Supervisor Obfuscation Against Actuator Enablement Attack

Yuting Zhu*, Liyong Lin*, Rong Su

Abstract—In this paper, we propose and address the problem of supervisor obfuscation against actuator enablement attack, in a common setting where the actuator attacker can eavesdrop the control commands issued by the supervisor. We propose a method to obfuscate an (insecure) supervisor to make it resilient against actuator enablement attack in such a way that the behavior of the original closed-loop system is preserved. An additional feature of the obfuscated supervisor, if it exists, is that it has exactly the minimum number of states among the set of all the resilient and behavior-preserving supervisors. Our approach involves a simple combination of two basic ideas: 1) a formulation of the problem of computing behavior-preserving supervisors as the problem of computing separating finite state automata under controllability and observability constraints, which can be efficiently tackled by using modern SAT solvers, and 2) the use of a recently proposed technique for the verification of attackability in our setting, with a normality assumption imposed on both the actuator attackers and supervisors.

Index Terms – cyber-physical systems, discrete-event systems, actuator attack, supervisory control

I. INTRODUCTION

Recently, cyber-physical systems have drawn a lot of attention from the supervisory control and formal methods research communities (see, for example, [1], [2], [3], [4], [5], [6], [7]). The supervisory control theory of discrete-event systems[1] [18] has been proposed as a general approach for the synthesis of correct-by-construction supervisors that ensure both safety and progress properties on the closed-loop systems. However, the correctness guarantee is implicitly based upon the assumption that the system has been fully and correctly modelled. In the presence of unmodeled adversarial attackers, the synthesized supervisor cannot guarantee safety any more. Thus, one is then required to synthesize resilient supervisors against adversarial attacks. In view of the importance and prevalence of cyber-physical systems in modern society [8], [9], [10], researchers in supervisory control theory have suggested various frameworks and techniques (see [1], [2], [6], [7]) for the design of mitigation and resilient control mechanisms against adversarial attacks. In this work, we follow the setup of [11] and consider the problem of synthesis of resilient supervisors against actuator enablement attack, with a normality assumption imposed on both the actuator attackers and supervisors.

[1]In this work, we focus on the class of cyber-physical systems that can be modelled using discrete-event systems.

In [11], a formal formulation of the supervisor (augmented with a monitoring mechanism), the actuator enablement attacker, the attacked closed-loop system and resilient supervisor has been provided. The attacker is assumed to observe no more than the supervisor does on the execution of the plant. However, the attacker can eavesdrop the control commands issued by the supervisor. In this common setting, the control commands issued by the supervisor could be used to refine the knowledge of the attacker about the execution of the plant. The attacker can modify each control command on a specified subset of attackable events. The attack principle of the actuator attacker is to remain covert until it can establish a successful attack and lead the attacked closed-loop system into generating certain damaging strings. In [11], a notion of attackability has been identified as a characterizing condition for the existence of a successful actuator enablement attacker, under a normality assumption imposed on the actuator attackers and supervisors; an algorithm for the verification of attackability has also been provided [11]. However, in [11], the problem of synthesis of resilient supervisors against actuator enablement attacks has not been addressed.

Instead of directly synthesizing a supervisor that is resilient against actuator enablement attacks, it is possible to obfuscate an insecure supervisor to make it resilient, while preserving the behavior of the original closed-loop system. This is motivated by the following simple observation. The attacker can observe information from both the execution of the closed-loop system and the control commands issued by the supervisor. For any behavior-preserving supervisor, it cannot influence[2] what the actuator attacker will observe from the execution of the closed-loop system but can partially determine what information the attacker can obtain[3] from the control commands it issued. In order to be secure, a supervisor should leak as few information as possible in its issued control commands to any potential attacker.

The main contributions of this paper are as follows.

1) We propose the problem of synthesis of resilient and behavior-preserving supervisors, i.e., the supervisor obfuscation problem, in the setting where there is a control command eavesdropping actuator enablement attacker. The supervisor that is provided as input to our problem may have been synthesized conforming to different kinds of complicated specifications and by using advanced synthesis techniques, but is insecure against actuator attack. Our approach avoids starting from scratch and developing a new resilient supervisor synthesis algorithm for each type of specification. An application

[2]For any behavior-preserving supervisor, by definition the behavior of the closed-loop system stays the same.

