Synchronizing Sequence Computation Under Forbidden Event Constraints

Khalid Hamada[1], Rabah Ammour[1], and Isabel Demongodin[1]

***Abstract*— This paper addresses the problem of driving a cyber-physical system (CPS) to a known state without knowledge of its current state, assuming it has suffered a cyber-attack that compromises state estimation and necessitates guiding it to a secure state. To achieve this, we analyze the CPS using a labeled finite-state automaton with inputs derived from an output synchronized Petri net that models the CPS, capturing both its dynamics and controller-plant communication. We propose a method for computing a synchronizing sequence (SS) under forbidden event constraints, i.e., a control sequence that drives the CPS to a target state without requiring knowledge of its current state while avoiding specific forbidden events. First, we address forbidden control inputs by pruning transitions in the labeled finite state automaton, removing those associated with forbidden events. Second, we handle forbidden sensor outputs by introducing a modified greedy algorithm based on depth-first search to compute a constrained SS. Finally, a case study involving autonomous mobile robots is considered to illustrate the effectiveness of our approach in computing an SS under both input and output constraints.**

I. Introduction

The security analysis of cyber-physical systems (CPS) has attracted increasing attention, largely due to their reliance on networked communication, which exposes them to various cyberattack vulnerabilities. To address these challenges, discrete event system (DES) abstractions [1] have proven valuable for analyzing CPS security [2]. In certain compromised scenarios, the current state of the CPS may become unknown or lost. In such cases, it is critical to develop recovery strategies that ensure the system can be guided back to a known and secure state.

The problem of driving a DES into a known state using a predefined sequence of control inputs has been extensively studied in automata theory [3]. Several approaches have been developed to address this problem, most notably synchronizing sequences (SS) and homing sequences (HS) [4]. A synchronizing sequence is an input sequence that, when applied to a system, brings it to a specific known state regardless of its initial state. Originally introduced for finite state machines, the concept of SS has since been extended to synchronized Petri nets [5], [6], [7]—a subclass of Petri nets that incorporates control input events [8]. Homing sequences offer an alternative strategy by enabling state determination through output observation while executing a control input sequence [9]. Unlike SS, which unconditionally force the system into a predetermined state, HS rely on state identification through output analysis.

The work in [10] addresses the problem of guiding a system, modeled as a finite automaton with unknown or partially known initial states, to a known state using adaptive input sequences. These sequences are dynamically constructed and adjusted based on the system's observed responses during execution. In our previous work [11], we have used output synchronized Petri nets (OutSynPN), an extension of traditional Petri nets that enables the modeling of CPS along with the information flow between the controller and the physical components. From an attacker's perspective, the objective was to compute a synchronizing sequence that drives the OutSynPN from an unknown initial state to a deadlock state.

In this paper, we adopt a controller's perspective and address the problem of recovering a CPS when the current state estimation is lost—such as in the aftermath of a cyberattack. We further assume that, in the compromised scenario, certain control inputs may be unavailable, and/or specific sensor outputs must be avoided due to security constraints. The goal is to compute an SS that respects both forbidden control inputs and restricted sensor outputs. Our method relies on labeled finite state automata with inputs (LFAI) derived from OutSynPN. In this abstraction, nodes represent the system's reachable states, while transitions are labeled with both control inputs and corresponding sensor outputs. To handle forbidden control inputs, we modify the LFAI by pruning transitions associated with forbidden events. For forbidden sensor outputs, we propose an enhanced greedy algorithm based on depth-first search (DFS), designed to compute a constrained SS that avoids generating undesired observations. The effectiveness of the proposed approach is demonstrated through a case study involving autonomous mobile robots, showcasing the computation of SS under both input and output constraints.

This paper is structured as follows. Section II provides the necessary background on the LFAI formalism and greedy algorithm used for SS computation. Section III focuses on SS computation under forbidden input constraints. Section IV extends the analysis to SS computation under forbidden output constraints. Finally, Section V presents a case study involving autonomous mobile robots to demonstrate the computation of SS under combined input and output constraints.

II. Backgrounds

In our previous work [12], we have introduced a specific class of Petri nets, called *output synchronized Petri nets (OutSynPN)*, which enable the modeling of CPS along with

This work is parially funded by the French National Research Agency under grant agreement ANR-22-CE10-0002.

[1]Authors are with Aix-Marseille Université, CNRS, LIS, Marseille, France.

its information flow. To analyze these Petri net models, we subsequently derived *labeled finite state automata with inputs (LFAI)*, a graph where nodes represent the system's state, while transitions encode control inputs and sensor outputs. In this section, we first recall the definition of an LFAI before introducing the completely specified LFAI. Next, we introduce a formal definition of an auxiliary graph and review the method, based on this graph, for computing a synchronizing sequence.

