-state opaque if for every string that leads from an initial state to a secret state, there exists a string that leads from an initial state to a non-secret state and these two strings have identical projections.

\[ \forall s \in L(G) \text{ such that } \delta(q_0, s) \in Q_S, \exists t \in L(G) \text{ such that } \delta(q_0, t) \in Q_{NS} \text{ and } P_{\Sigma_s}(s) = P_{\Sigma_s}(t) \]

### 2.3 Enforcing opacity through supervisory control

It is natural to want to ensure that a given system satisfies our well-defined notion of opacity. This problem has been addressed in the literature in many different forms, in this paper we name it the opacity control problem. This problem is defined by a set of general parameters (Bryans et al. 2008; Dubreil et al. 2010):

1. the set of events that are observable and controllable for the supervisor ($\Sigma_s$ and $\Sigma_c$ respectively);
2. the set of events that are observable by the adversary ($\Sigma_a$);
3. the secret and type of opacity to enforce; and
4. the adversary’s knowledge of any supervisory control policy.

Since the supervisor and adversary each have a set of observable events, we label these $\Sigma_s$ and $\Sigma_a$ respectively to avoid ambiguity. The most general formulation of the opacity control
problem assumes no relationship between $\Sigma_s$, $\Sigma_c$, and $\Sigma_a$. Other reasonable relationships to assume between these three sets is that the adversary sees a subset of the events that the supervisor does, $\Sigma_a \subseteq \Sigma_s$, that the supervisor can only control events that it can see, $\Sigma_c \subseteq \Sigma_s$, and that the system contains only events that are controllable and observable by the supervisor, $\Sigma_c = \Sigma_s = \Sigma$. Regardless of the specific relationship between event sets, the most principled way to approach the opacity control problem is to consider an adversary that is aware of the supervisor’s control policy and thereby avoid any appeals to security through obscurity. In this paper we use the formulation given in Problem 1.

**Problem 1 (Opacity Control Problem)** Given a DFA $G = (Q, \Sigma, \delta, q_0, Q_m)$ where $Q_m$ is the set of secret states, an alphabet of events the supervisor can observe $\Sigma_s \subseteq \Sigma$ and an alphabet of events the supervisor can control $\Sigma_c \subseteq \Sigma_s$. Given an supervisor-aware adversary and an alphabet of events that it can observe $\Sigma_a \subseteq \Sigma_s$. Find a supervisor $S$ such that the closed-loop behaviour of the plant composed with the supervisor, $S/G$, is current-state opaque with respect to the alphabet observable by the adversary, $\Sigma_a$, the set of secret states $Q_S = Q_m$, and the set of non-secret states $Q_{NS} = Q \setminus Q_m$.

The naïve solution for this formulation is to iteratively calculate the adversary’s view of the plant, apply supervisory control, and then check the opacity of the controlled system until a fixed point is reached. This approach has been noted by multiple authors and dismissed as very computationally expensive due to the fact that an observer automaton must be calculated for every iteration (Dubreil et al. 2010; Saboori and Hadjicostis 2012).

### 2.4 Other approaches to the opacity control problem

The opacity control problem has been addressed in the literature under different assumptions. In some works the adversary is unaware of the supervisor, while in others the adversary has complete knowledge of the supervisor’s control policy. We consider the latter case, since this introduces the requirement to iteratively calculate the supervisor’s control policy and the adversary’s estimate of the plant; Table 1 summarizes the characteristics of these works.

Addressing language-based opacity, Badouel et al. presented an iterative method to design a supervisor that guaranteed concurrent opacity assuming there are no uncontrollable events in the plant (Badouel et al. 2007). Takai and Oka proposed a method with one-step convergence under the strong assumption that, for any pair of indistinguishable strings, any uncontrollable and observable event that can occur after one string can occur after the other (Takai and Oka 2008). Finally, Ben-Kalefa and Lin iteratively enforced opacity and controllability for the language generated by the plant (Ben-Kalefa and Lin 2011). This is the naïve approach applied to languages and still requires the construction of a supervisor to enforce the final language produced.

Considering current-state opacity, Yin and Lafortune presented the *All-Inclusive Controller for Opacity* (Yin and Lafortune 2015) and the *All Enforcement Structure* (Yin and Lafortune 2016), which both embed a game between supervisor and plant in a bipartite transition structure where the supervisor’s goal is to enforce the specified property and the plant’s is to violate the specified property. More similar to our approach to the opacity control problem are *condensed state estimates*, Dubreil et al. (2010), and the *augmented l-observer*, Tong et al. (2018).
Dubreil et al. showed that solving the opacity control problem under full observation induces a general solution to the opacity control problem when $\Sigma_a \subseteq \Sigma_s$ (Dubreil et al. 2010). Their approach to enforcing opacity through supervisory control produces an automaton whose states track the plant’s actual state as well as condensed state estimates, which contain the set of states that the adversary believes the plant could be in if the last transition it observed was the last transition that occurred in the plant. From the condensed state estimates, the supervisor then reasons about “loosing paths,” which are the traces from the current state that would lead to the disclosure of a secret (Dubreil et al. 2010).

Our approach to the opacity control problem is inspired by the condensed state estimates method, which we improve upon in two ways. First, the parallel composition between the plant and an observer represent the adversary’s beliefs about the plant more intuitively than condensed state estimates do. Second, our representation of the adversary’s beliefs removes the need for unrolling condensed state estimates to determine whether a particular string will disclose a system secret.

Finally, Tong et al.’s augmented I-observer is the parallel composition between the plant and the observer representing the adversary’s view of the plant (Tong et al. 2018). The supremal G-opaque sublanguage – a language-based property equivalent to current-state opacity – is computed and a supervisor synthesized to enforce it as a specification (Tong et al. 2018, Definition 7). This process iterates until the system’s generated language is G-opaque, i.e., the system is current-state opaque. The augmented I-observer method allows $\Sigma_a$ and $\Sigma_s$ to be incomparable, but Tong et al. restricted the majority of their discussion to non-supervisor-aware adversaries. With a supervisor-aware adversary, this generality comes with the additional computational cost of updating both the plant and augmented I-observer at every step of the

| Reference                        | Approach                                         | Event Sets | Opacity Type          |
|----------------------------------|--------------------------------------------------|------------|-----------------------|
| Badouel et al. (2007)            | Sub-lattice of kernels                           | $\Sigma_a \subseteq \Sigma_s$ | Language-based        |
| Takai and Oka (2008)             | Supremal closed, controllable, and opaque sublanguage | $\Sigma_a \subseteq \Sigma_s$ | Language-based        |
| Ben-Kalefa and Lin (2011)        | Supremal controllable, observable, and opaque language | $\Sigma_a \subseteq \Sigma_s$ | Language-based        |
| Dubreil et al. (2010)            | Condensed state estimates                        | $\Sigma_a \subseteq \Sigma_s$ | Current-state         |
| Yin and Lafortune (2015)         | All-Inclusive Controller for Opacity             | $\Sigma_a \subseteq \Sigma_s$ | Current-state         |
| Yin and Lafortune (2016)         | All-Enforcement Structure                        | $\Sigma_a = \Sigma_s$      | Current-state         |
| Tong et al. (2018)               | Augmented I-observer                             | $\Sigma_a \subseteq \Sigma_s$ | Current-state         |
| This paper                      | Plant behaviour composed with the adversary’s view | $\Sigma_a \subseteq \Sigma_s$ | Current-state         |
iteration. With the reasonable assumption that $\Sigma_a \subseteq \Sigma_s$, our method is able to simply refine the parallel composition of the plant and adversary’s view instead.

3 Relating observers: subautomata versus subobservers

The key to our method is that we are able to relate each successive observer automaton to the one that came before it. To begin, when supervisory control is applied to enforce a regular-language specification for an automaton $G$, the controlled system can be modelled as another automaton, $G'$. We begin, therefore, with an understanding that there is a relationship between $G$ and $G'$. This is the subautomaton relationship, a general relationship that goes beyond supervisory control (Cassandras and Lafortune 2008, p. 86).

**Definition 5 (Subautomaton)** We say that $G' = (Q', \Sigma, \delta_{G'}, q'_0, Q_m')$ is a subautomaton of $G = (Q, \Sigma, \delta_G, q_0, Q_m)$, denoted by $G' \subseteq G$, if

$$\delta_{G'}(q'_0, s) = \delta_G(q_0, s) \quad \forall s \in L(G')$$

The subautomaton relationship is a strong form of correspondence between two automata. It allows us to reason, for example, that the states of $G'$ are a subset of those in $G$, that the initial state in $G'$ is the same as in $G$, and, more generally, that we can match the states in $G'$ with a corresponding state in $G$ (Cassandras and Lafortune 2008, p. 86). This relationship can be very helpful when reasoning about the language produced by each automaton or about termination conditions for certain algorithms.

