Verification of Current State Opacity using Switching Output Automata

Tianyu Liu, Carla Seatzu and Alessandro Giua

July 2023

Abstract

We present a new discrete event model called *switching output automaton* where with each state is associated a set of discrete output values. The evolution of such a system, as state and output change, produces as observation a piecewise constant signal. The goal of this paper is that of designing a suitable observer for estimating the current discrete state of this model as a function of the observed output signal. In addition, we show how the observer can be used to verify if the system is current-state opaque.

Published as:
T. Liu, C. Seatzu and A. Giua, "Verification of Current State Opacity using Switching Output Automata," *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 2665-2670, 2023.
DOI:10.1109/CoDIT58514.2023.10284385

This work was partially supported by the China Scholarship Council.

Tianyu Liu, Carla Seatzu and Alessandro Giua are with DIEE, University of Cagliari, Italy. Emails: t.liu@studenti.unica.it, carla.seatzu@unica.it, giua@unica.it.

1 Introduction

In the research on security and privacy of Cyber Physical Systems, opacity is a property that has received significant attention. The purpose of opacity is to ensure that external intruders cannot detect the system's secret information (current or initial state, generated words). Several discrete event models have been considered in the literature, such as Moore or Mealy machines, automata, Petri nets, etc. [1]. In this paper, we study current state opacity considering a new model, named *switching output automaton*.

In much of the DES literature on partially observed systems, the measurable outputs, i.e., the observations generated by the plant, are associated to event occurrence. This is the typical setting of Mealy machines. A Moore machine is a finite state machine whose current output values are determined only by its current state [2, 3]. This setting is common in stochastic models such as hidden Markov models [4] and has also been used for state reconstruction in logical models [5].

Unlike previous works, we consider a different model where outputs are piecewise constant boolean signals, as opposed to impulsive signals (namely observations produced each time a state transition occurs). This seems more natural when studying physical systems where the output is a continuous-time signal: as an example, the power consumption of an electrical user, which depends on the devices currently active, is a signal of this type. Furthermore, the output produced at a given state may not be unique due to several reasons: as an example, the state of an electrical user may correspond to the fact that a device is active but its observed power consumption may depend on different operating modes.

There exists a rich literature on current state opacity in DESs [6, 7, 8, 9, 10, 11]. Saboori and Hadjicostis [7] verify current state opacity by converting a nondeterministic finite automaton (NFA) into an equivalent deterministic finite automaton (DFA) and the complexity is $O\left(2^n\right)$ with n being the number of states in the NFA. Tong et al. [11] use notions of basis markings and explanations to construct the basis reachability graph and further verify the opacity of labeled Petri nets. Zhang et al. [12] propose a multi-linear algebraic expression to characterize the intrusion estimator, and verify current state opacity with event-state observation and state observation by some matrix manipulations. They first build the structure matrix of state machine and use Boolean Semi-tensor product of matrices algorithm for opacity analysis on this basis. The complexity of the whole algorithm is based on the dimensionality of the matrix. The model in [12] is a Moore finite state machine, then each state of the machine only has one output. The observation of the machine is a time-independent sequence of state outputs, where two adjacent symbols in the sequence can take the same value.

In this paper, we provide the following contributions.

First, we define a new model that we call *switching output automaton* (SOA) and describe its evolution and the produced observations. SOA are a quite natural model to describe man-made systems where the output can take discrete values, or even piecewise-continuous values which can be quantized.

An SOA can be seen as a Moore automaton with a multivalued output function that associates multiple output values with each state. There are, however, two main differences with respect to Moore automata.

- The evolution of an SOA can be characterized in terms of both state and output changes.
- A Moore automaton produces as observation a logical sequence of output values which may contain repeated values, say, aab. An SOA produces as observation a piecewise-constant signal, say, a for $t \in [0, t_1]$ and b for $t \in [t_1, t_f]$.

Next, we construct an observer which provides the set of states consistent with a given output observation for state. Then, we extend the notion of current state opacity defined in [7] for automata to the proposed model. Finally, we show how to verify it using the proposed observer.

2 Switching Output Automaton

Definition 1. *A switching output automaton is defined as* $G = (X, Y, B, h, x_0, y_0)$*, where*

- X *is a finite set of states;*
- Y *is an* output alphabet*;*
- $B \subseteq X \times X$ *is a set of arcs (or edges) ;*
- $h : X \to 2^Y$ *is the* output function*;*
- $x_0 \in X$ *is the initial state;*
- $y_0 \in h(x_0)$ *is the initial output symbol.*

We denote by $h(x) \subseteq Y$ the set of possible output symbols which may be produced when the current state is x. An arc $b = (x, x') \in B$ is considered to be directed from state x to state x'; in such a case x' is said to be a *direct successor* of state x and state x is said to be a direct predecessor of state x'. We denote by $\sigma(x) = \{x' \in X | \exists b = (x, x') \in B\}$ the set of direct successors of x.