[3]The supervisor is required to be behavior-preserving and thus cannot issue arbitrary control commands.

of this approach is to first synthesize a supervisor that takes care of all the safety specifications, except for the resilience property; the insecure supervisor can then be obfuscated to become resilient, if it exists.

2) The problem of computing behavior-preserving supervisor is formulated as the problem of computing separating finite state automata [12], but with additional constraints such as controllability and observability, which can be efficiently reduced to the boolean satisfiability problem (SAT) and efficiently solved using modern SAT solvers.

3) The algorithmic solution for the problem of computing behavior-preserving supervisors is used as a subroutine for the supervisor obfuscation problem. The obfuscated supervisor computed using our algorithm, if it exists, has exactly the minimum number of states among the set of all the resilient and behavior-preserving supervisors.

The problem of computing behavior-preserving supervisors is essentially a flexible version of the supervisor reduction problem [13], [14], [15], the only differences being that there is no requirement on the state size of the computed supervisor and the control constraint of the computed supervisor does not have to be the same as that of the original supervisor. In this sense, the problem of computing behavior-preserving supervisors is a simple generalization of the supervisor reduction problem[4]. The current implementation of the supervisor reduction procedure is not flexible and well tuned enough to be used as a solution to our problem, compared with the SAT-based approach. Apart from the SAT-solving technique, automaton learning and SMT solving [12], [16] can also be adopted to solve the problem. This work is mainly inspired by the work of [12] on computing minimal separating finite state automata and the reduction technique developed in [12] is adapted for solving the supervisor obfuscation problem.

The paper is organized as follows. Section II is devoted to the general preliminaries; in addition, we recall the attack architecture, system formulation and definition of attackability provided in [11]. Then, in Section III, we propose the problem of synthesis of resilient and behavior-preserving supervisors, i.e., the supervisor obfuscation problem, and then provide an algorithm to solve it. Finally, in Section IV, we conclude the paper and discuss some future research directions.

II. PRELIMINARIES

In this section, we shall provide the basic preliminaries that are necessary to understand our paper. We assume the reader to be familiar with the basic theories of formal languages, finite automata and supervisory control [18], [19]. Some additional notations and terminologies are introduced in the following. After that, we recall the attack architecture, system formulation and the definition of attackability in [11] that are specific to our setting.

Let A and B be any two sets. We write $A \backslash B$ to denote their set-theoretic difference, i.e., $A \backslash B := \{x \in A \mid x \notin B\}$. Let $A \times B$ denote their Cartesian product. We write $|A|$ to

---

[4]However, the plants and supervisors used in this work are non-marking. The technique developed in this work can be easily extended to the marking case, which can be used for solving the standard supervisor reduction problem.

denote the cardinality of A. A partial finite state automaton G over alphabet Σ is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite set of states, $\delta : Q \times \Sigma \longrightarrow Q$ the partial transition function[5], $q_0 \in Q$ the initial state and Q_m the set of marked states. G is said to be a complete finite state automaton if $\delta : Q \times \Sigma \longrightarrow Q$ is a total function. Let $L(G)$ (respectively, $L_m(G)$) denote the closed-behavior and the marked-behavior of G, respectively. When $Q_m = Q$, we also write $G = (Q, \Sigma, \delta, q_0)$ for simplicity, in which case we have $L_m(G) = L(G)$. In this work, whenever we talk about a plant, we mean a partial finite state automaton $G = (Q, \Sigma, \delta, q_0)$. G is said to be n-bounded if $|Q| \leq n$. For two finite state automata $G_1 = (X_1, \Sigma_1, \delta_1, x_{1,0}), G_2 = (X_2, \Sigma_2, \delta_2, x_{2,0})$, we write $G := G_1 \| G_2$ to denote their synchronous product. Then, $G = (X := X_1 \times X_2, \Sigma := \Sigma_1 \cup \Sigma_2, \delta = \delta_1 \times \delta_2, x_0 := (x_{1,0}, x_{2,0}))$ where the (partial) transition map δ is defined as follows. $(\forall x = (x_1, x_2) \in X)(\forall \sigma \in \Sigma)\delta(x, \sigma)$

$$:= \begin{cases} (\delta_1(x_1, \sigma), x_2), & \text{if } \sigma \in \Sigma_1 \backslash \Sigma_2 \\ (x_1, \delta_2(x_2, \sigma)), & \text{if } \sigma \in \Sigma_2 \backslash \Sigma_1 \\ (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \end{cases}$$