When the specification of legal behaviour is a subautomaton of $G$, the resulting automaton $G'$ will also be a subautomaton of $G$. Although the idea of a specification is not directly applicable to the opacity control problem, we demonstrate in Section 5.3 how our proposed method inherently ensures that supervisory control produces a subautomaton at each iteration.

Beyond the relationship between the uncontrolled and controlled plant, we are interested in how an adversary’s observations of a system evolve as supervisory control is enacted. We would like to know whether or not we can establish a relationship between the observer automata $H'$ and $H$, transformations of $G'$ and $G$ respectively with respect to the same alphabet of observable events (Fig. 1). Does the subautomaton relationship survive the transformation; is $H'$ a subautomaton of $H$? More formally, is it true, given automata $G, G', H := T_{\Sigma_o}(G)$ and $H' := T_{\Sigma_o}(G')$, that

$$G' \subseteq G \implies H' \subseteq H?$$

![Fig. 1](image-url) We wish to characterize the relationship that exists between $H'$ and $H$. 

© Springer
Unfortunately this statement is not true because $H'$ and $H$ will have their states coming from $2^Q$ and $2^Q$ respectively, allowing for the possibility that states in $H'$ will not properly match up with states in $H$ (Fig. 2). Although the transition graph of any subautomaton is necessarily a subgraph of the parent automaton’s transition graph, the same is not true for automaton related by the subobserver property since states may be “split” from $H$ to $H'$. Even if we avoid this, the state labels in $H$ and $H'$ are semantically meaningful, namely they encode the adversary’s belief about the state of the system, and these semantics would be lost if we allowed, for example, the state labelled $\{5\}$ to be matched with the state labelled $\{5,6\}$.

All is not lost, however. Given our insistence that state labels carry semantic meaning, we can relax the definition of subautomaton to allow for this and produce a relationship that is suitable to relate two observers together. We call this the subobserver relationship, Definition 6.

**Definition 6 (Subobserver)** We say that $G' = (Q', \Sigma, \delta_{G'}, q'_0, Q'_m)$ is a subobserver of $G = (Q, \Sigma, \delta_G, q_0, Q_m)$, denoted $G' \subseteq G$, if

$$\delta_{G'}(q'_0, s) \subseteq \delta_G(q_0, s) \quad \forall \ s \in L(G').$$

Because we parallel the definition of a subautomaton in our definition of a subobserver it immediately follows that all subautomaton relationships are also subobserver relationships and that the converse is not true. We claim that the subobserver property does complete the relationships between automata $G, G', H,$ and $H'$ seen in Fig. 1.

Before proving this result formally, we begin by establishing some technical notes. To begin, throughout our proofs we will define automata in terms of $G$. Specifically, $G'$ is a

![Fig. 2 Counterexample showing that $H'$ is not a subautomaton of $H$. Initial states are denoted by a small arrow; marked states are denoted with a double ring. Observable events have a solid arrow and unobservable events have a dashed arrow. Controllable events have marks across their arrows.](image-url)
subautomaton of $G$ and $H' := T_{\Sigma_n}(G')$ and $H := T_{\Sigma_n}(G)$ are observer automata with our characteristic marking.

$$
G = (Q, \Sigma, \delta, q_0, Q_m) \quad H = T_{\Sigma_n}(G) = (2^Q, \Sigma, \delta_H, q_{oh}, Q_{mh})
$$

$$
G' = (Q', \Sigma, \delta', q_0, Q'_m) \quad H' = T_{\Sigma_n}(G') = (2^{Q'}, \Sigma, \delta_H, q_{oh'}, Q_{m'h'})
$$

Next, we define a state’s $\epsilon$-reach as the set of states that can be reached from it via the empty string (Definition 7) (Cassandras and Lafortune 2008, p. 71).

**Definition 7 ($\epsilon$-reach)** For an automaton $G$ and projection $P_{\Sigma_n}$, we call the set of states that can be reached from state $q$ via a string whose projection is $\epsilon$ as the $\epsilon$-reach of $q$. Formally,

$$
e_{R_G}(q) := \{ q' \in Q | \exists s \in \Sigma^*, P_{\Sigma_n}(s) = \epsilon, \delta_G(q, s) = q' \}.
$$

By definition a state is always in its own $\epsilon$-reach, $q \in eR(q)$.

We also show that if $G' \subseteq G$ then a state’s $\epsilon$-reach in $G'$ is a subset of that state’s $\epsilon$-reach in $G$.

**Lemma 1 (The $\epsilon$-reach of any state in a subautomaton is a subset of that state’s $\epsilon$-reach in the parent automaton)** Given automata $G = (Q, \Sigma, \delta_G, q_0, Q_m)$ and $G' = (Q', \Sigma, \delta_{G'}, q_0', Q'_m)$. Then

$$
G' \subseteq G \implies e_{R_{G'}}(q) \subseteq e_{R_G}(q) \forall q \in Q'.
$$

**Proof** To begin, $G' \subseteq G$ implies that $Q' \subseteq Q$. We can therefore determine the $\epsilon$-reach of state $q$ with respect to both $G$ and $G'$ for all $q \in Q'$.

$$
\forall q \in Q' \exists s \in \Sigma^*, P_{\Sigma}(s) = \epsilon, \delta_{G'}(q', s) = q'.
$$

We are now ready to prove that the relationship between the observer automata $H$ and $H'$ (the dashed line in Fig. 1) is the subobserver relationship defined in Definition 6.

**Theorem 1 (The transformation of a subautomaton is itself a subobserver)** Given automata $G$ and $G'$ and their respective transformations, $H := T_{\Sigma_n}(G)$ and $H' := T_{\Sigma_n}(G')$. Then

$$
G' \subseteq G \implies H' \subseteq H.
$$

**Proof** The proof proceeds by induction.

**Base case:** $s = \epsilon$, $|s| = 0$.

$$
q_{oh'} = e_{R_{G'}}(q_0) \subseteq e_{R_G}(q_0)
$$

(by Lemma 1)
Inductive Hypothesis: Suppose \( s \in L(H') \), \( \|s\| = n \), and \( \bar{Q}' = \delta_{H'}(q_{00}', s) \subseteq \delta_{H}(q_{00}, s) = \bar{Q} \).

Inductive Step: Now we show that for \( s\sigma \in L(H') \), \( \|s\sigma\| = n + 1 \), then 
\[
\delta_{H'}(q_{00}', s\sigma) \subseteq \delta_{H}(q_{00}, s\sigma).
\]

We proceed by translating the set of states \( \delta_{H'}(q_{00}', s\sigma) \) into set builder notation and then adding well-chosen states to produce the set of states \( \delta_{H}(q_{00}, s\sigma) \).

\[
\delta_{H'}(q_{00}', s\sigma) = \delta_{H'}(\bar{Q}', \sigma) = \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \cup \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \cup \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \delta_{H}(\bar{Q}', \sigma) \subseteq \delta_{H}(q_{00}, s\sigma),
\]

by definition in our IH

\[
\delta_{H'}(q_{00}', s\sigma) = \delta_{H'}(\bar{Q}', \sigma) = \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \cup \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G'}(q', \sigma)) \cup \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \bigcup_{q' \in \bar{Q}'} e_{R_G}(\delta_{G}(q', \sigma)) \subseteq \delta_{H}(\bar{Q}', \sigma) \subseteq \delta_{H}(q_{00}, s\sigma),
\]

by definition in our IH

Having proven Theorem 1, we can relate our four automata from Fig. 1: \( G' \) is a subautomaton of \( G \), \( H' \) and \( H \) are transformations of \( G' \) and \( G \) respectively, and \( H' \) is a subobserver of \( H \).

4 Updating the observer automaton

An adversary’s view of a plant is tied to the plant’s structure, so we explicitly link the adversary’s plant estimate and the plant’s actual state through the parallel composition of their respective automata (Definition 8 (Cassandras and Lafortune 2008, p. 80)).

Definition 8 (Parallel Composition) Given two automata \( G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1}) \) and \( G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2}) \), their parallel composition is defined as:

\[
G_1 \parallel G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{12}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})
\]

where \( \delta_{12}((q_1, q_2), \sigma) = \begin{cases} 
(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \\
(\delta_1(q_1, \sigma), q_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \\
(\delta_2(q_1, \sigma), q_2) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \\
\text{undefined} & \text{otherwise}
\end{cases} \)
The parallel composition $G \parallel H$, contains all of the information required to track the evolution of the plant’s behaviour and the adversary’s beliefs about the plant. We will denote states in $G \parallel H$ using the notation $q_A$ where $q$ is a state in $G$ and $A$ is a state in $H$. It is useful to observe that a state $q_A$ is accessible in $G \parallel H$ if and only if there is a string $s \in L(G)$ that leads to state $q$ in $G$ and whose projection leads to estimate $A$ in $H$ (Lemma 2). As a corollary to this Lemma, if a state $q_A$ is accessible in $G \parallel H$ then the state $q$ is an element of the estimate $A$.