An *alphabet* Y is a finite set of symbols and Y^* denotes the set of all (finite length) strings of symbols in Y, while $Y^+ = YY^*$.

The evolution of the state can be described by means of a state run

$$\xrightarrow{t_0} x^{(0)} \xrightarrow{t_1} x^{(1)} \xrightarrow{t_2} \cdots \xrightarrow{t_k} x^{(k)}$$

where for all $i = 0, \ldots, k$ it holds that $x^{(i)} \in X$ and for all $i = 0, \ldots, k-1$ it holds that $(x^{(i)}, x^{(i+1)}) \in B$. Such a run has length $k \geq 0$. It describes the evolution process of an automaton, which initially is in state $x^{(0)}$ at time t_0, and then at time t_i transitions from state $x^{(i-1)}$ to state $x^{(i)}$. While the automaton is in a given state its output may change. This is described by an output run associated to a state $x^{(i)}$.

$$\xrightarrow[\tau_0^{(i)}]{t_i} y_0^{(i)} \xrightarrow[\tau_1^{(i)}]{} y_1^{(i)} \xrightarrow[\tau_2^{(i)}]{} \cdots \xrightarrow[\tau_{h_i}^{(i)}]{} y_{h_i}^{(i)}$$

where $\forall j = 0, \ldots, h_i$, $y_j^{(i)} \in h(x^{(i)})$ and for all $j = 1, \ldots, h_i$, $y_{(j-1)}^{(i)} \neq y_{(j)}^{(i)}$. Such an output run has length $h_i \geq 0$. When state $x^{(i)}$ is reached at time $\tau_0^{(i)} = t_i$ the output takes value $y_0^{(i)}$ and changes from $y_{j-1}^{(i)}$ to $y_j^{(i)}$ at time $\tau_j^{(i)}$.

A state-output run can thus be represented as follow:

$$\begin{aligned}
&\xrightarrow[\tau_0^{(0)}=0]{t_0=0} \begin{pmatrix} x^{(0)} \\ y_0^{(0)} \end{pmatrix} \xrightarrow[\tau_1^{(0)}]{} \begin{pmatrix} x^{(0)} \\ y_1^{(0)} \end{pmatrix} \xrightarrow[\tau_2^{(0)}]{} \cdots \xrightarrow[\tau_{h_i}^{(0)}]{} \begin{pmatrix} x^{(0)} \\ y_{h_i}^{(0)} \end{pmatrix} \xrightarrow[\tau_0^{(1)}]{t_1} \\
&\begin{pmatrix} x^{(1)} \\ y_0^{(1)} \end{pmatrix} \xrightarrow[\tau_1^{(1)}]{} \cdots \xrightarrow[\tau_0^{(i)}]{t_i} \begin{pmatrix} x^{(i)} \\ y_0^{(i)} \end{pmatrix} \xrightarrow[\tau_1^{(i)}]{} \cdots \xrightarrow[\tau_{h_i}^{(i)}]{} \begin{pmatrix} x^{(i)} \\ y_{h_i}^{(i)} \end{pmatrix} \\
&\xrightarrow[\tau_0^{(i+1)}]{t_{i+1}} \begin{pmatrix} x^{(i+1)} \\ y_0^{(i+1)} \end{pmatrix} \xrightarrow[\tau_1^{(i+1)}]{} \cdots \xrightarrow[\tau_{h_i}^{(k)}]{} \begin{pmatrix} x^{(k)} \\ y_{h_i}^{(k)} \end{pmatrix}
\end{aligned}$$

where each state of the run $q = (x, y) \in X \times Y$ is a *global state* of the system: it consists of a pair whose first element is a discrete state $x \in X$ and whose second element is an output value $y \in h(x) \subseteq Y$.

An SOA has multiple state-output runs, which consists of three types of transitions.

Type 1 refers to a state change with no output change. It can be explained as follows. During the time interval $[\tau_{h_i}^{(i)}, t_{i+1})$, the system is in state $x^{(i)}$ and produces output $y_{h_i}^{(i)}$. At time instant t_{i+1}, the system transitions to state $x^{(i+1)}$, while maintaining the same output value, i.e., $y_{h_i}^{(i)} = y_0^{(i+1)}$.

Type 2 indicates a simultaneous state and output change. During the time interval $[\tau_{h_i}^{(i)}, t_{i+1})$, the system is in state $x^{(i)}$ and produces output $y_{h_i}^{(i)}$. At moment t_{i+1}, the system transitions to state $x^{(i+1)}$ and generates a new output, which means that $y_{h_i}^{(i)} \neq y_0^{(i+1)}$.

Type 3 indicates an output change with no state change. In a run as described above, the system undergoes h_i changes in the output while the current state remains $x^{(i)}$ and the output changes are represented as $y_0^{(i)} \to y_1^{(i)} \to \cdots \to y_{h_i}^{(i)}$.