Propositional formulas ϕ [12] [17] are constructed from (Boolean) variables by using logical connectives ($\wedge, \vee, \rightarrow, \neg$). The truth value of a propositional formula is determined by the variables' truth value. A literal is a variable or its negation. A clause is a disjunction $l_1 \vee \ldots \vee l_n$ of literals. A formula in conjunctive normal form (CNF) is conjunction of clauses. Each propositional formula can be converted into an equivalent formula that is in conjunctive normal form. Let $Var(\phi)$ denote the set of variables (Boolean variables $x_0, \ldots, x_n$) occurring in ϕ; A model of ϕ is a mapping $\mathbf{M}: Var(\phi) \rightarrow \{0, 1\}$ (0 representing False, 1 representing True) such that ϕ is evaluated to be True if all variables x_i in ϕ are substituted by $\mathbf{M}(x_i)$.

A control constraint $\mathcal{A}$ over Σ is a tuple (Σ_c, Σ_o), where $\Sigma_c \subseteq \Sigma$ denotes the subset of controllable events and $\Sigma_o \subseteq \Sigma$ denotes the subset of observable events. Let $\Sigma_{uo} := \Sigma \backslash \Sigma_o$ denote the subset of unobservable events and $\Sigma_{uc} := \Sigma \backslash \Sigma_c$ the subset of uncontrollable events. For each sub-alphabet $\Sigma' \subseteq \Sigma$, the natural projection $P : \Sigma^* \rightarrow \Sigma'^*$ is defined and naturally extended to a mapping between languages [18].

Attack Architecture: We consider the architecture for actuator attack shown in Fig. 2. The plant G is under the control of a partially observing supervisor V w.r.t. (Σ_c, Σ_o). In addition, there exists an actuator attacker A that can observe the execution of the plant and eavesdrop the control commands issued by the supervisor. We assume that each time when the supervisor observes an event (and makes a state transition), it issues a new control command that can be intercepted by the attacker. It can modify the control command γ issued by the supervisor on a designated subset of attackable events $\Sigma_{c,A} \subseteq \Sigma_c$, each time when it intercepts a control command. The plant G follows the modified control command γ' instead of γ. The attacker can observe events in $\Sigma_{o,A} \subseteq \Sigma_o$ in the execution of the plant. $(\Sigma_{c,A}, \Sigma_{o,A})$ is said to be an attack constraint over Σ. We assume $\Sigma_c \subseteq \Sigma_o$

---

[5]As usual, δ can also be viewed as a relation $\delta \subseteq Q \times \Sigma \times Q$.

and $\Sigma_{c,A} \subseteq \Sigma_{o,A}$. That is, both the actuator attackers and supervisors are assumed to be normal [11]. The supervisor is augmented with a monitoring mechanism that monitors the execution of the closed-loop system (under attack). The supervisor has a mechanism for halting the execution of the closed-loop system after discovering an actuator attack. The goal of the attacker is to drive the attacked closed-loop system into executing specific damaging strings outside the controlled behavior, without risking itself being discovered by the supervisor before it causes damages.