**Lemma 2** Given the parallel composition $G \parallel H$ where $G = (Q, \Sigma, \delta_G, q_0, Q_m)$ is a DFA and $H : = T_{\Sigma}(G)$ is an observer automaton. A state $q_A$ is accessible in $G \parallel H$ if and only if

$$(\exists s \in L(G)) (\delta_G(q_0, s) = q) (\{q' \in Q \mid (\exists s' \in L(G)) P_{\Sigma}(s) = P_{\Sigma}(s') (\delta_G(q_0, s') = q')\} = A).$$

**Proof** See the Appendix.

The parallel composition has strong properties; first, it follows that $L(G \parallel H) = L(G)$ (Jirásková and Masopust 2012). More interesting is that the accessible states of $T_{\Sigma}(G \parallel H)$ are pairwise disjoint sets of states of $G \parallel H$, i.e., every state in $G \parallel H$ belongs to one and only one state in $T_{\Sigma}(G \parallel H)$. An automaton that satisfies this property is a *state-partition automaton* (SPA). The SPA property allows for simple refinement operations, but we will not explicitly use it as Cho and Marcus used it, Cho and Marcus (1989), since exploiting the SPA property would require a second transformation operation, $T_{\Sigma}(G \parallel H)$, and would introduce additional preprocessing overhead to our algorithms.

### 4.1 Using the subobserver property

We are given the parallel composition of a plant, $G$, and an adversary’s view of that plant, $H$. Given that the plant’s behaviour has been restricted, the **REFINE** algorithm (Algorithm 1) returns the parallel composition of the restricted plant and the adversary’s updated view of the plant without explicitly calculating this adversary view. To begin, we remove states from the parallel composition that should be made inaccessible. Next, we account for the adversary’s ability to reason about the restricted plant. If there is a string $s$ in $L(G)$ that is no longer possible in $G'$, then the adversary cannot confuse other strings in $L(G')$ with $s$. We must therefore remove the state that $s$ leads to in $G$ from estimates where the adversary confused $s$ with other strings.

To simplify our reasoning, we assume that the plant $G$ and the parallel composition $G \parallel H$ have isomorphic state transition diagrams. Although this is not always the case, we will demonstrate in Section 5.3 that the parallel composition operation inherently creates an automaton $\hat{G}$ that is language equivalent to $G$ and whose state transition diagram is isomorphic with that of $G \parallel H$.

### 4.2 Technical developments

Before proving that the **REFINE** algorithm is correct, we need to develop a few ideas that will be necessary to connect our state-based formulation of opacity with the string-based confusion of the adversary. First, we know that **REFINE** is guaranteed to produce a
subautomaton of $G$ in the first elements of the states of $M$ because it removes states from $G \parallel H$.

**Lemma 3** Given the parallel composition $G \parallel H$, where $G = (Q, \Sigma, \delta_G, q_0, Q_m)$ is a DFA and $H := T_{\Sigma'}(G)$ is an observer automaton, and $\Delta$ is a list of states to remove from $G \parallel H$. Then the first elements of the states remaining in $G \parallel H$ represent a subautomaton of $G$.

**Proof** See the Appendix.

Second, we note that a subautomaton $G'$ can be defined by its parent automaton $G$ and the set of strings that have been removed from $L(G)$. We define an equivalence relation $\rho$ on $\Sigma^*$, which relates strings if they are both in $L(G')$, were both removed from $L(G)$, or were never in $L(G)$.

**Definition 9** The binary relation $\rho_{G,G'}$ on $\Sigma^*$ is defined as $(s, s') \in \rho_{G,G'}$ if and only if

$$(s, s' \in L(G')) \lor (s, s' \notin L(G) \setminus L(G')) \lor (s, s' \in \Sigma^* \setminus L(G)).$$

Third, in Refine we treat strings the same if they lead to the same state in $G$ and the same estimate in $H$. We define an equivalence relation $\text{match}_{\Sigma'}$ on $\Sigma^*$ to capture this, where strings are related if they will receive identical treatment from Refine.

**Definition 10** Given a DFA $G$ and observer automaton $H := T_{\Sigma'}(G)$. The binary relation $\text{match}_{\Sigma'}$ on $\Sigma^*$ is defined as

$$(s, s') \in \text{match}_{\Sigma'} \iff (s \equiv s' \mod \text{eq}(G)) \land (P_{\Sigma'}(s) \equiv P_{\Sigma'}(s') \mod \text{eq}(H))$$

and if $s \notin L(G)$ then $(s, s') \in \text{match}_{\Sigma'}$ if and only if $s' \notin L(G)$ as well.

Algorithm 1 The Refine algorithm
We can show that the partition defined by \( \text{match}_{\Sigma_o} \) is equivalent to the partition defined by \( \text{eq}(G \parallel H) \).

**Lemma 4** Given a DFA \( G = (Q, \Sigma, \delta, q_0, Q_m) \), a set of observable events \( \Sigma_o \subseteq \Sigma \), and an observer automaton \( H := T_{\Sigma_o}(G) \). Then

\[
\text{match}_{\Sigma_o} = \text{eq}(G \parallel H).
\]

**Proof** We will show that two strings \( s, s' \in L(G) \) are related in \( \text{match}_{\Sigma_o} \) if and only if they are related in \( \text{eq}(G \parallel H) \).

\[
(s, s') \in \text{match}_{\Sigma_o} \iff (s \equiv s' \mod \text{eq}(G)) \land (P_{\Sigma_o}(s) \equiv P_{\Sigma_o}(s') \mod \text{eq}(H)) \quad \text{by Definition 10}
\]

\[
\iff \delta_G(q_0, s) = \delta_G(q_0, s') \land \delta_H(A_0, P_{\Sigma_o}(s)) = \delta_H(A_0, P_{\Sigma_o}(s')) \quad \text{by Definition 1}
\]

\[
\iff \delta_H |_{H}([q_0], s) = \delta_H |_{H}([q_0], s') \quad \text{by Definition 8}
\]

\[
\iff s \equiv s' \mod \text{eq}(G \parallel H) \quad \text{by Definition 1}
\]

Finally, we can build on Lemma 4 by showing that if \( \text{match}_{\Sigma_o} \) refines \( \rho_{G,G'} \), then we can produce \( H' \) by relabelling states in \( H \) (i.e., the state transition graph of \( H' \) is guaranteed be a subgraph of \( H \)'s state transition graph). This condition implicitly enforces that \( \text{REFINE} \) must remove states from \( G \parallel H \) and thereby treat strings that lead to the same state and estimate identically. This is a reasonable restriction since we are considering state-based formulations of opacity and because alternate automaton representations of a language can be used if strings currently related by \( \text{match}_{\Sigma_o} \) should be treated differently.

**Lemma 5** Given DFAs \( G = (Q, \Sigma, \delta, q_0, Q_m) \), a set of observable events \( \Sigma_o \subseteq \Sigma \), and an observer automaton \( H = T_{\Sigma_o}(G) \). If \( G' \) is a subautomaton of \( G \) and \( H' = T_{\Sigma_o}(G') \), then

\[
(\forall s, s' \in L(H))(s \equiv s' \mod \text{eq}(H)) \land \text{match}_{\Sigma_o} \subseteq \rho_{G,G'} \implies s \equiv s' \mod \text{eq}(H').
\]

**Proof** The proof proceeds by contradiction. Assume that \( \text{match}_{\Sigma_o} \subseteq \rho_{G,G'} \) and that there exists strings \( s \) and \( s' \) in \( L(H) \) such that \( s \equiv s' \mod \text{eq}(H) \) but \( s \not\equiv s' \mod \text{eq}(H') \).

\[
s \equiv s' \mod \text{eq}(H) \implies \delta_H(A_0, s) = \delta_H(A_0, s')
\]  \hspace{1cm} (1)

We can enumerate exactly those states that appear in \( \delta_H(A_0, s) \), and therefore that appear in \( \delta_H(A_0, s') \) as well.

\[
\delta_H(A_0, s) = \{ q \in Q \mid (\exists t \in L(G))(P_{\Sigma_o}(t) = s)\delta_G(q_0, t) = q \}
\]

\[
= \{ q \in Q \mid (\exists t \in L(G'))(P_{\Sigma_o}(t) = s)\delta_G(q_0, t) = q \} \cup
\]

\[
\{ q \in Q \mid (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_o}(t) = s)\delta_G(q_0, t) = q \}
\]

\[
= \delta_H(A_0, s) \cup \{ q \in Q \mid (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_o}(t) = s)\delta_G(q_0, t) = q \}
\]