We assume that the time distance between the occurrence of any two such transitions must be greater than or equal to the dwell time δ, i.e., $\forall i = 0, \ldots, k$, $\forall j = 0, \ldots, h_i - 1$: $\tau_{j+1}^{(i)} - \tau_j^{(i)} \geq \delta$ and $\forall i = 0, \ldots, k-1$: $\tau_0^{(i+1)} - \tau_{h_i}^{(i)} \geq \delta$. The dwell time has been introduced to avoid Zeno phenomena, namely to avoid an infinite number of switches in the ouput and the discrete state in a time interval of finite length.

The observation of the system is a mapping $T \to Y$, where $T = \mathbb{R}_{\geq 0}$ represents the time that can be divided into a countably finite number of time intervals as detailed above, in accordance with the state-output run. We define $y(t)$ as the output symbol produced by the system at time t. Function $y(t)$ is piecewise continuous and each value it takes has a duration at least equal to δ. Note that, by definition, the output of two adjacent time intervals is different. We use (y, τ) to denote that the output takes value y for a time interval of duration τ where $y \in Y$ and $\tau \in \mathbb{R}_{\geq 0}$. The behavior of G is defined as $L(G) = \{\omega \in (Y, \mathbb{R}_{\geq \delta})^+ | \forall i = 1, 2, \ldots, n : \omega = (y_{[0,\tau_1)}, \tau_1)(y_{[\tau_1,\tau_2)}, \tau_2 - \tau_1) \cdots (y_{[\tau_{i-1},\tau_i)}, \tau_i - \tau_{i-1})$ and $y_{[\tau_{k-1},\tau_k)} \neq y_{[\tau_k,\tau_{k+1})}, k = 1, \ldots, i-1$ and $\tau_i - \tau_{i-1} \geq \delta\}$.

The state of the system is also a piecewise continuous function $x : T \to X$. The state $x(t)$ cannot be directly measured but an estimation of it can be computed based on the knowledge of the system model and the output $y(t)$. Clearly, since the same observation can in general be produced by different states, $x(t)$ cannot always be uniquely reconstructed.

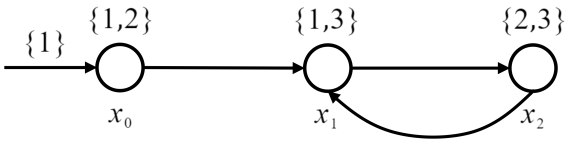

Figure 1: The switching output automaton

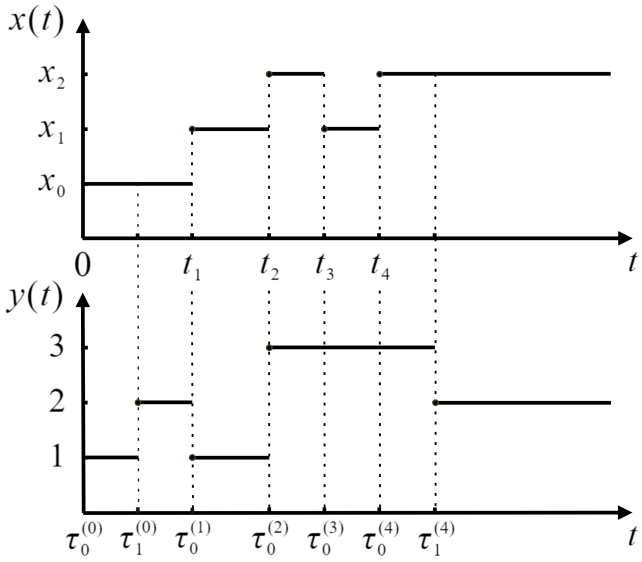

Figure 2: One possible evolution of the switching output automaton

Example 1. *Let us consider the SOA $G = (X, Y, B, h, x_0, y_0)$ in Fig. 1, with states set $X = \{x_0, x_1, x_2\}$, the output alphabet $Y = \{1, 2, 3\}$, the arcs set $B = \{(x_0, x_1), (x_1, x_2), (x_2, x_1)\}$, output function defined by $h(x_0) = \{1, 2\}, h(x_1) = \{1, 3\}, h(x_2) = \{2, 3\}$, initial state is x_0 and initial output symbol y_0 is 1. The set of direct successors of x_0, x_1, and x_2 are $\sigma(x_0) = \{x_1\}$, $\sigma(x_1) = \{x_2\}$, and $\sigma(x_2) = \{x_1\}$.*