Figure 1: The architecture for actuator attack

System Formulation:

1) Supervisor: A supervisor[6] V on G w.r.t. (Σ_c, Σ_o) is effectively a map $V : P_o(L(V/G)) \to \Gamma$, where $\Gamma = \{\gamma \subseteq \Sigma \mid \Sigma_{uc} \subseteq \gamma\}$ and $L(V/G)$ is the closed behavior of the closed-loop system V/G [18]. Once any string $w \notin P_o(L(V/G))$ is observed by the supervisor, the closed-loop system is immediately halted, since an attack has been discovered by the supervisor. For any $w \in P_o(L(V/G))$, we have $V(w) \supseteq \Sigma_{uc} \supseteq \Sigma_{uo}$ by the definition of Γ and since $\Sigma_c \subseteq \Sigma_o$. A supervisor V (on G) w.r.t. control constraint (Σ_c, Σ_o) is realized by a partial finite state automaton $S = (X, \Sigma, \xi, x_0)$ that satisfies the controllability and observability constraints [23]:

- **C)** (controllability) for any state $x \in X$ and any uncontrollable event $\sigma \in \Sigma_{uc}$, $\xi(x, \sigma)!$,
- **O)** (observability) for any state $x \in X$ and any unobservable event $\sigma \in \Sigma_{uo}$, $\xi(x, \sigma)!$ implies $\xi(x, \sigma) = x$,

where we have $L(V/G) = L(S \parallel G)$ and, for any $s \in L(G)$ such that $P_o(s) \in P_o(L(V/G))$, $V(P_o(s)) = \{\sigma \in \Sigma \mid \xi(\xi(x_0, s_0), \sigma)!, s_0 \in L(S) \cap P_o^{-1}P_o(s)\}$. For a normal supervisor S w.r.t. control constraint (Σ_c, Σ_o), the observability constraint is then reduced to: for any state $x \in X$ and any unobservable event $\sigma \in \Sigma_{uo}$, $\xi(x, \sigma) = x$.

2) Damage-inflicting set: The goal of the actuator attacker is to drive the attacked closed-loop system into executing certain damaging strings. Let $L_{dmg} \subseteq L(G)$ be some regular language over Σ that denotes a so-called "damage-inflicting

---

set", where each string is a damaging string. We require that[7] $L_{dmg} \cap L(V/G) = \varnothing$, that is, no string in $L(V/G)$ could be damaging. Then, we have that $L_{dmg} \subseteq L(G) - L(V/G)$. We remark that L_{dmg} is a general device for specifying what are damaging strings and may not correspond to the specification that is used for synthesizing V. L_{dmg} is given by the damage automaton $H = (Z, \Sigma, \eta, z_0, Z_m)$, i.e., $L_{dmg} = L_m(H)$. We require H to be a complete automaton and thus $L(H) = \Sigma^*$.

3) Actuator Attacker: The attacker's observation sequence is simply a string in $((\Sigma_{o,A} \cup \{\epsilon\}) \times \Gamma)^*$. The attacker's observation sequence solely depends on the supervisors observation sequence $P_o(s) \in P_o(L(V/G))$, including the observation on the execution of the plant and the control commands issued by the supervisor. $\hat{P}_{o,A}^V : P_o(L(V/G)) \to ((\Sigma_{o,A} \cup \{\epsilon\}) \times \Gamma)^*$ is a function that maps supervisor's observation sequences to attacker's observation sequences, where, for any $w = \sigma_1\sigma_2 \ldots \sigma_n \in P_o(L(V/G))$, $\hat{P}_{o,A}^V(w)$ is defined to be

$$(P_{o,A}(\sigma_1), V(\sigma_1))(P_{o,A}(\sigma_2), V(\sigma_1\sigma_2)) \ldots$$
$$(P_{o,A}(\sigma_n), V(\sigma_1\sigma_2 \ldots \sigma_n)) \in ((\Sigma_{o,A} \cup \{\epsilon\}) \times \Gamma)^*.$$

An attacker on (G, V) w.r.t. attack constraint $(\Sigma_{c,A}, \Sigma_{o,A})$ is a function $A : \hat{P}_{o,A}^V(P_o(L(V/G))) \to \Delta\Gamma$, where $\Delta\Gamma = 2^{\Sigma_{c,A}}$. Here, $\Delta\Gamma$ denotes the set of all the possible attack decisions that can be made by the attacker on the set $\Sigma_{c,A}$ of attackable events. Intuitively, each $\Delta\gamma \in \Delta\Gamma$ denotes the set of enabled attackable events that are determined by the attacker. For any $P_o(s) \in P_o(L(V/G))$, the attacker determines the set $\Delta\gamma = A(\hat{P}_{o,A}^V(P_o(s)))$ of attackable events to be enabled based on its observation $\hat{P}_{o,A}^V(P_o(s))$.