For notational convenience, we will denote the second term as

\[
Q_o(s) := \{ q \in Q \mid (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_o}(t) = s)\delta_G(q_0, t) = q \}.
\]  \hspace{1cm} (2)

These sets of states are disjoint, which can be shown by contradiction. Assume that there exists a state \( q \in Q \) that belongs to the intersection of these two sets.
\( q \in \delta_H(A_0, s) \cap Q_\rho(s) \implies (\exists t \in L(G'), t' \in L(G) \setminus L(G')) \\
(\delta_G(q_0, t) = \delta_G(q_0, t') = q) \\
(P_{\Sigma_\psi}(t) = P_{\Sigma_\psi}(t') = s) \text{ definition of } \delta_H(A_0, s) \text{ and } Q_\rho(s) \\
\implies (\exists t \in L(G'), t' \in L(G) \setminus L(G')) \\
(\delta_G(q_0, t) = \delta_G(q_0, t') = q) \\
(\delta_\rho(A_0, P_{\Sigma_\psi}(t)) = \delta_\rho(A_0, P_{\Sigma_\psi}(t'))) \quad \text{since } P_{\Sigma_\psi}(t) = P_{\Sigma_\psi}(t') \\
\implies (\exists t \in L(G'), t' \in L(G) \setminus L(G')) \\
(t \equiv t' \mod eq(G)) \\
(P_{\Sigma_\psi}(t) \equiv P_{\Sigma_\psi}(t') \mod eq(H)) \quad \text{by Definition 1} \\
\implies (\exists t \in L(G'), t' \in L(G) \setminus L(G')) \\
((t, t') \in \text{match}_{\Sigma_\psi}) \quad \text{by Definition 10} \\
\implies (\exists t \in L(G'), t' \in L(G) \setminus L(G')) \\
(t, t' \in L(G') \lor t, t' \in L(G) \setminus L(G')) \quad \text{since match}_{\Sigma_\psi} \leq \rho_{G,G'} \\
\end{equation*}

Since \( L(G') \) and \( L(G) \setminus L(G') \) are disjoint, this is a contradiction. We can therefore say that \( \delta_H(A_0, s) \cap Q_\rho(s) = \emptyset \) and that

\[
\delta_H(A_0, s) = \delta_H(A_0, s) \setminus Q_\rho(s). \tag{3}
\]

For the sake of contradiction, we assumed that \( s \not\equiv s' \mod eq(H') \). By Definition 1 this implies that \( \delta_H(A_0, s) \not\equiv \delta_H(A_0, s') \). From Eq. 3 we know for \( s \) and \( s' \) that

\[
\delta_H(A_0, s) = \delta_H(A_0, s) \setminus Q_\rho(s) \\
\delta_H(A_0, s') = \delta_H(A_0, s') \setminus Q_\rho(s')
\]

and since we have reasoned that \( \delta_H(A_0, s) = \delta_H(A_0, s') \), Eq. 1, this implies that \( Q_\rho(s) \neq Q_\rho(s') \).

\[
Q_\rho(s) \neq Q_\rho(s') \implies (\exists q^* \in Q)(q^* \in Q_\rho(s))(q^* \not\in Q_\rho(s')) \quad \text{WLOG, since } eq(H) \text{ is symmetric} \\
\implies (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_\psi}(t) = s)(\delta_G(q_0, t) = q^*) \\
\quad (\exists t' \in L(G) \setminus L(G'))(P_{\Sigma_\psi}(t') = s')(\delta_G(q_0, t') = q^*) \quad \text{by Eq. 2} \\
\implies (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_\psi}(t) = s)(\delta_G(q_0, t) = q^*) \\
\quad (\exists t' \in L(G) \setminus L(G'))(P_{\Sigma_\psi}(t') = s')(\delta_G(q_0, t') = q^*) \quad \text{since } L(G') \cap L(G) \setminus L(G') = \emptyset \\
\implies (\exists t \in L(G) \setminus L(G'))(P_{\Sigma_\psi}(t) = s)(P_{\Sigma_\psi}(t') = s') \\
\quad (t \equiv t' \mod eq(G)) \quad \text{by Definition 1} \\
\implies (\exists t \in L(G) \setminus L(G'), t' \in L(G')) \\
\quad (t \equiv t' \mod eq(G \parallel H)) \quad \text{since } s \equiv s' \mod eq(H) \\
\implies (\exists t \in L(G) \setminus L(G'), t' \in L(G')) \\
\quad (t, t') \in \text{match}_{\Sigma_\psi} \quad \text{by Lemma 4} \\
\implies (\exists t \in L(G) \setminus L(G'), t' \in L(G')) \\
\quad (t, t' \in L(G') \lor t, t' \in L(G) \setminus L(G')) \quad \text{since match}_{\Sigma_\psi} \leq \rho_{G,G'} \\
\end{equation*}

Again, since we know that \( L(G') \) and \( L(G) \setminus L(G') \) are disjoint, this is a contradiction. This implies that \( Q_\rho(s) = Q_\rho(s') \), which further implies that \( s \equiv s' \mod eq(H') \). We have therefore proved that

\[
(\forall s, s' \in L(H))(s \equiv s' \mod eq(H)) \land \text{match}_{\Sigma_\psi} \leq \rho_{G,G'} \implies s \equiv s' \mod eq(H').
\]

Lemma 5 guarantees us that for any state in \( H \)—cell in the partition defined by \( eq(H) \)—we can produce the corresponding state in \( H' \) by relabelling that state. It goes beyond this too, by telling us that this relabelling removes exactly those states in \( Q_\rho \) from the state label in \( H \).
4.3 Proof of correctness for the REFINE algorithm

As a first point, the REFINE algorithm is guaranteed to terminate because it works by removing and relabelling states in the input DFA, $G \parallel H$. For the first loop, the number of states and transitions to be inspected is finite and each state and transition can only be inspected once, therefore the loop terminates. For the second loop, there are a finite number of states to assess for accessibility and no changes are made to the structure of $M$. For the third loop, there are a finite number of states to relabel and relabelling a states in $M$ never requires relabelling another state in $M$, therefore the loop terminates. Finally, taking the accessible portion of $M$ requires assessing the accessibility of a finite number of states, so this operation also terminates.

Secondly, we claim that the REFINE algorithm uses the subobserver relationship to calculate $G' \parallel H'$ without computing $H'$ explicitly. This is formalized as Theorem 2.

**Theorem 2** Given the parallel composition $G \parallel H$ and list of states $\Delta$ where: $G = (Q, \Sigma, \delta_G, q_0, Q_m)$ is a DFA; $H := T_{\Sigma_i}(G)$ is an observer automaton; and $\Delta$ is a list of states in $G \parallel H$. Denote by $G'$ the automaton that results in the first elements of $G \parallel H$ once the states in $\Delta$ are made inaccessible. Then the REFINE algorithm produces $G' \parallel H'$ where $H' := T_{\Sigma_i}(G')$.

**Proof** We show that, given $G \parallel H$, where $H$ is the transformed automaton $T_{\Sigma_i}(G)$, and $\Delta$, a list of states to remove from $G \parallel H$ that produces $G'$ in the first elements of the states in $G$, the REFINE algorithm constructs $G' \parallel H'$.

We assume that $\Delta$ is not equal to the set of states in $G \parallel H$, otherwise $G'$ is the null automaton and we simply need to return $G' \parallel H' = \text{null}$. For the case where $\Delta$ is not equal to the set of states in $G \parallel H$, we reason about the elements of the states separately, since these correspond to the states of $G'$ and $H'$.

First, we argue that the first loop removes the correct states from $M$ to make the states in $\Delta$ inaccessible. For the purposes of this proof, we define a new set $\Delta_{Q}^{\text{All}}$ as the set of all states that were added to $\Delta_Q$. If we wanted to include this set in our implementation of REFINE, we could an operation after Line 3 and after Line 8 where the state that was added to $\Delta_Q$ is also added to $\Delta_{Q}^{\text{All}}$. Since states are never removed from $\Delta_{Q}^{\text{All}}$, it will contain all of the states that were at one point in $\Delta_Q$ and were removed from the automaton $M$. This allows us to reason that after the first loop:

$$\frac{q}{A} \in \Delta_{Q}^{\text{All}} \iff \left( \frac{q}{A} \in \Delta \right) \lor \left( \frac{q}{A} \text{ leads uncontrollably to a state in } \Delta_{Q}^{\text{All}} \right)$$

1. $\implies$. States are in $\Delta_{Q}^{\text{All}}$ for only two reasons. The first is if they are added to $\Delta_Q$ at Line 3, in which case they are in $\Delta$. The second is if they are added to $\Delta_Q$ at Line 8, in which case they lead uncontrollably to a state in $\Delta_{Q}^{\text{All}}$. So every state in $\Delta_{Q}^{\text{All}}$ meets one of the two conditions in the consequent.