Fig. 2 shows the evolution of the state $x(t)$ and the output $y(t)$ corresponding to the following run:

$$\overset{t_0=0}{\underset{\tau_0^{(0)}=0}{\longrightarrow}} \begin{pmatrix} x_0 \\ 1 \end{pmatrix} \underset{\tau_1^{(0)}}{\longrightarrow} \begin{pmatrix} x_0 \\ 2 \end{pmatrix} \overset{t_1}{\underset{\tau_0^{(1)}}{\longrightarrow}} \begin{pmatrix} x_1 \\ 1 \end{pmatrix} \overset{t_2}{\underset{\tau_0^{(2)}}{\longrightarrow}} \begin{pmatrix} x_2 \\ 3 \end{pmatrix} \overset{t_3}{\underset{\tau_0^{(3)}}{\longrightarrow}} \begin{pmatrix} x_1 \\ 3 \end{pmatrix} \overset{t_4}{\underset{\tau_0^{(4)}}{\longrightarrow}} \begin{pmatrix} x_2 \\ 3 \end{pmatrix} \underset{\tau_1^{(4)}}{\longrightarrow} \begin{pmatrix} x_2 \\ 2 \end{pmatrix}$$

We can find three types of transitions in the above run. At time $\tau_1^{(0)}$, the system transitions from state $(x_0, 1)$ to state $(x_0, 2)$, which corresponds to Type 2 transitions. The system moves from state $(x_0, 2)$ to state $(x_1, 1)$ at time t_1, a situation that is consistent with Type 3 transitions. The system moves from state $(x_2, 3)$ to state $(x_1, 3)$ at time t_3, a situation consistent with Type 1 transitions.

3 The Observer

Given a sequence of observations $\omega \in (Y, \mathbb{R}_{\geq\delta})^+$ we denote as $C(\omega)$ the set of states in which the system can be.

In this section we explain in detail the procedure for constructing the observer of an SOA, i.e., a structure that allows one to compute the set $C(\omega)$ for any observed sequence $\omega \in (Y, \mathbb{R}_{\geq\delta})^+$ without online computation. In this way $C(\omega)$ can be computed simply following a path in a deterministic automaton.

To describe the evolution of an SOA G, we construct a nondeterministic *evolution automaton* G_e whose global state describes the current discrete state and the current output value. In addition, the evolution of the SOA is constrained by the notion of dwell time, since unless the dwell time has expired, no change of state or output is possible. Thus each global state $q = (x, y) \in X \times Y$ needs to be further refined to specify if the dwell time constraint has already been satisfied.

To keep track of this, we associate with each discrete state $x \in X$ two new states x' and x'': the first one denotes that the current state is x but the dwell time constraint has not yet been satisfied, while the second one denotes the current state is x and the dwell time constraint has been satisfied. We define $X' = \{x' | x \in X\}$ the set of *waiting discrete states* and

$X'' = \{x''|x \in X\}$ the set of *ready discrete states*. The set of states of the evolution automaton is a set $Q \subseteq X_e \times Y$ where $X_e = X' \cup X''$.

Definition 2. *Given an SOA $G = (X, Y, B, h, x_0, y_0)$, its* evolution automaton *is a nondeterministic finite automaton $G_e = (Q, Y_e, \Delta, q_0)$ where*

- *Q is a finite state set, $q = (x_e, y) \in Q$ where $x_e \in X_e = X' \cup X'' = \{x'|x \in X\} \cup \{x''|x \in X\}$ and $y \in Y$;*
- *$Y_e = Y \cup \{\delta\}$ is the alphabet;*
- *$\Delta \subseteq Q \times Y_e \cup \{\varepsilon\} \times Q$ is the transition relation;*
- *$q_0 = (x'_0, y_0)$ is the initial state.*

From a discrete state no change of state or output is possible until the dwell time expires and the waiting state becomes ready. In the evolution automaton we denote this by a transition labeled δ from a state (x', y) to a state (x'', y).

From a ready discrete state x three types of transitions may occur: type 1 (state change with no output change), type 2 (simultaneous state and output change), type 3 (output change with no state change).

When in G a transition of type 1 occurs from a discrete state x to a discrete state $\bar{x}$, the output does not change and an external observer cannot detect its occurrence. In the evolution automaton we denote this by a transition labeled with the empty string ε from a state (x'', y) to state $(\bar{x}', y)$ where $x \neq \bar{x}$.

When a transition of type 2 or 3 occurs, the output changes from y to $\bar{y}$ and thus an external agent detects that a transition has occurred. However, since the output sets of different states may have a non-null intersection the output agent may not be able to detect if the state has also changed or, if it has changed, which is the new state. We denote this with a transition from (x'', y) to $(\bar{x}', \bar{y})$ labeled with $\bar{y} \in Y$.

Algorithm 1 describes how an evolution automaton can be constructed. Starting from an initial state, new states are added considering, in the following order, the different situations.