A. Labeled finite state automaton with inputs

Definition 1: A *labeled finite state automaton with inputs (LFAI)* is defined as a 7-tuple: $G = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$, where:

- X is a finite set of states.
- E is a finite set of input events, with $E_\lambda = E \cup \{\lambda\}$, where λ represents an *internal* or "always occurring" event.
- $\delta : X \times E_\lambda \to X$ is the (possibly partially defined) transition function.
- $x_0 \in X$ denotes the initial state.
- Q is a finite set of output labels, with $Q_\varepsilon = Q \cup \{\varepsilon\}$, where ε signifies the absence of a label.
- $Obs : X \times E_\lambda \to 2^{Q_\varepsilon}$ is the labeling function, mapping state and event pairs to output labels.
- $X_f = X$ is the finite set of marked states. ■

We consider that the LFAI is *deterministic* with respect to input events, i.e., $\forall x \in X, \forall e \in E_\lambda, \delta(x, e)$ is at most of cardinality 1. Note that an LFAI could be reduced with respect to the λ events [12]. In the rest of the paper and without loss of generality, we apply and illustrate the proposed approaches directly to the reduced LFAI such that both functions δ and Obs are defined over E rather than E_λ. Moreover, the labeling function is restricted to the set Q_ε such that $Obs : X \times E_\lambda \to Q_\varepsilon$.

We refer to an LFAI as a *completely specified* LFAI when the transition function $\delta(x, e)$ is defined for all states $x \in X$ and all events $e \in E$. To complete an incompletely specified LFAI, self-loops are added such that for any state $x \in X$ where $\delta(x, e)$ is undefined for some $e \in E$, we set $\delta(x, e) = x$ and assign $Obs(x, e) = \varepsilon$. The resulting automaton is a completely specified LFAI, denoted by $\tilde{G}$. We denote by E^* is the Kleene closure of the set E, and by ϵ the empty sequence. The transition function is extended to input sequences $\omega \in E^*$ through the function δ^*. Finally, the set of outputs generated by a given input sequence $\omega = e_1 e_2 \dots e_k \in E^*$ applied from state x is given by $Obs^*(x, \omega) = \bigcup_{e_i \in \omega} Obs(\delta^*(x, e_1 \dots e_{i-1}), e_i)$.

B. Synchronizing sequence computation

For an LFAI G, a *synchronizing sequence* (SS) for a given target state $\bar{x} \in X$, denoted $\bar{\omega} \in E^*$, is an input control sequence that drives the LFAI G to $\bar{x}$ regardless of its initial state, i.e., $\forall x \in X, \delta^*(x, \bar{\omega}) = \bar{x}$.

An important step in SS computing is the construction of an auxiliary graph [13] associated to a completely specified LFAI $\tilde{G}$. This graph consists of $(|X| \cdot (|X|+1)/2)$ nodes, each representing an unordered couple of states $\{x_i, x_j\}$ within $\tilde{G}$, denoted as $\upsilon_{ij} = \upsilon_{ji}$. When a given pair includes identical states, i.e., $\{x_i, x_i\}$, it is simply given by $\{x_i\}$ and denoted υ_i. To take into account the outputs of the LFAI, we introduce the definition of the auxiliary graph as follows.

Definition 2: Let $\tilde{G} = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$ be a completely specified LFAI, its *auxilary graph with outputs* is given by $\mathcal{A}(\tilde{G}) = \langle \Upsilon, E, \delta^A, \upsilon_0, Q, Obs^A, \Upsilon_f \rangle$, where:

- Υ is a finite set of states $\upsilon_{ij} = \upsilon_{ji} = \{x_i, x_j\}$.
- E is a finite set of input events.
- $\delta^A : \Upsilon \times E \to \Upsilon$ is the transition function such that $\delta^A(\upsilon_{ij}, e) = \upsilon_{i'j'}$ if $\{\delta(x_i, e), \delta(x_j, e)\} = \{x'_i, x'_j\}$.
- $\upsilon_0 = \{x_0, x_0\} = \{x_0\} \in \Upsilon$ denotes the initial state.
- Q is a finite set of output labels, with $Q_\varepsilon = Q \cup \{\varepsilon\}$, where ε signifies the absence of a label.
- $Obs^A : \Upsilon \times E \to Q_\varepsilon \times Q_\varepsilon$ is the labeling function such that $Obs^A(\upsilon_{ij}, e) = \{Obs(x_i, e), Obs(x_j, e)\}$.
- $\Upsilon_f = \Upsilon$ is a finite set of marked states. ■

For a given LFAI G, its completely specified LFAI $\tilde{G}$ and, its corresponding auxiliary graph $\mathcal{A}(\tilde{G})$, the greedy Algorithm 1 [5] is used for the determination of an SS, when it exists, that drives G to a given target state $\bar{x}$.