2. $\impliedby$. First, Line 3 makes $\Delta_Q$ equal to $\Delta$ before beginning the loop, which means that all states in $\Delta$ are also in $\Delta_{Q}^{\text{All}}$. Second, for any state $\frac{q}{A}$ that leads uncontrollably to a state $\frac{q'}{A'}$ in $\Delta_{Q}^{\text{All}}$, there is an uncontrollable transition that leads from $\frac{q}{A}$ to $\frac{q'}{A'}$. When the iteration for the state $\frac{q}{A'}$ is reached in the loop, that transition will be identified as uncontrollable and $\frac{q}{A}$ will be added to $\Delta_Q$ at Line 8 and therefore will be added to $\Delta_{Q}^{\text{All}}$ as well.
Since states are never removed from $\Delta^\text{All}_Q$, these two lines of reasoning guarantee that all states in $\Delta$ are in $\Delta^\text{All}_Q$ and that all states that lead uncontrollably to a state in $\Delta_Q$ are in $\Delta^\text{All}_Q$ as well.

After the first loop is complete, $\Delta^\text{All}_Q$ contains exactly those states that must be removed to make the states in $\Delta$ inaccessible. These are exactly the states that are removed from $M$, since any state that appears in $\Delta^\text{All}_Q$ was in $\Delta_Q$ and was therefore removed from $M$ at Line 10. REFINE doesn’t remove any other states from $M$. Therefore the only states that have been removed from $M$ are those that were necessary to make the set of states in $\Delta$ inaccessible. We denote the first element of the remaining states in $M$ as an automaton $G'$.

Second, because we created $G'$ by removing states from $M = G \parallel H$, we know by Lemma 3 that $G' \subseteq G$. Since $G' \subseteq G$, $H := T_\Sigma(G)$, and $H' := T_\Sigma(G')$, we know from Theorem 1 that $H' \subseteq H$. We argue that because $H' \subseteq H$, the estimates related to some strings need to be updated and that this can be accomplished by relabelling the states. We also argue that because the strings removed from $L(G)$ to produce $L(G')$ were determined by a set of states in $G \parallel H$, we have that $\text{match}_\Sigma \leq \rho_{G,G}$ by Lemma 4 and therefore relabelling the estimates from $H$ is all that is required to produce $H'$ (Lemma 5).

The second loop in REFINE produces a list of states $A_r(A)$ for each estimate $A$. We claim that a state $q$ will be added to $A_r(A)$ if and only if $q$ was in the estimate $A$ to begin with and there are no longer any strings in $L(G')$ that lead to $q$ and which share projections with strings leading to the other states in $A$:

$$q \in A_r(A) \iff (q \in A) \land ((\forall s \in L(G) \mid \delta_G(q_0, s) = q \land \delta_H(A_0, P_\Sigma(s)) = A)(s \not\in L(G'))).$$

We will reason about the two directions separately.

1. $\implies$. If $q \in A_r(A)$ then it was added at Line 13 because the state $\frac{q}{A}$ was in $G \parallel H$ but no longer accessible in $M$ at Line 11. Because the state $\frac{q}{A}$ was accessible in $G \parallel H$, Lemma 2 tells us that $q \in A$, satisfying the consequent’s first expression.

   For the consequent’s second expression, we will proceed by contradiction. Assume for the sake of contradiction that there exists a string $s \in L(G)$ such that

   $$\delta_G(q_0, s) = q \land \delta_H(A_0, P_\Sigma(s)) = A$$

   and that this string $s$ is also in $L(G')$. Since $s$ is in $L(G')$ and since the first elements of states remaining in $M$ after the first loop exactly reflect $G'$, this means that the state to which $s$ leads has not been removed from $M$ and hence $\frac{q}{A}$ remains accessible in $M$ at Line 12. We therefore know that $q$ would not have been added to $A_r(A)$ at Line 13, which is a contradiction. Therefore $q \in A_r(A)$ implies the consequent’s second expression as well.

2. $\impliedby$. Because $(q \in A) \land ((\forall s \in L(G) \mid \delta_G(q_0, s) = q \land \delta_H(A_0, P_\Sigma(s)) = A)$ we have that the state $\frac{q}{A}$ was accessible in $G \parallel H$. The last expression in the antecedent tells us, however, that any string $s$ that led to $\frac{q}{A}$ in $G \parallel H$ is not in $L(G')$. Since the first elements of states remaining in $M$ after the first loop exactly reflect $G'$, this means that $\frac{q}{A}$ is no longer accessible in $M$ at Line 12. The state $q$ is therefore added to $A_r(A)$ at Line 13. Since states are never removed from $A_r(A)$, this guarantees that $q \in A_r(A)$. 

 Springer
Third, the third loop correctly updates all of the estimates in $M$ because all states in $M$ are visited and because all states in $M$ sharing an estimate must be updated in the same way (Lemma 5).

To conclude, after the $\text{REFINE}$ algorithm has completed its work, the resulting automaton $M$ is equal to $G' \parallel H'$ such that

$$\delta_M\left(\frac{q_0}{A_0}, s\right) = \frac{q}{A} \iff \delta_{G'}(q_0, s) = q \land \delta_{H'}(A_0, P_{\Sigma}(s)) = A.$$ 

### 4.4 Time complexity analysis of the $\text{REFINE}$ algorithm

We tie the asymptotic time complexity of the $\text{REFINE}$ algorithm to the size of the automaton $G \parallel H$ with $N$ states, an alphabet of size $|\Sigma|$, and, because it is a DFA, an upper bound of $N \times |\Sigma|$ transitions.

Line 1 is a memory operation where we denote the input $G \parallel H$ by $M$. This happens in constant time. Line 2 is also a memory operation, where the set of states to remove from at most $N$ estimates are initialized to $\emptyset$. The complexity class for this operation is $O(N)$. Line 3 is another memory operation where we denote the input $\Delta$ by $\Delta_0$. This happens in constant time.

The first loop, starting at Line 4, will in the worst case examine every transition in $G \parallel H$ once. Practically speaking, transitions could be stored in a sorted list with all uncontrolled transitions at the start of the list. For each transition, at most four constant time operations are performed: verifying if the transition is uncontrolled; adding the origin state to $\Delta_Q$; and removing the destination state from both $\Delta_Q$ and $M$. The complexity class for this loop is $O(N \times |\Sigma|)$.

The second loop, starting at Line 11, will examine every state in $G \parallel H$ once. For each state, at most two constant time operations are performed: checking if the state is inaccessible; and adding the first element of the state to $A_r(A)$. The complexity class for this loop is $O(N)$.

The third loop, starting at Line 14 will examine every state in $G \parallel H$ once. For each state, exactly one constant time operation is performed: relabelling the state’s estimate by removing the previously stored set $A_r(A)$ from the current estimate. The complexity class for this loop is $O(N)$.

Line 16 requires returning the accessible portion of $M$. Practically, the inaccessible components of $M$ can be removed during the second loop, and so this is a constant time operation.

Taken all together, the complexity class for $\text{REFINE}$ is $O((N \times 3) + (N \times |\Sigma|))$, which simplifies to $O(N \times |\Sigma|)$. If we consider that the number of states in an automaton is generally larger than the number of events, this further simplifies to $O(N)$.

### 4.5 Discussion

The subobserver relationship is a more general form of the subautomaton relationship. We showed that the subobserver relationship captures how the observer automaton changes to reflect evolutions in the plant. Specifically, the subobserver relationship allows the parallel composition of a plant and an observer automaton for that plant, $G \parallel H$, to be refined without recomputing the observer. Instead, our method removes transitions in the parallel composition if and only if they must be disabled by supervisory control and updates the
observer’s estimates. The REFINE algorithm implements this procedure and its asymptotic time complexity is linear in the number of states of $G \parallel H$.

Although the REFINE algorithm was developed within the framework of the opacity control problem, it more generally produces the joint behaviour of plant and observer as the plant evolves over time. In the remainder of this paper we consider the opacity control problem exclusively, but we note that this approach is applicable to any DES problem where evolving discrete-event processes are observed including online control, dynamic discrete-event systems, and decentralized control.