Algorithm 1: Constructing the evolution automaton

Input: $G = (X, Y, B, h, x_0, y_0)$, a switching output automaton
Output: $G_e = (Q, Y_e, \Delta, q_0)$, the evolution automaton

Set $X_e = X' \cup X'' = \{x'|x \in X\} \cup \{x''|x \in X\}$;
Set $Y_e = Y \cup \{\delta\}$;
Set $q_0 = (x'_0, y_0)$, $Q_{new} = \{q_0\}$, $Q = \emptyset$;
Set $\Delta = \emptyset$;
while $Q_{new} \neq \emptyset$ **do**
    select a state $q = (x_e, y) \in Q_{new}$;
    if $x_e = x' \in X'$ **then**
        $\bar{q} = (x'', y)$, $\Delta = \Delta \cup \{(q, \delta, \bar{q})\}$;
        if $\bar{q} \notin Q_{new} \cup Q$ **then**
            $Q_{new} = Q_{new} \cup \{\bar{q}\}$;
    else if $x_e = x'' \in X''$ **then**
        forall $\bar{x} \in \sigma(x)$ **do**
            if $y \in h(\bar{x})$ **then**
                $\bar{q} = (\bar{x}', y)$, $\Delta = \Delta \cup \{(q, \epsilon, \bar{q})\}$;
                if $\bar{q} \notin Q_{new} \cup Q$ **then**
                    $Q_{new} = Q_{new} \cup \{\bar{q}\}$;
            forall $\bar{y} \in h(\bar{x}) \setminus \{y\}$ **do**
                $\bar{q} = (\bar{x}', \bar{y})$, $\Delta = \Delta \cup \{(q, \bar{y}, \bar{q})\}$;
                if $\bar{q} \notin Q_{new} \cup Q$ **then**
                    $Q_{new} = Q_{new} \cup \{\bar{q}\}$;
        forall $\bar{y} \in h(x) \setminus \{y\}$ **do**
            $\bar{q} = (x', \bar{y})$, $\Delta = \Delta \cup \{(q, \bar{y}, \bar{q})\}$;
            if $\bar{q} \notin Q_{new} \cup Q$ **then**
                $Q_{new} = Q_{new} \cup \{\bar{q}\}$;
    $Q_{new} = Q_{new} \setminus \{q\}$, $Q = Q \cup \{q\}$;

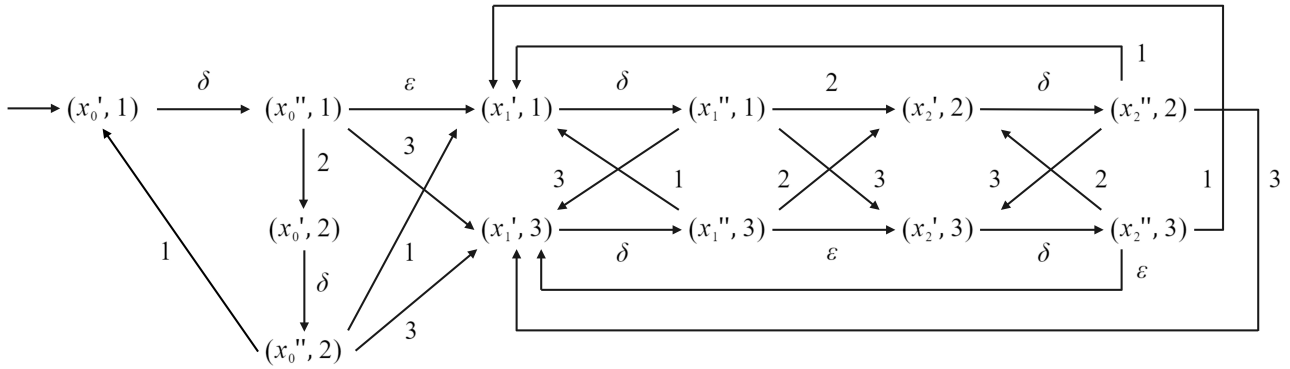

Figure 3: The evolution automaton of the switching output automaton shown in Fig. 1