Algorithm 1: Greedy algorithm for SS computation

Input: Completely specified LFAI $\tilde{G}$, Auxiliary graph with outputs $\mathcal{A}(\tilde{G})$ and, target state $\bar{x} \in X$.
Output: An SS $\bar{\omega}$ for state $\bar{x}$

```
Let i ← 0; ω_0 ← ϵ; φ(ω_0) ← X;
while φ(ω_i) ≠ {x̄} do
    i ← i + 1;
    Pick two states x_p, x_q ∈ φ(ω_{i−1}) such that p ≠ q;
    if There does not exist any path in A(G̃) from node υ_pq to node ῡ = {x̄, x̄}; then
        Stop the computation; there exists no SS for x̄.
    else
        Find the shortest path from node υ_pq to ῡ and let ω be the input sequence along this path, do
        ω_i ← ω_{i−1}ω;
        φ(ω_i) ← δ*(φ(ω_{i−1}), ω);
ω̄ ← ω_i
```

III. Synchronizing sequence computation under forbidden input constraints

In this section, we address the problem of computing a synchronizing sequence (SS) under forbidden input constraints (FIC). This implies that certain control inputs, denoted by $\mathcal{F}_{\text{input}} \subset E$, cannot be used for some reason. To handle this constraint, we propose a method based on arc pruning to determine, when possible, the existence of a valid synchronizing sequence. This computation is based on a modified LFAI.

A. SS under FIC and modified LFAI

Let us first define an SS under FIC $\mathcal{F}_{\text{input}} \subseteq E$.

Definition 3 (SS under FIC): Consider a completely specified LFAI $\tilde{G} = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$, a target state $\bar{x} \in X$ and, a given set of forbidden inputs $\mathcal{F}_{\text{input}} \subseteq E$. A *synchronizing sequence under forbidden input constraints*, denoted $\bar{\omega}$, for state $\bar{x}$ is an SS that does not contain any event in $\mathcal{F}_{\text{input}}$, i.e., $\forall x \in X, \delta^*(x, \bar{\omega}) = \bar{x}$ and $\forall e \in \bar{\omega}, e \notin \mathcal{F}_{\text{input}}$. ■

The computation of an SS under FIC is based on a Modified LFAI (MLFAI). We herein present its definition.

Definition 4 (Modified LFAI): Let $G = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$ be an LFAI and, $\mathcal{F}_{\text{input}}$ a set of forbidden inputs. A *Modified Labeled Finite Automaton with Inputs* (MLFAI) is given by $G_m = \langle X, E_m, \delta_m, x_0, Q_m, Obs_m, X_f \rangle$ with:

- $E_m = E \setminus \mathcal{F}_{\text{input}}$ is the finite set of authorized input events.
- $\delta_m : X \times E_m \to X$ is the (possibly partially defined) transition function such that $\delta_m(x, e) = \delta(x, e)$ if $e \in E_m$ and undefined otherwise.
- $Q_m \subseteq Q$ is a subset of output labels and $Q_{\varepsilon_m} = Q_m \cup \{\varepsilon\}$.
- $Obs_m : X \times E_m \to Q_{\varepsilon_m}$ is the labeling function such that $Obs_m(x, e) = Obs(x, e)$ if $e \in E_m$ and undefined otherwise. ■

B. Computation method of SS under FIC based on arc prunning

Given an LFAI G, a target stat $\bar{x}$ and, a set of forbidden control inputs $\mathcal{F}_{\text{input}}$, the following steps are proposed to determine, when it exists, an SS under FIC.

1) Compute G_m, the MLFAI of G with respect to $\mathcal{F}_{\text{input}}$;
2) Deduce $\tilde{G}_m$, the completely specified MLFAI by completing G_m with respect to E_m;
3) Construct the corresponding auxiliary graph with outputs $\mathcal{A}(\tilde{G}_m)$;
4) Determine by greedy Algorithm 1 an SS under FIC $\mathcal{F}_{\text{input}}$, if it exists, for target state $\bar{x}$ using $\tilde{G}_m$ and $\mathcal{A}(\tilde{G}_m)$ as inputs.

If step 4) above fails to return an SS for MLFAI G_m then no SS satisfying FIC $\mathcal{F}_{\text{input}}$ exists in the original LFAI G. This claim is established by the following proposition.

Proposition 1: Consider an LFAI $G = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$, a target state $\bar{x} \in X$ and, a given set of forbidden inputs $\mathcal{F}_{\text{input}} \subseteq E$. Let $\tilde{G}_m = \langle X, E_m, \delta_m, x_0, Q_m, Obs_m, X_f \rangle$ be its completely specified MLFAI. If an SS does not exist under $\tilde{G}_m$ for target state $\bar{x}$, then an SS under FIC $\mathcal{F}_{\text{input}}$ does not exist under G.