5 Synthesizing opacity-enforcing supervisors

Our opacity control problem formulation is similar to that from other works (see Problem 1). We assume that all controllable events are observable, $\Sigma_c \subseteq \Sigma_s$, and that the adversary observes a subset of the events that the supervisor observes, $\Sigma_a \subseteq \Sigma_s$. These assumptions allow us to use Dubreil et al.’s reduction of the general opacity control problem to the opacity control problem with full observation, i.e. $\Sigma_s = \Sigma$ (Dubreil et al. 2010, Proposition 5). We assume a supervisor-aware adversary, which implies that the adversary’s view will evolve with $G' = S/G$. For simplicity we enforce current-state opacity.

Recalling approaches to the opacity control problem, the naïve, computationally inefficient solution is to apply supervisory control and calculate observers iteratively (Fig. 3a) (Dubreil et al. 2010). Condensed state estimates (Fig. 3b) require alternately constructing partial maps and automata; since the only states in a condensed state estimate are those that were reached by the last event observable to the adversary, “loosing traces” are calculated at each iteration to determine which states will reveal a secret (Dubreil et al. 2010). The augmented I-observer (Fig. 3c) solves a more general form of the problem but requires calculating the supremal G-opaque sublanguage, synthesizing a supervisor and refining both the plant and augmented I-observer at each step. Tong et al. avoid this complication by restricting themselves to the case of an adversary who is unaware of the supervisor (Tong et al. 2018).

Our method uses the subobserver property to calculate $G' \parallel H'$ directly from $G \parallel H$, refining the parallel composition of plant and adversary view (Fig. 3d). Advantages of this method include: not producing any intermediary structures or languages; capturing the adversary’s beliefs better than condensed state estimates, thus not requiring “loosing
traces” to be calculated; and avoiding unnecessary calculations compared to the augmented I-observer with a supervisor-aware adversary.

5.1 The SYNTHESIZE algorithm

As in Section 4, we use the parallel composition of $G$ and $H := T_{\Sigma}(G)$ to track the plant’s evolving structure along with the adversary’s beliefs about the plant. Importantly, we mark states in $H$ to capture the scenarios in which no non-secret state can be confused by the adversary with a true secret state. With a traditional observer automaton, a state is marked in $H$ when one of the states that it contains is itself marked in $G$; in our work, by contrast, a state in $H$ is marked when all the states it contains are themselves marked in $G$, as described in Section 2.1.1. Thus, in our work, if $H$ – the observer – reaches a marked state, then a string has occurred in $G$ – the plant – that allows the adversary to be sure that a secret state has been reached. This marking carries through to $G \parallel H$, where marked states must be made inaccessible to produce an opacity-enforcing supervisor.

The SYNTHESIZE algorithm (Algorithm 2) enforces current-state opacity by applying supervisory control as long as the controlled plant composed with the adversary’s estimations of the plant contains marked states (Lines 3-8), the adversary’s view of the plant is updated (Line 5), and the new composition of plant behaviour and adversarial belief is checked for current-state opacity (Lines 6-8).

5.2 Worked example

We will demonstrate how the SYNTHESIZE algorithm works for a concrete example, beginning with the plant and the initial parallel composition (Fig. 4). This plant has two marked states, 6 and 10, an event that the supervisor cannot control, $\beta$, and events that the adversary cannot observe, $\alpha$ and $\gamma$.

The initial parallel composition has a single marked state, $^{10}_{10}$, which is added to $\Delta$ and passed to REFINE. In order to prevent the state $^{10}_{10}$ from being reached, we must also mark the state $^{6}_{9}$ in $M$ since it leads uncontrollably to our marked state. Remov-
ing these two states from \( M \) implies that when the supervisor is in state \( \frac{8}{3,4,8} \), it will disable the event \( \lambda \) via supervisory control since the transition \( \left( \frac{8}{3,4,8}, \lambda, \frac{9}{6,9} \right) \) is no longer defined. Removing these states from \( M \), however, means that our supervisor will not allow the plant \( G \) to reach state 9, which will affect the adversary’s estimates. Specifically here, when the plant is in state 6, the adversary will no longer confuse the string in \( G \) that led to state 6 with a string in \( G \) that leads to state 9 since the latter is now inaccessible in the plant. This leads to the new parallel composition in Fig. 5a, which also has one marked state, \( \frac{6}{6} \).

Removing the state \( \frac{6}{6} \) from \( M \) produces the parallel composition in Fig. 5b. This automaton has no marked states, indicating that it can be used as a supervisor to enforce current-state opacity in the plant \( G \).

### 5.3 The case of state splitting

As remarked in Section 3, we are not guaranteed that applying supervisory control to \( G \) will result in a subautomaton of \( G \). For example, the observer automaton in Fig. 6 illustrates how although the secret states 8 and 9 are repeatedly visited, the adversary is only able to determine that the plant is in a secret state once it sees a fourth \( a \). This means that a supervisor can allow state 8 to be visited up to three times and must therefore enact different control patterns for state 8 depending on how many times the plant has visited state 8.

State splitting is required for opacity enforcement when a state can be visited in the prefixes of strings leading to secret states but only a finite number of times, requiring the supervisor to know which string led to the state for a particular visit (Cassandras and Laforetune 2008, p. 141). Note that if a state can be visited an infinite number of times then the supervisor can always enact the same control pattern. Happily, this state splitting occurs implicitly in Line 2 of the SYNTHESIZE algorithm when the first parallel composition is

![Fig. 4](image)

(a) The original plant, \( G \)

(b) Initial parallel composition, \( G \parallel H \)

**Fig. 4** The initial plant and parallel composition for the worked example
constructed (Fig. 7). Because a state that requires splitting can only be safely visited a finite number of times, it is possible to transform any DFA, $G$, into a language-equivalent DFA, $\hat{G}$, by replicating the original state for each time that it can be safely visited. In fact, taking only the first element of the states in $G$'s state transition diagram produces $\hat{G}$'s state transition diagram. Following this initial transformation, we are then guaranteed that $G/\text{uni}$ will be a strict subautomaton of $\hat{G}$ and the SYNTHESIZE algorithm can proceed from there. We note that this state splitting is not an artifact of our proposed method: any method for producing a minimally restrictive opacity-enforcing supervisor from the original plant $G$ must be able to reason about the number of times that such states have been reached. The structure $G/\text{uni}$ embeds this information by splitting states the necessary number of times; it is therefore the

Fig. 6 States 8 and 9 in $G$ are secret and only $a$ is observable to the adversary. The observer automaton shows that after observing a fourth $a$, an adversary will know the plant is in a secret state. Note that only \{8,9\} is marked in $H$ because this is the only state whose constituent states are all marked in $G$.

Fig. 7 Continuing from Fig. 6, the first parallel composition inherently constructs the needed, language-equivalent automaton to represent the plant.
smallest memoryless structure that can be used to solve the opacity control problem in the context of a supervisor-aware adversary.

5.4 Proof of correctness for the SYNTHESIZE algorithm

First, the SYNTHESIZE algorithm is guaranteed to terminate. As with the REFINE algorithm, termination is guaranteed because the automaton $G \parallel H$ has a finite number of states and the SYNTHESIZE algorithm makes states inaccessible until $M$ enforces opacity or is the null automaton. Specifically, if $M$ has any marked states then the algorithm removes states – never adding states – to make the marked states inaccessible. Since $M$ has a finite number of states, and the algorithm terminates if $M$ is the null automaton, the algorithm is guaranteed to terminate.

Second, we prove the correctness of SYNTHESIZE through Theorem 3.

**Theorem 3** Given a DFA $G = (Q, \Sigma, \delta_G, q_0, Q_m)$, set of secret states $Q_m$, set of events visible to the adversary $\Sigma_a \subseteq \Sigma$, and set of events controllable by the supervisor $\Sigma_c \subseteq \Sigma$. Then the supervisor $S$ produced by the SYNTHESIZE algorithm is a correct and maximally-permissive opacity-enforcing supervisor for $G$.

**Proof** We begin by proving by contradiction that the supervisor $S$ correctly enforces opacity for $G$. We assume that $S$ does not enforce opacity for $G$. In this case, at least one of the states in $S = M$ must be marked, reflecting that all of the states in the adversary’s estimate of the plant are secret states in $G$. If this was the case, however, then the while loop at Lines 3 to 8 would have executed again and removed the last controllable transition along each path to this state, rendering it inaccessible. Therefore, when the SYNTHESIZE algorithm terminates, $S = M$ has no marked states and it therefore correctly enforces opacity for $G$.

Next, we prove by contradiction that the supervisor $S$ is maximally-permissive while enforcing opacity for $G$. Assume that there exists a supervisor $S'$ that correctly enforces opacity for $G$ while being more permissive than supervisor $S$. This implies that there is a transition in $G \parallel H, \delta^*$, that $S'$ enables and that $S$ disables. But the only transitions that $S$ disables in $G \parallel H$ are those that lead directly to a marked state or to a state which leads uncontrollably to a marked state, so $\delta^*$ must either lead directly or uncontrollably to a marked state. Therefore, $S'$ does not correctly enforce opacity for $G$.