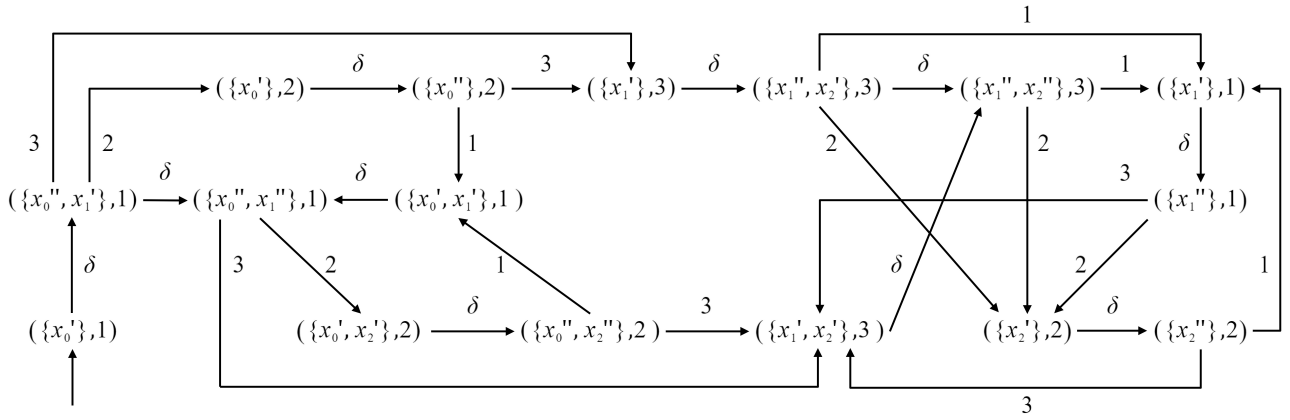

Figure 4: The observer of the switching output automaton shown in Fig. 1

- New states could be created because a δ event occurs. This may only occur at states of the evolution automaton having the first entry in X' and lead to new states having the first entry in X'' and the same second entry (line 6 to line 11).
- New states could be created based on type 1 transitions (line 12 to line 19).
- New states could be created based on type 2 transitions (line 20 to line 26).
- New states could be created based on type 3 transitions (line 27 to line 33).

Example 2. *Consider the SOA shown in Fig. 1. The evolution automaton is shown in Fig. 3.*

The evolution automaton G_e is a nondeterministic structure which describes all possible runs of an SOA G. An equivalent deterministic automaton, that we call *observer*, can be used to estimate the set of states consistent with any observation. In the following we denote the *observer* of G as $G_{obs} = (Z, Y_e, \Delta_o, z_0)$ where

- $Z \subseteq 2^{X_e} \times Y$ is a finite state set ($X_e = X' \cup X''$);
- Y_e is the alphabet;
- $\Delta_o : Z \times Y_e \to Z$ is a partial transition function;
- $z_0 = (\{x_0'\}, y_0)$ is the initial state.

Based on the above definition, each state of the observer is defined as a pair whose first entry is a subset of X_e and whose second entry is the current output. If the first entry of a state contains both the waiting and ready states x' and x'' associated with a discrete state $x \in X$, we simply cut it keeping only x''. In fact, such an observer state can be reached by different runs where state x could be either waiting or ready. However, since anything that could happen in x' could also happen in x'' and since we need to keep track of the waiting/ready state we can simply ignore x'. Algorithm 2 is presented as a way to remove state x' when also state x'' is present. However, this is also changing the observer state from

$z' \subseteq Q \subseteq (X_e, y) \times Y$ to $z \in 2^{X_e} \times Y$. We define ζ as function: $\zeta : 2^Q \to Z$ where Q is the set of states of the evolution automaton G_e and Z is the set of states of the observer G_{obs}. We define $Q_{sub} \subset Q$ as the set of global states with the same label and ζ is only defined for Q_{sub}.

Example 3. *Consider the evolution automaton G_e in Fig. 3. We set $Q_{sub} = \{(x_0'', 1), (x_1', 1), (x_1'', 1)\}$. We can obtain $X_s(Q_{sub}) = \{x_0'', x_1', x_1''\}$. Since x_1'' and x_1' correspond to the same state x_1, we can update $X_s(Q_{sub}) = \{x_0'', x_1''\}$. And $\zeta(Q_{sub}) = (\{x_0'', x_1''\}, 1)$.*

Algorithm 2: Computing $\zeta(Q_{sub})$

Input: Q_{sub}, set of global states with same label y of G_e
Output: $\zeta(Q_{sub})$, simplified state

$X_s(Q_{sub}) = \{x_e | q = (x_e, y) \in Q_{sub}\}$;
foreach $x_e, \bar{x_e} \in X_s(Q_{sub})$ **do**
  if $x_e = x' \in X', \bar{x_e} = \bar{x}'' \in X''$ *and* $x = \bar{x}$ **then**
    $X_s(Q_{sub}) = X_s(Q_{sub}) \setminus \{x_e\}$;
return $\zeta(Q_{sub}) = (X_s(Q_{sub}), y)$;

Algorithm 3 describes the construction of the observer and makes use of the following notation.

- For all states $q \in Q$ of G_e we define $D_\epsilon(q) = \{\bar{q} \in Q \mid (q, \epsilon, \bar{q}) \in \Delta^*\}$ as the set containing all states reachable from q executing zero or more ϵ-transitions. Note that by definition $q \in D_\epsilon^*(q)$.
- For all states $q \in Q$ of G_e and for all symbols $y \in Y_e$ we define $D_y(q) = \{\bar{q} \in Q \mid (q, y, \bar{q}) \in \Delta\}$ as the set containing all states reachable from q executing exactly one observable y-transition.
- For all sets of states $Q_{sub} \subseteq Q$ and for all symbols $y \in Y_e$, we define

$$\alpha(Q_{sub}, y) = \bigcup_{q \in Q_{sub}} D_y(q)$$

  as the set of states containing the states reachable in G_e from a state $q \in Q_{sub}$ executing exactly one observable y-transition. We define

$$\beta(Q_{sub}, y) = \bigcup_{q \in \alpha(Q_{sub}, y)} D_\epsilon(q)$$

  as the set containing all states reachable in G_e from a state $q \in \alpha(Q_{sub}, y)$ executing zero or more ϵ-transitions.