Proof: We proceed by contradiction. Suppose the greedy algorithm applied to $\tilde{G}_m$ does not return any SS for reaching $\bar{x}$, i.e., no SS exists in G_m for $\bar{x}$. Assume, for contradiction, that there exists an SS $\bar{\omega} \in E^*$ in G for target state $\bar{x}$ such that for all $e \in \bar{\omega}$, $e \notin \mathcal{F}_{\text{input}}$. That is, $\bar{\omega}$ is a valid SS under FIC in G. From Definition 4 of a MLFAI, we know that $E_m = E \setminus \mathcal{F}_{\text{input}}$ and $\delta_m(x, e) = \delta(x, e)$ if $e \in E_m$. Since $\bar{\omega}$ contains no events from $\mathcal{F}_{\text{input}}$, $\bar{\omega}$ is also a valid SS in G_m and should lead to the target state $\bar{x}$. This implies that an SS exists in G_m for $\bar{x}$, contradicting our initial assumption. Therefore, we conclude that if no SS exists in G_m, then no SS under FIC exists in G. ■

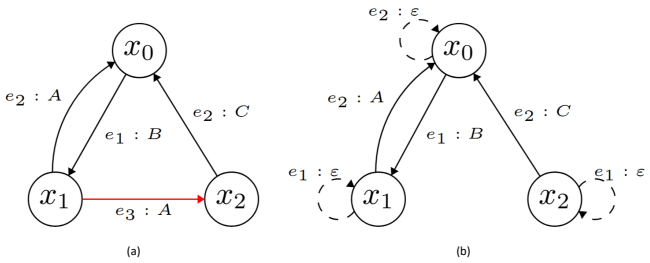

Fig. 1. LFAI G_1 (a) and its completely specified MLFAI $\tilde{G}_{m_1}$ (b).

Example 1: Consider the LFAI G_1 shown in Figure 1-(a) where $X = \{x_0, x_1, x_2\}$, $E = \{e_1, e_2, e_3\}$, $Q = \{A, B, C\}$. The initial state is not indicated and assumed to be unknown. The objective is to compute an SS under FIC $\mathcal{F}_{\text{input}} = \{e_3\}$ that leads to target state x_1. The completely specified MLFAI $\tilde{G}_{m_1}$ is shown in Figure 1-(b). It is obtained by: removing the arc associated with e_3 and completing the graph with self-loops. Its corresponding auxiliary graph with outputs $\mathcal{A}(\tilde{G}_{m_1})$ is constructed and represented on Figure 2. Using the greedy algorithm, an SS under FIC $\{e_3\}$ is computed $\bar{\omega} = e_2 e_1$. This sequence successfully drives the original LFAI G_1 to the target state x_1 from any initial state, ensuring that the forbidden event e_3 is excluded. ♢

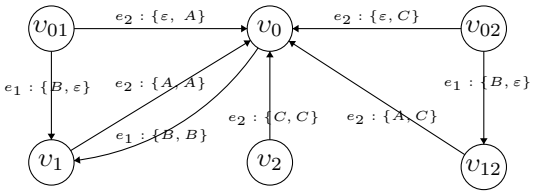

Fig. 2. Auxiliary graph with outputs $\mathcal{A}(\tilde{G}_{m_1})$

IV. Synchronizing sequence computation under forbidden output constraints

In this section, the problem of computing an SS under forbidden output constraints (FOC) is addressed. The goal is to compute an SS that, when applied, avoids generating certain forbidden sensor outputs included in $\mathcal{F}_{out} \subseteq Q$. To handle this constraint, we first illustrate through an example that applying an arc pruning method based on outputs generally fails in such case. Subsequently, we propose a method based on a modified greedy algorithm to compute, when possible, an SS under FOC.

A. SS under FOC and arc prunning method

Let us first define an SS under FOC $\mathcal{F}_{\text{out}} \subseteq Q$.

Definition 5 (SS Under FOC): Consider a completely specified LFAI $\tilde{G} = \langle X, E, \delta, x_0, Q, Obs, X_f \rangle$, a target state

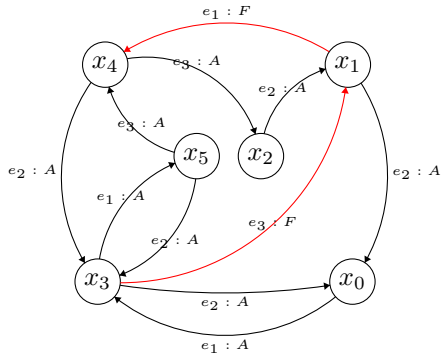

Fig. 3. LFAI G_2 with a forbidden output F associated to red arcs

$\bar{x} \in X$ and, a given set of forbidden outputs $\mathcal{F}_{\text{out}} \subseteq Q$. A *synchronizing sequence under forbidden output constraints* $\bar{\omega}$ for state $\bar{x}$ is an SS that does not generate any forbidden output in $\mathcal{F}_{out}$, i.e., $\forall x \in X, \delta^*(x, \bar{\omega}) = \bar{x}$ and $\forall q \in Obs^*(x, \bar{\omega}), q \notin \mathcal{F}_{out}$. ■

To compute an SS under FOC, a first intuitive approach — similar to the one introduced in Section III.B — would be to apply arc pruning based on forbidden outputs. This involves removing all transitions associated with outputs in the forbidden set $\mathcal{F}_{\text{out}}$. However, this method generally fails to produce a valid SS under FOC, as illustrated below.