5.5 Time complexity analysis of the SYNTHESIZE algorithm

Although the SYNTHESIZE algorithm presents a straightforward way of seeing how we can iteratively generate a list of transitions, $\Delta$, and then refine the parallel composition of plant and adversary’s view, its asymptotic time complexity is in a higher complexity class than other methods in the literature. We therefore present Algorithm 3, which embeds the REFINE algorithm into the SYNTHESIZE algorithm in a computationally efficient manner. We tie the asymptotic behaviour of Algorithm 3 to the input of interest, the automaton $G$ with $|Q|$ states and an alphabet of size $|\Sigma|$.
Initializing the algorithm, Lines 1-3, requires constructing the adversary’s view, \( H = T_{\Sigma_c}(G) \), and the parallel composition of \( G \parallel H \). The time complexity of this process is established to be \( O(|Q| \times 2^{|\Sigma|}) \). The production of the hash table requires an operation for each \(|Q|\) state in \( M \) as well and in practice it can be constructed alongside the parallel composition.

The outer loop iterates through the marked states in \( M \), Lines 4 to 11. This loop will run a variable number of times, but we will be able to assess the loop’s complexity by establishing upper bounds for each of the lines inside this loop. First, the loop responsible for inspecting transitions in \( M \), Lines 5-7, visits each transition in \( M \) at most once and at most one constant time operation is performed. This is guaranteed because each visited transition is visited because its destination state is marked. Because that destination state will be removed at Line 8 of the current iteration, the transition in question will not be visited again. Lines 5-7 therefore have an asymptotic time complexity over the algorithm’s whole run time of \( O(|Q| \times 2^{|\Sigma|} \times |\Sigma|) \). Second, the lines responsible for processing marked states in \( M \), Lines 8-11 consists of up to four operations. Removing the state \( m \) from \( M \) has a constant time complexity; because each state in \( M \) can be removed at most once this line has an asymptotic time complexity over the algorithm’s whole run time of \( O(|Q| \times 2^{|\Sigma|}) \). Updating the hash table key similarly has a constant time complexity and can only occur once per state in \( M \). This again leads to an asymptotic time complexity over the algorithm’s whole run time of \( O(|Q| \times 2^{|\Sigma|}) \). Finally, checking if the new estimate consists of solely secret states (Line 10) takes at most \(|Q_m|\) checks and then marking states in the hash table slot (Line 11) takes constant time by state. Since every state can only be marked once over the algorithm’s whole run time, this again leads to an asymptotic time complexity of \( O(|Q| \times 2^{|\Sigma|}) \).

Taken together, the asymptotic time complexity of Algorithm 3 is \( O(|Q| \times 2^{|\Sigma|} \times |\Sigma|) \). Since in many problems the size of the alphabet does not grow and is much smaller than the size of the state space, we can consider \(|\Sigma|\) as a constant and reduce this time complexity to \( O(|Q| \times 2^{|\Sigma|}) \). Our result is in keeping with the exponential time required for computing an observer automaton and we note that there is no algorithm for verifying current-state opacity for a system \( G \) whose time complexity is polynomial in the number of states in \( G \) (Wu and Lafortune 2013).
5.6 Discussion

We have demonstrated how our method for refining parallel compositions of plant and observer can be used to synthesize an opacity-enforcing supervisor. The SYNTHESIZE algorithm implements this procedure in a direct manner and Algorithm 3 trades a small increase in memory usage to reduce the asymptotic time complexity to $O(|Q| \times 2^{(|Q|)})$ where $|Q|$ is the number of states in the original plant.

We showed two examples to demonstrate how our algorithm works. The first example highlighted the interplay between enacting supervisory control and enforcing opacity when faced with a supervisor-aware adversary: multiple refinements of the automaton $G \parallel H$ may be required before an output is produced that can be used as an opacity-enforcing supervisor. The second example showed how our approach inherently addresses splits states when a secret state can be safely visited only a finite number of times without revealing the system secret. Recalling the two methods in the literature that are most similar to ours, we note that Algorithm 3 offers improvements on both approaches. Although each method has the same asymptotic time complexity, Tong et al. (2018), our method produces only a single structure at each step: the parallel composition of plant and adversary estimate.

The method of condensed state estimates, by contrast, produces both a partial map and an automaton at every step (Dubreil et al. 2010). Additionally this method requires “loosing paths” to be assessed in order to verify the opacity property because condensed state estimates account only for states that the plant might have reached if the last observed event is the last event that occurred in the plant. Our method does away with the need to reason about these “loosing paths,” which produces two advantages. First, opacity is immediately verified by the lack of marked states in the output which permits an opacity-enforcing supervisor to be easily verified by an independent party. Second, the output’s state labels are semantically meaningful: if a trace occurs in the system then this trace leads to a state in the parallel composition whose label is the current state of the system and the states that the observer believes the system could be in. This information can be used by an engineer or a computer program to make decisions about the system.

The augmented I-observer method, solves a more general problem by allowing $\Sigma_s$ and $\Sigma_a$ to be incomparable (Tong et al. 2018). Balanced against this, it focuses on the case of a single iteration to enforce opacity against a non-supervisor-aware adversary. Because this method calculates the supremal G-opaque sublanguage of the plant’s language and a supervisor to enforce this language as a specification, Tong et al. (2018), this results in unnecessary calculations when a supervisor-aware adversary makes multiple iterations necessary. Our method makes the realistic assumption that $\Sigma_a \subseteq \Sigma_s$ and addresses the case of a supervisor-aware adversary with only a single structure computed at each step.

6 Conclusion

The opacity control problem has been of interest in the DES literature for many years and has been addressed a number of times. It is a complex problem whose straightforward solution is computationally expensive. Compared with our method, the solutions
to the opacity control problem in the literature variously require additional computations to deal with the problem in its general form or use structures that do not intuitively align with the opacity control problem.

This paper introduced and described the subobserver relationship. This relationship is analogous to the subautomaton relationship and links the observers of a plant whose structure evolves. We demonstrated the usefulness of the subobserver relationship by using it as the basis of an algorithm to solve the opacity control problem and we believe that this relationship has broader application in DES research including online control, dynamic discrete-event systems, and decentralized control.

Time complexity analysis shows that our algorithms are computationally efficient, with Algorithm 3 matching the asymptotic time complexity of previous methods in the literature. We also demonstrated that our algorithms intuitively solve the opacity control problem under reasonable assumptions. Future research applying these ideas may include relaxing the requirement that $\Sigma_c \subseteq \Sigma_o$ for the opacity control problem and applying the subobserver relationship to the iterative refinements necessary when a single supervisor enacts online control or several agents enact decentralized control.

Appendix : A: Omitted Proofs

**Proof (Proof of Lemma 2)** We begin by noting that $L(G \parallel H) = L(G)$ and that because $\Sigma_o \subseteq \Sigma$, the alphabet for $G \parallel H$ is $\Sigma$ as well.

$\Rightarrow$. If a state $q$ is accessible in $G \parallel H$, then there exists a string $s \in L(G \parallel H)$ such that $\delta_{G \parallel H}(q, s) = q$. By Definition 8, this implies that that $\delta_G(q_0, s) = q$ and that $\delta_H(A_0, P_{\Sigma}(s)) = A$. Since $\delta_H(A_0, P_{\Sigma}(s)) = A$, we know that every string that shares a projection with $s$ leads to a state in $A$ and that every state in $A$ is reached by a string that shares a projection with $s$. So we can conclude that

$$\exists s \in L(G) \delta_G(q_0, s) = q \quad | \quad \exists q' \in Q \ | \ \exists s' \in L(G) \ | \ P_{\Sigma_o}(s) = P_{\Sigma}(s') \delta_G(q_0, s') = q' \ = A.$$ \[
\]

$\Leftarrow$. We are given that

$$\exists s \in L(G) \delta_G(q_0, s) = q \quad | \quad \exists q' \in Q \ | \ \exists s' \in L(G) \ | \ P_{\Sigma_o}(s) = P_{\Sigma}(s') \delta_G(q_0, s') = q' \ = A.$$ \[
\]

We therefore know that $\delta_G(q_0, s) = q$ and that $\delta_H(A_0, P_{\Sigma}(s)) = A$. By Definition 8, this implies that $\delta_{G \parallel H}(q_0, s) = q$ and therefore $q$ is accessible in $G \parallel H$.

**Proof (Proof of Lemma 3)** According to the definition, $G'$ is a subautomaton of $G$, denoted by $G' \subseteq G$, if $\delta_G(q_0', s) = \delta_G(q_0, s) \ \forall \ s \in L(G')$.