Now, given the supervisor V and the actuator attacker A, we could lump them together into an equivalent (attacked) supervisor $V_A : P_o(L(V/G)) \to \Gamma$ such that, for any string $w \in P_o(L(V/G))$ observed by the attacked supervisor, the control command issued by V_A is $V_A(w) = (V(w) - \Sigma_{c,A}) \cup A(\hat{P}_{o,A}^V(w))$. The plant G follows the control command issued by the attacked supervisor V_A and the attacked closed-loop system is denoted by V_A/G. The definition of the (attacked) closed-behavior $L(V_A/G)$ of V_A/G is inductively defined as follows.

1) $\epsilon \in L(V_A/G)$,
2) if $s \in L(V_A/G)$, $P_o(s) \in P_o(L(V/G))$, $\sigma \in V_A(P_o(s))$ and $s\sigma \in L(G)$, then $s\sigma \in L(V_A/G)$,
3) no other strings belong to $L(V_A/G)$.

The goal of the actuator attacker is formulated as follows: $L(V_A/G) \cap L_{dmg} \neq \varnothing$, i.e., at least some damaging string can be generated before the execution of the closed-loop system is halted. If the goal is achieved, then we say A is a successful actuator attacker on (G, V) w.r.t. $(\Sigma_{c,A}, \Sigma_{o,A})$ and L_{dmg}. Under the normality assumption, without loss of generality, we can assume A is an enablement attacker [11], i.e., $A(\hat{P}_{o,A}^V(w)) \supseteq V(w) \cap \Sigma_{c,A}$, for any $w \in P_o(L(V/G))$.

Attackability: We now recall the definition of attackability, which plays an important role in characterizing the existence of a successful actuator attacker.

Definition 1. (G, V) *is said to be attackable w.r.t.* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and* L_{dmg} *if there exists a string* $s \in L(V/G)$ *and an attackable event* $\sigma \in \Sigma_{c,A}$, *such that*

1) $s\sigma \in L_{dmg}$
2) *for any* $s' \in L(V/G)$, $\hat{P}_{o,A}^V(P_o(s)) = \hat{P}_{o,A}^V(P_o(s'))$ *and* $s'\sigma \in L(G)$ *together implies* $s'\sigma \in L_{dmg}$.

We recall the following result [11].

Theorem 1. *There exists a successful actuator attacker on* (G, V) *w.r.t.* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and* L_{dmg} *iff* (G, V) *is attackable w.r.t* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and* L_{dmg}.

An algorithm for (non-)attackability verification is available in [11].

III. SUPERVISOR OBFUSCATION PROBLEM

In this section, we shall propose and address the problem of supervisor obfuscation, which is an approach for the synthesis of resilient supervisors against actuator attackers, while preserving desired behavior of the original closed-loop system.

A. Problem Formulation

A supervisor V' is said to be an *obfuscation* of V (against actuator attacks) on G w.r.t. $(\Sigma_{c,A}, \Sigma_{o,A})$ and L_{dmg} if the following two conditions are satisfied: 1) there is no successful actuator attacker on (G, V') w.r.t. $(\Sigma_{c,A}, \Sigma_{o,A})$ and L_{dmg}, and 2) $L(V/G) = L(V'/G)$. In general, we do not require V' to be over the same control constraint as V and, in practice, V is insecure (for us to be interested in the supervisor obfuscation problem). It is worth noting that $L(V'/G) = L(V/G)$ does not imply the equi-attackability of (G, V) and (G, V') because in general $V \neq V'$ and then $\hat{P}_{o,A}^V \neq \hat{P}_{o,A}^{V'}$ (c.f. Definition 1). This shall be illustrated by the next example.