Example 2: Consider LFAI G_2 in Figure 3 with x_2 as the target state (i.e., $\bar{x} = x_2$), and the forbidden output set defined as $\mathcal{F}_{\text{out}} = \{F\}$. As in the approach presented in Section III.B, the first step consists in removing all arcs corresponding to the forbidden output set F. An SS is then computed on the auxiliary graph derived from the resulting completely specified LFAI. The obtained sequence is $\bar{\omega} = e_3e_3e_2e_1e_1e_3e_3$. However, one can observe that if the system is initially in state x_3, the execution of $\bar{\omega}$ on the initial graph G_2 will inevitably produce the forbidden output F. This implies that while $\bar{\omega}$ is a valid SS to target state x_2 for the considered example, it does not satisfy the FOC requirement and therefore is not an SS under FOC $\{F\}$. Thus, arc pruning approach based on forbidden outputs fails to produce a valid SS under FOC. ♢

B. SS under FOC based on modified greedy algorithm

To address the problem of computing an SS under FOC, we propose Algorithm 2, which adapts the classical greedy algorithm (see Algorithm 1) to handle output restrictions. This new algorithm relies on a depth-first search (DFS) strategy to explore sequences that avoid forbidden outputs. DFS is chosen for its simplicity and practical efficiency in finding feasible solutions, even though it does not guarantee optimality in terms of sequence length. Since the exploration operates over the auxiliary graph, we introduce a disjoint unordered state pairs operator for a given set of states $X' \subseteq X$ as follows.

Definition 6 (Disjoint unordered state pairs): Let X' be a subset of states X, a complete partition into pairs of unordered states is defined as follows:

- If $|X'| = 2k$ with $k \in \mathbb{N}$, $P(X') = \{\{x_1, x_2\}, \{x_3, x_4\}, \ldots, \{x_{2k-1}, x_{2k}\}\}$ where $|P(X')| = k$ and all elements $x_i \in X'$ appear in exactly one pair.
- If $|X'| = 2k + 1$, then exactly one pair will consist of two identical states x_{2k+1} such that $P(X') = \{\{x_1, x_2\}, \{x_3, x_4\}, \ldots, \{x_{2k-1}, x_{2k}\}, \{x_{2k+1}, x_{2k+1}\}\}$. ■

Algorithm 2: Adapted Greedy Algorithm for SS under FOC computation

Input: $\mathcal{A}(\tilde{G})$, $\mathcal{F}_{\text{out}}$, $\bar{x}$
Output: *SS* $\bar{\omega}$ under FOC

1 $k \leftarrow 0$, $\omega_0 \leftarrow \epsilon$, $\phi(\omega_0) \leftarrow X$;
2 **while** $\phi(\omega_k) \neq \{\bar{x}\}$ **do**
3 $\quad k \leftarrow k + 1$;
4 $\quad$ Select $x_p, x_q \in \phi(\omega_{k-1})$ such that $p \neq q$;
5 $\quad \mathcal{S} \leftarrow$ empty stack;
6 $\quad \mathcal{S}$.push$(\upsilon_{pq}, \epsilon, \phi(\omega_k))$;
7 $\quad$ Tag υ_{pq} as *visited*;
8 $\quad$ **while** $\mathcal{S} \neq \emptyset$ ***or*** $\bar{\upsilon} = (\bar{x}, \bar{x})$ *is not visited* **do**
9 $\quad\quad (\upsilon, \omega, \Phi) \leftarrow \mathcal{S}$.pop();
10 $\quad\quad$ Select $e \in E$ such that $\delta^A(\upsilon, e)$ is not visited
11 $\quad\quad$ **if** $\bigcup_{\upsilon' \in P(\Phi)} Obs^A(\upsilon', e) \cap \mathcal{F}_{out} = \emptyset$ **then**
12 $\quad\quad\quad$ Tag $\delta^A(\upsilon, e)$ as *visited*;
13 $\quad\quad\quad \Phi \leftarrow \bigcup_{\upsilon' \in P(\Phi)} \delta^A(\upsilon', e)$;
14 $\quad\quad\quad \mathcal{S}$.push$(\delta^A(\upsilon, e), \omega e, \Phi)$;
15 $\quad$ **if** $\mathcal{S} = \emptyset$ **then**
16 $\quad\quad$ **Stop:** No SS under FOC exists for $\bar{x}$;
17 $\quad$ **else**
18 $\quad\quad (\upsilon, \omega, \Phi) \leftarrow \mathcal{S}$.pop();
19 $\quad\quad \omega_k \leftarrow \omega_{k-1}\omega$;
20 $\quad\quad \phi(\omega_k) \leftarrow \Phi$;
21 $\bar{\omega} \leftarrow \omega_k$.