For each iteration we compute two sets. The first, denoted $\bar{z}'$, is the set of states consistent with the sequence $s \in Y_e^*$. The second, denoted $\bar{z}$, is a simplification of the previous set and will be the new state of the observer. The state $\bar{z}'$ is calculated in two steps. First to compute $\alpha(z', y)$ where $z' \subseteq Q$ and $y \in Y_e$; then, compute $\beta(z', y)$, and $\bar{z}' = \beta(z', y)$.

It is worth noting that since the global state $q_e = (x'', y)$ of the evolution automaton G_e does not generate a new state after activating the event δ, we omit the selfloop of the event δ at q_e. However the physical meaning is that the system remains in the same state after a time equal to the dwell time has elapsed. Therefore, in the string s whenever an event δ occurs, the new state $\bar{z}'$ contains all states $q_e = (x'', y)$ of the old state z' that has elapsed dwell time δ.

Note that each state of the observer $z = (X_z, y) \in Z$ is such that $X_z \subseteq X_e = X' \cup X''$, i.e, it contains only states of the evolution automaton. We define $g(X_z) = \{x \in X \mid \{x', x''\} \cap X_z \neq \emptyset\}$, i.e., the set of states of G which correspond to the states of X_z.

When a sequence ω is observed, we can follow the corresponding path in the observer taking into account that events δ correspond to the elapsing of an interval of time equal to δ while the current output does not change. Let $z = (X_z, y)$ be the state of the observer reached following the observation ω, the set of states consistent with ω is $C(\omega) = g(X_z)$.

Example 4. *Consider the SOA depicted in Fig. 1, whose evolution automaton is shown in Fig. 3. The observer is shown in Fig. 4. Assume $\delta = 0.1$ and let the observation be $\omega = (1, 0.15)(3, 1)$. The state of the observer reached following ω is $z = (\{x_1'', x_2''\}, 3)$, which corresponds to a set of consistent states $C(\omega) = g(\{x_1'', x_2''\}) = \{x_1, x_2\}$. The state of the observer z is reached visiting the intermediates states $(\{x_0'\}, 1)$, $(\{x_0'', x_1'\}, 1)$, $(\{x_1'\}, 3)$, $(\{x_1'', x_2'\}, 3)$.*

Due to Algorithm 2, the number of states of the observer is $|Z| < 2^{(|X_e| \times |Y|)}$. Therefore, the space complexity of the proposed approach is $O < 2^{(2|X| \times |Y|)}$.

Algorithm 3: Constructing the observer

Input: $G_e = (Q, Y_e, \Delta, q_0)$, the evolution automaton
Output: $G_{obs} = (Z, Y_e, \Delta_o, z_0)$, the observer

Set $z_0' = D_\epsilon(q_0)$, $Z' = \{z_0'\}$, assign no tag to z_0';
Set $z_0 = z_0'$, $Z = \{z_0\}$;
while *states with no tag exists in* Z' **do**
  select a state $z' \in Z'$ with no tag;
  foreach $y \in Y_e$ **do**
    if $y{=}\delta$ **then**
      $\alpha(z', y) = \bigcup_{q \in z'} D_y(q) \cup (z' \cap X'')$;
    else
      $\alpha(z', y) = \bigcup_{q \in z'} D_y(q)$
    $\beta(z', y) = \bigcup_{q \in \alpha(z', y)} D_\epsilon(q)$;
    $\bar{z}' = \beta(z', y)$;
    if $\bar{z}' \notin Z'$ **then**
      $Z' = Z' \cup \{\bar{z}'\}$, assign no tag to $\bar{z}'$;
    $z = \zeta(z')$;
    $\bar{z} = \zeta(\bar{z}')$, $Z = Z \cup \{\bar{z}\}$;
    if $z \neq \bar{z}$ **then**
      $\Delta_o(z, y) = \bar{z}$;
  tag node z' 'old';
Remove all tags;

4 Verifying Current State Opacity

In this section, we propose a definition of current state opacity which applies to SOA and propose an approach to verify such a property using the observer.

We consider a secret set $S \subseteq X$. According to the usual notion of current state opacity, a system is opaque if for all generated observations ω, the set of consistent states $C(\omega)$ computed by an intruder, which observes the system, the set $C(\omega)$ is not a subset of the secret S.

In our setting, where observation ω is a continuous time signal and the system's evolution is constrained by a dwell time, we give a different definition. Let us first denote by Ω_s the set of *stable observations*, i.e., those observations ω, whose set of consistent states $C(\omega)$ cannot change unless a new system output is generated.