Denote the first elements of states remaining in $G \parallel H$ as $G'$. Then we know that

$$\forall s \in L(G') \delta_{G'}(q_0, s) = q \implies \delta_G(q_0, s) = q$$

since any first element of a state that survives in $G \parallel H$ was originally in $G$. This means that any string that occurs in $G'$ leads to the same state that it did in $G$ and therefore that $\delta_G(q_0', s) = \delta_G(q_0, s) \ \forall \ s \in L(G')$.

We conclude that $G'$ is a subautomaton of $G$. 

© Springer
Acknowledgements The authors acknowledge that Queen’s University is situated on traditional Anishinaabe and Haudenosaunee Territory. The authors would like to thank Richard Ean and Bryony Schonewille of the Queen’s Discrete-Event Systems Lab as well as the reviewers for their thoughtful and constructive comments. This research was supported by the Natural Sciences and Engineering Research Council of Canada as well as the Queen’s University Faculty of Engineering and Applied Science. RHM held a Walter C. Sumner Memorial Fellowship during the conduct of this work. All figures with automata were produced using the Integrated Discrete-Event Systems Tool, Rudie (2006), which is freely available online at https://github.com/krudie/IDES.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

Badouel E, Bednarczyk M, Borzyszkowski A, Caillaud B, Darondeau P (2007) Concurrent secrets. Discrete Event Dyn Syst 17(4):425–446. https://doi.org/10.1007/s10626-007-0020-5

Ben-Kalefa M, Lin F (2011) Supervisory control for opacity of discrete event systems. In: 49Th annual allerton conference on communication, control, and computing. IEEE, pp 1113–1119. https://doi.org/10.1109/Allerton.2011.6120292

Bryans JW, Koutny M, Mazaré L, Ryan PY (2008) Opacity generalised to transition systems. Int J Inf Secur 7(6):421–435. https://doi.org/10.1007/s10207-008-0058-x

Cassandras CG, Lafortune S (2008) Introduction to discrete event systems, 2nd edn. Springer, Boston. https://doi.org/10.1007/978-0-387-68612-7

Cho H, Marcus SI (1989) Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. Math Syst Theory 22(1):177–211. https://doi.org/10.1007/BF02088297

Chung SL, Lafortune S, Lin F (1992) Limited lookahead policies in supervisory control of discrete event systems. IEEE Trans Autom Control 37(12):1921–1935. https://doi.org/10.1109/9.182478

Dubreil J, Darondeau P, Marchand H (2010) Supervisory control for opacity. IEEE Trans Autom Control 55(5):1089–1100. https://doi.org/10.1109/TAC.2010.2042008

Grigorov L, Rudie K (2006) Near-Optimal Online control of dynamic Discrete-Event systems. Discrete Event Dyn Syst 16(4):419–449. https://doi.org/10.1007/s10626-006-0020-x

Hopcroft JE, Ullman JD (1979) Introduction to automata theory languages and computation

Jacob R, Lesage JJ, Faure JM (2016) Overview of discrete event systems opacity: models, validation, and quantification. Annu Rev Control 41:135–146. https://doi.org/10.1016/j.arcontrol.2016.04.015

Jirásková G, Masopust T (2012) On properties and state complexity of deterministic state-partition automata. Lect Notes Comput Sci 7604:164–178. https://doi.org/10.1007/978-3-642-33475-7_12

Lin F (2011) Opacity of discrete event systems and its applications. Automatica 47(3):496–503. https://doi.org/10.1016/j.automatica.2011.01.002

Lin F, Wonham WM (1988) Decentralized supervisory control of discrete-event systems. Inf Sci 44(3):199–224. https://doi.org/10.1016/0020-0255(88)90002-3

Lin F, Wonham WM (1988) On observability of discrete-event systems. Inf Sci 44(3):173–198. https://doi.org/10.1016/0020-0255(88)90001-1

Mazaré L (2004) Using unification for opacity properties. In: Proceedings of the workshop on issues in the theory of security (WITS’04), vol 7, pp 165–176

Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230. https://doi.org/10.1137/0325013

Rosen KH (2019) Discrete mathematics and its applications, 8th edn. New York, McGraw-Hill Education

Saboori A, Hadjicostis CN (2008) Verification of initial-state opacity in security applications of DES. 9th International Workshop on Discrete Event Systems (WODES 08:328–333. https://doi.org/10.1109/WODES.2008.4605967

Saboori A, Hadjicostis CN (2011) Verification of K-Step opacity and analysis of its complexity. IEEE Trans Autom Sci Eng 8(3):549–559. https://doi.org/10.1109/TASE.2011.2106775
Saboori A, Hadjicostis CN (2012) Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Trans Autom Control 57(5):1155–1165.  https://doi.org/10.1109/TAC.2011.2170453

Takai S, Oka Y (2008) A formula for the supremal controllable and opaque sublanguage arising in supervisory control. SICE Journal of Control Measurement, and System Integration 1(4):307–311.  https://doi.org/10.9746/jcmsi.1.307

Tong Y, Li Z, Seatzu C, Giua A (2018) Current-state opacity enforcement in discrete event systems under incomparable observations. Discrete Event Dyn Syst: Theory Appl 28(2):161–182.  https://doi.org/10.1007/s10626-017-0264-7

Wonham WM, Cai K (2019) Supervisory control of Discrete-Event systems. Communications and control engineering. Springer International Publishing Cham.  https://doi.org/10.9746/jcmsi.1.307

Wu YC, Lafortune S (2013) Comparative analysis of related notions of opacity in centralized and coordinated architectures. Discrete Event Dyn Syst: Theory Appl 23(3):307–339.  https://doi.org/10.1007/s10626-012-0145-z

Yin X, Lafortune S (2015) A new approach for synthesizing opacity-enforcing supervisors for partially-observed discrete-event systems. In: 2015 American control conference (ACC). IEEE, Chicago, pp 377–383.  https://doi.org/10.1109/ACC.2015.7170765

Yin X, Lafortune S (2016) A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. IEEE Trans Autom Control 61(8):2140–2154.  https://doi.org/10.1109/TAC.2015.2484359

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Richard Hugh Moulton received the Master of Computer Science degree from the University of Ottawa in 2018. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Queen’s University. His research interests include artificial intelligence, the control of discrete-event systems, and their applications in computational neuroscience. His research is supported by the Natural Sciences and Engineering Research Council of Canada, the Faculty of Engineering and Applied Science at Queen’s University, and The Walter C. Sumner Foundation.

Behnam Behinaein Hamgini received his Ph.D. from Queen’s University in Electrical and Computer Engineering in 2016. He is currently a Postdoctoral Fellow in the Department of Electrical and Computer Engineering at Queen’s University. His research interests include the application of machine learning and deep learning in time-series data analysis of wearables, opacity in discrete-event systems, systems biology, and complex systems.
Zahra Abedi Khouzani received the Master of Applied Science in Electrical and Computer Engineering from Queen’s University in 2019. Her main research interests are opacity and control of discrete-event systems, algorithm design, and robotics.

Rômulo Meira-Góes is a joint Research Fellow at the University of Michigan and the Carnegie Mellon University. He received the B.Eng degree from Universidade Tecnológica Federal do Paraná-Curitiba in 2015, the M.Eng degree from the University of Michigan in 2017, all in electrical engineering, and the Ph.D. degree in Electrical and Computer Engineering at the University of Michigan in 2020. His research interests include supervisory control of discrete event systems and formal methods, especially, their application in cybersecurity of cyber-physical systems.

Fei Wang received his Ph.D. from Xi’an Jiaotong University (China) in 2009. Since 2009, he has been at Huaqiao University (China) where he is an Associate Professor in the College of Information Science and Engineering. In 2018-19, he was a Visiting Scholar at Queen’s University (Canada). His research focuses on supervisory control of discrete event systems.
Karen Rudie received her Ph.D. in 1992 from the University of Toronto, in the Systems Control Group. In 1992-93 she was a postdoctoral researcher at the Institute for Mathematics and its Applications, Minnesota. Since 1993 she has been at Queen’s University (Canada) where she is a Professor of Electrical and Computer Engineering, with a cross-appointment in the School of Computing. From 2004-06 she was an IEEE Control Systems Society Distinguished Lecturer. She has served on the editorial board of the Journal of Discrete Event Dynamic Systems (since 2000), where she is currently a Department Editor, and has served as an Associate Editor for IEEE Transactions on Control Systems Technology (2015-2020), IEEE Transactions on Automatic Control (1996-99), and IEEE Control Systems Magazine (2003). She is a Fellow of the IEEE. Her research focuses on the control of discrete-event systems.