The adapted greedy algorithm (Algorithm 2) computes an SS under FOC for a completely specified LFAI $\tilde{G}$, using its associated auxiliary graph $\mathcal{A}(\tilde{G})$. The algorithm proceeds iteratively, gradually reducing the current set of possible states $\phi(\omega_k)$ to the target singleton state $\{\bar{x}\}$ through successive input sequences ω_k. At each iteration, a pair of distinct states (x_p, x_q) from $\phi(\omega_{k-1})$ is selected to form the auxiliary graph state υ_{pq} (step 4), and a depth-first search (DFS) is initiated from this node using a stack-based exploration strategy.

During the DFS, transitions are only followed if the corresponding input event does not produce any forbidden output for the current set of state pairs $P(\Phi)$ (step 11). If a valid sequence is found that merges the selected state pair without violating the output constraint, the resulting input sequence is appended to the current sequence ω_{k-1} (step 19), and the reachable state set $\phi(\omega_k)$ is updated accordingly (step 20). This process is repeated until the current set of possible states $\phi(\omega_k)$ equals $\{\bar{x}\}$, or until no such sequence can be

found, in which case the algorithm terminates, indicating that an SS under FOC does not exist (step 15).

Example 3: Consider again LFAI G_2 shown in Figure 3, along with its auxiliary graph $\mathcal{A}(\tilde{G}_2)$, partially depicted in Figure 4. The objective is to compute an SS under FOC $\mathcal{F}_{\text{out}} = \{F\}$ with respect to target state x_2. Suppose Algorithm 2 selects the state pair (x_2, x_4). The stack $\mathcal{S}$ is then initialized with the tuple (v_{24}, ϵ, X). At the first level of depth, assume that event e_2 is selected. This leads to state $\delta^A(v_{24}, e_2) = v_{13}$ and no forbidden output is generated from any state pair. Specifically, we have $\bigcup_{v' \in P(X)} Obs^A(v', e_2) = Obs^A(v_{01}, e_2) \cup Obs^A(v_{23}, e_2) \cup Obs^A(v_{45}, e_2) = \{A\}$. The stack is then updated as $\mathcal{S}.\text{push}(v_{13}, e_2, \{x_0, x_1, x_3\})$ and v_{24} is tagged as visited. At the second depth, the algorithm pops the last element from the stack and evaluates an outgoing transition. It proceeds with e_2, as e_1 generates a forbidden output. Since $\delta^A(v_{13}, e_2) = v_0$ has not been visited before and $F \notin \bigcup_{v' \in P(\{x_0, x_1, x_3\})} Obs^A(v', e_2)$, the state is pushed onto the stack as $(v_0, e_2 e_2, \{x_0\})$ and tagged as visited. The algorithm continues execution and finally computes the sequence $\bar{\omega} = e_2 e_2 e_1 e_1 e_3 e_3$. As can be verified in Figure 3, this sequence is a valid SS under FOC $\{F\}$ for target state x_2. ♢

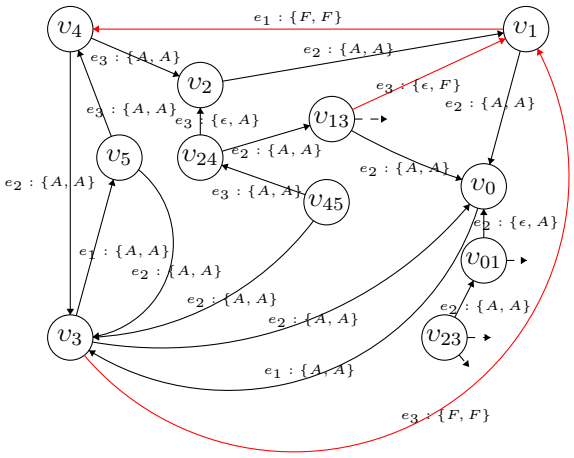

Fig. 4. Partial representation of the auxiliary graph $\mathcal{A}(\tilde{G}_2)$

The complexity of determining an SS under FOC is driven by the DFS process used in the algorithm. In the worst case, DFS time complexity is $O(b^d)$ [14], where d represents the longest path between a chosen v_{ij} and $\bar{v}$ in the auxiliary graph. Specifically, d is bounded by $(|X| \cdot (|X|+1)/2) - 1$. The maximum branching factor b corresponds to the number of possible transitions, which is bounded by $|E|$. Therefore, the resulting complexity is $O(|E|^{((|X| \cdot (|X|+1))/2)-1})$.