Definition 3. *A switching output automaton $G = (X, Y, B, h, x_0, y_0)$ is called* current state opaque *(CSO) with respect to a secret $S \subseteq X$, if for all stable observations $\omega \in \Omega_s$, $C(\omega) \nsubseteq S$ holds.*

This definition is motivated by the following consideration. If an observation is not stable, as time elapses without a change of the output, the set of consistent states will keep changing (every δ units of time) until after a time $T \leq n \cdot \delta$ a stable estimation is reached. We assume the intruder has no way of exploiting the information obtained from a non-stable observation because in a very short time it becomes outdated. Thus only stable observations should be considered for current state opacity verification.

One can readily verify that any sequence $\omega \in \Omega_s$ reaches in the observer a state that does not enable the event δ. We call such states *stable* and denote them by $Z_s = \{z \in Z \mid \Delta_o(z, \delta) \text{ is not defined}\} \subset Z$. Note that each stable state of the observer $z = (X_z, y) \in Z_s$ is such that $X_z \subseteq X''$, i.e, it contains only ready states of the evolution automaton.

Based on this, we can provide a necessary and suffient condition to verify current state opacity for an SOA.

Proposition 1. *Consider a switching output automaton G and a secret set S. Let Z_s be the set of stable states of its observer. The SOA G is current state opaque wrt S iff for all $z = (X_z, y) \in Z_s$, it holds that $g(X_z) \nsubseteq S$.*

Proof. ($\Rightarrow$) Assume that G is CSO with respect to S. By definition of CSO, for all stable observations $\omega \in \Omega_s$, it is $C(\omega) \nsubseteq S$. Sequences $\omega \in \Omega_s$ lead to stable states in the observer, namely to states $z = (X_z, y) \in Z_s$. Therefore for all such states, it is $g(X_z) \nsubseteq S$.

($\Leftarrow$) Assume that $\exists z = (X_z, y) \in Z_s$ such that $X_z \subseteq S$. This implies that there exists a stable observation $\omega \in \Omega_s$ such that $C(\omega) \subseteq S$. Therefore, the system is not CSO. □

As a result, if we want to determine whether an SOA is CSO, we need to focus on the stable states of the observer. We have to verify whether for at least one of them, the set of states of the system associated with it is a subset of the secret S. If such is the case, the SOA is not opaque with respect to S.

Example 5. *Consider the system in Fig.1. Let $S = \{x_2\}$. Since there exists a stable state $(\{x_2''\}, 2)$ of the observer such that $g(\{x_2''\}) = \{x_2\} = S$, the system is not CSO.*

5 Conclusions and Future Work

In this paper, we propose a new discrete event model called switching output automaton. Each state of the automaton can produce multiple discrete outputs. We design an observer to estimate the current state of such a model based on output observations consisting of piecewise-constant signals. We use the state observer to verify current state opacity of the system. In the future, we will further investigate state-based opacity for switching output automata with continuous state output values.

References

[1] C. G. Cassandras and S. Lafortune, Introduction to Discrete Event Systems. Cham: Springer International Publishing, 2021.

[2] C. Seatzu, M. Silva, and J. H. van Schuppen, Eds., Control of Discrete-Event Systems, vol. 433. London: Springer London, 2013.

[3] E. F. Moore, Experiments on Sequential Machines, in Automata Studies. (AM-34), C. E. Shannon and J. McCarthy, Eds. Princeton University Press, pp. 129-154, 1956.

[4] D. Thorsley, A necessary and sufficient condition for diagnosability of stochastic discrete event systems, Discrete Event Dynamic Systems, vol. 27, no. 3, pp 481–500, 2017.

[5] C. Kai, A. Giua and C. Seatzu, Consistent reduction in discrete-event systems, Automatica, vol. 142, 2022.

[6] J. W. Bryans, M. Koutny, and P. Y. Ryan, Opacity using Petri nets, Electronic Notes in Theoretical Computer Science, vol. 121, pp. 101-115, 2005.

[7] A. Saboori and C. N. Hadjicostis, Security and opacity in discrete event systems, in Proceedings of the 46th IEEE Conference on Decision and Control. IEEE, pp. 5056-5061, 2007.

[8] A. Saboori and C. N. Hadjicostis, Verification of initial-state opacity in security applications of discrete event systems, Information Sciences, vol. 246, pp. 115-132, 2013.

[9] Y. Wu and S. Lafortune, Comparative analysis of related notions of opacity in centralized and coordinated architectures, Discrete Event Dynamic Systems, vol. 23, no. 3, pp. 307-339, 2013.

[10] Y. Falcone and H. Marchand, Enforcement and validation (at runtime) of various notions of opacity, Discrete Event Dynamic Systems, vol. 25, no. 4, pp. 531-570, 2015.

[11] Y. Tong, Z. Li, C. Seatzu, and A. Giua, Verification of State-Based Opacity Using Petri Nets, IEEE Trans. Automat. Contr., vol. 62, no. 6, pp. 2823-2837, Jun. 2017.

[12] Z. Zhang, C. Xia, and S. Chen, Security and privacy with opacity based state observation for finite state machine, Asian Journal of Control, vol. 24, no. 2, pp. 614-625, Mar. 2022.