V. Case Study

This section presents a case study illustrating the utilization of SS under both input and output constraints. Consider two autonomous mobile robots R_1 and R_2 tasked with navigating between four workstations: 1, 2, 3, and 4 as depicted on Figure 5.

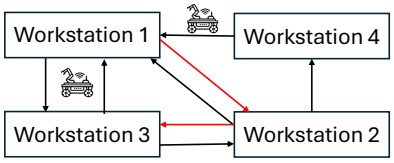

Fig. 5. The considered CPS

The robots are controlled through input commands sent via a communication network, while their states are observed through sensor outputs transmitted to the controller over the same network. The controller, robots, and communication network together constitute a cyber-physical system (CPS).

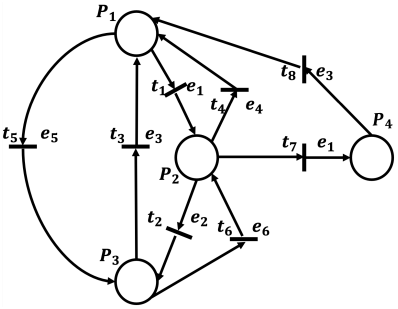

Fig. 6. Output synchronized Petri net model of the considered CPS

Label	Formal definition	Physical meaning
F_1	$(\downarrow M(P_1) \wedge \uparrow M(P_2)) \vee (\downarrow M(P_1) \wedge \uparrow M(P_4))$	A robot leaves workstation 1 and enters workstation 2
F_2	$\downarrow M(P_2) \wedge \uparrow M(P_3)$	A robot leaves workstation 2 and enters workstation 3
ε	–	All remain scenarios of movements

TABLE I
CPS OUTPUTS

The CPS is modeled using an OutSynPN given by Figure 6, where places represent the physical locations, i.e., the workstations. The transitions are associated with movement commands, and the outputs correspond to sensor readings that provide feedback on the robot's state. For example, if at least one robot is present in workstation 1, event e_5, associated with transition t_5, enables the movement of a robot from workstation 1 to workstation 3. Event e_3, linked to both transitions t_3 and t_8, is a synchronizing control input that allows the simultaneous evacuation of robots from workstations 3 and 4 back to workstation 1. Note that if two robots are located in the same workstation, a single control event can move only one robot at a time. Several sensors are deployed in the physical system to detect specific robot movements. Without loss of generality, we consider only the forbidden outputs $\mathcal{F}_{\text{out}} = \{F_1, F_2\}$, as defined in Table I. All other authorized outputs are abstracted as ε.

We assume that the CPS has been compromised by an attack, resulting in the loss of knowledge about the current positions of the robots. In this scenario, control input e_4

is considered compromised and is therefore unavailable to the controller. The controller's objective is to compute a synchronizing sequence that safely guides both robots to workstation 1—regarded as a secure location—without any prior knowledge of their positions, without using the compromised event e_4, and without generating the forbidden outputs F_1 or F_2, which could potentially leak sensitive information to the attacker.

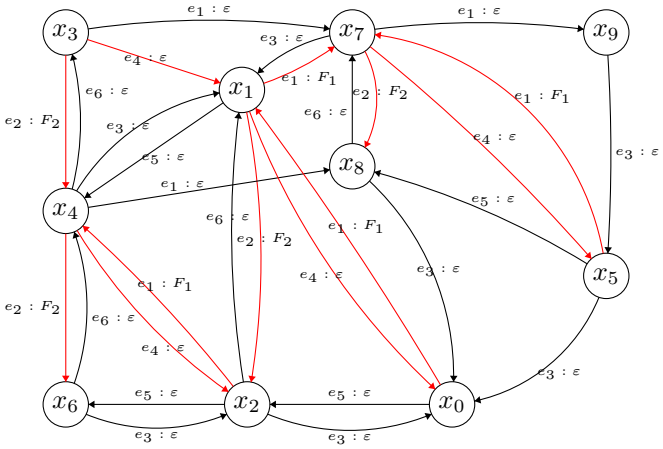

Fig. 7. LFAI of the case study

LFAI state	Associated robots position
x_0	$2P_1$
x_1	P_1P_2
x_2	P_1P_3
x_3	$2P_2$
x_4	P_2P_3
x_5	P_1P_4
x_6	$2P_3$
x_7	P_2P_4
x_8	P_3P_4
x_9	$2P_4$

TABLE II
STATES OF LFAI

Based on the OutSynPN model, the corresponding LFAI is derived and presented in Figure 7. The meaning of each LFAI state is detailed in Table II, where the notation P_i (respectively, $2P_i$) indicates that one robot (respectively, two robots) is located in workstation i. An MLFAI is first derived from the original LFAI by pruning the transition labeled with e_4. Subsequently, an auxiliary graph is constructed from the resulting completely specified MLFAI. This auxiliary graph contains 55 states in total. Algorithm 2 is then applied to compute an SS to target state x_0 under both FIC $\{e_4\}$ and FOC $\{F_1, F_2\}$. The resulting sequence is given by $\bar{\omega} = e_5e_3e_5e_3e_5e_5e_3e_3e_5e_5e_6e_1e_1e_3e_5e_3$.

To illustrate the application of the proposed approach, consider a scenario where both robots are initially located at workstation 2, which corresponds to the CPS being in state $x_3 = 2P_2$. When the computed synchronizing sequence is applied, it leads to the following sequence of robot movements: $x_3 \xrightarrow{e_5:\varepsilon} x_3 \xrightarrow{e_3:\varepsilon} x_3 \xrightarrow{e_5:\varepsilon} x_3 \xrightarrow{e_3:\varepsilon} x_3 \xrightarrow{e_5:\varepsilon} x_3 \xrightarrow{e_3:\varepsilon} x_3 \xrightarrow{e_5:\varepsilon} x_3 \xrightarrow{e_5:\varepsilon} x_3 \xrightarrow{e_6:\varepsilon} x_3 \xrightarrow{e_1:\varepsilon} x_7 \xrightarrow{e_1:\varepsilon} x_9 \xrightarrow{e_3:\varepsilon} x_5 \xrightarrow{e_5:\varepsilon} x_8 \xrightarrow{e_3:\varepsilon} x_0$. Note that only ε is generated as output.

The synchronizing sequence $\bar{\omega}$ guarantees that both robots reach workstation 1, regardless of their positions, while ensuring that the forbidden event e_4 is avoided and no forbidden outputs $\{F_1, F_2\}$ are generated.

VI. CONCLUSIONS

This paper addressed the problem of recovering a cyber-physical system to a known safe state after a loss of state estimation, which may occur due to cyberattacks. The proposed method is based on labeled finite automata with inputs derived from an output synchronized Petri net. It consists of arc pruning method and modified greedy algorithm with depth-first search to compute synchronizing sequences under forbidden input and output constraints. A case study involving mobile robots is presented. Future work includes analyzing the scalability of the approach on larger systems and developing more efficient search strategies, possibly incorporating heuristic-guided techniques. Another direction is the use of adaptive synchronizing sequences that adjust dynamically based on observed outputs.

REFERENCES

[1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer, 2008.

[2] S. Oliveira, A. B. Leal, M. Teixeira, and Y. K. Lopes, "A classification of cybersecurity strategies in the context of discrete event systems," *Annual reviews in Control*, vol. 56, p. 100907, 2023.

[3] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, Aug. 1996.

[4] M. Soucha, "Finite state machine state identification sequences," *Open informatics—computer and information science. Faculty of Electrical Engineering. Department of Cybernetics*, 2014.

[5] M. Pocci, "Test and diagnosis of discrete event systems using Petri nets," Ph.D. dissertation, Aix-Marseille, 2013.

[6] M. Pocci, I. Demongodin, N. Giambiasi, and A. Giua, "Synchronizing sequences on a class of unbounded systems using synchronized Petri nets," *Discrete Event Dynamic Systems*, vol. 26, no. 1, pp. 85–108, Mar. 2016.

[7] C. Wu, I. Demongodin, and A. Giua, "Correction to "Synchronizing sequences on a class of unbounded systems using synchronized Petri nets"," *Discrete Event Dynamic Systems*, vol. 29, no. 4, pp. 521–526, Dec. 2019.

[8] R. David and H. Alla, *Discrete, continuous, and hybrid Petri nets*. Springer, 2010, vol. 1.

[9] S. Sandberg, "1 homing and synchronizing sequences," in *Model-based testing of reactive systems*. Springer, 2005, pp. 5–33.

[10] M. Christou and C. N. Hadjicostis, "Driving a Discrete Event System to a Known State via Minimal Length Adaptive Control Sequences," *IFAC-PapersOnLine*, vol. 58, no. 1, pp. 210–215, 2024.

[11] K. Hamada, R. Ammour, L. Brenner, and I. Demongodin, "Attack Synchronizing Sequence Computation for Output Synchronized Petri Nets with Multiple Deadlocks," in *2023 IEEE International Conference on Networking, Sensing and Control (ICNSC)*. Marseille, France: IEEE, Oct. 2023, pp. 1–6.

[12] R. Ammour, S. Amari, L. Brenner, I. Demongodin, and D. Lefebvre, "Observer design for bounded output synchronized Petri nets," in *2021 European Control Conference (ECC)*. IEEE, 2021, pp. 746–751.

[13] M. Pocci, I. Demongodin, N. Giambiasi, and A. Giua, "Testing experiments on unbounded systems: synchronizing sequences using Petri nets," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 155–161, Jan. 2014.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.