Example 1. *We shall provide a simple supervisor obfuscation example below. Let us consider the plant G and the supervisor S shown in Fig. 2. The colored state 8 is the only bad state to avoid.* $\Sigma = \{a, b, c, d, a'\}, \Sigma_o = \{a, c, d, a'\}, \Sigma_c = \Sigma_o, \Sigma_{c,A} = \{a'\}, \Sigma_{o,A} = \{c, a'\}$. *We can check that S has ensured safety, without the presence of an attacker. The damage automaton H (with the dump state and the corresponding transitions being omitted) is also shown in Fig. 2, where state 8 is the only marked state. However, (G, S) can be attacked w.r.t.* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and H. When the plant generates string $\omega = acd$, the supervisor knows the current state is exactly state 6. However, event d is unobservable to the attacker; before the execution of event d. the attacker will firstly estimate the current state should be state 5 or state 6. Then, with the eavesdropped control command information from the supervisor after d is executed, it can know that event d has occurred because $V(ac) = \{a, b, d\}$ in supervisor state 3 while $V(acd) = \{b\}$ in supervisor state 4. Then, it knows the plant is in state 6 and thus an attack on event a' can be established.*

We have also shown another supervisor S' in Figure 2, which also has ensured the safety of the closed-loop system. We can easily check that $L(S\|G) = L(S'\|G)$ and thus S' is behavior-preserving. To prevent the attacker from detecting

the occurrence of event d, compared with S, the supervisor S' self-loops event d at state 3 instead and thus $V'(ac) = V'(acd) = \{a, b, d\}$. Effectively, the attacker cannot determine whether it is event d or a that occurs, even if it eavesdrops a control command and knows that a transition has occurred. Therefore, the attacker will not take the risk to carry out the attack and (G, S') is not attackable w.r.t. $(\Sigma_{c,A}, \Sigma_{o,A})$ and H.

Figure 2: The plant G, the damage automaton H, the original supervisor S and the obfuscated supervisor S'

In the rest, we implicitly fix the alphabet Σ and use automata representations as problem inputs. We are now ready to state the main problems.

Problem 1. *Given a plant G, a supervisor S, a control constraint $\mathcal{A} = (\Sigma_c, \Sigma_o)$, an attack constraint $(\Sigma_{c,A}, \Sigma_{o,A})$, a damage automaton H and a positive integer n, does there exist an n-bounded supervisor S' over $\mathcal{A}$ such that 1) $L(S' \| G) = L(S \| G)$ and 2) (G, S') is not attackable w.r.t.* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and H.*

Problem 1 asks whether there exists a resilient and behavior-preserving supervisor over $\mathcal{A}$ of state size no more than n. The optimization version of Problem 1, which is given as Problem 2, is the supervisor obfuscation problem that we intend to solve.

Problem 2. *Given a plant G, a supervisor S, a control constraint $\mathcal{A} = (\Sigma_c, \Sigma_o)$, an attack constraint $(\Sigma_{c,A}, \Sigma_{o,A})$, a damage automaton H, compute a supervisor S' over $\mathcal{A}$, if it exists, such that 1) $L(S' \| G) = L(S \| G)$, 2) (G, S') is not attackable w.r.t.* $(\Sigma_{c,A}, \Sigma_{o,A})$ *and H, and 3) S' has the minimum number of states among the set of all the supervisors over $\mathcal{A}$ that satisfy both 1) and 2)).*

We shall develop an algorithmic solution in Section III-B that can be used for solving both Problem 1 and Problem 2. We have the following remark.

Remark 1. *Since S and G are given, we can compute $K = L(S\|G)$. Thus, we do not need to know the control constraint of the original supervisor S and the computed supervisor S' can be over a different control constraint than that of S.*

B. Algorithm for Supervisor Obfuscation

Our proposed solution for solving the supervisor obfuscation problem is given in Algorithm 1. In particular, Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$ is used to generate the stack I of all the supervisors that are behavior-preserving with respect to S and have state size n. This procedure can be completed efficiently with the help of an ALL-SAT solver [20], [21]. Algorithm **NA**(G, S, H) is used to verify the non-attackability of (G, S') w.r.t. $(\Sigma_{c,A}, \Sigma_{o,A})$ and H. The details of Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$ and Algorithm **NA**(G, S', H) will be explained in the following subsections.

The algorithm for supervisor obfuscation starts by searching for behavior-preserving supervisors of state size $n = 1$. If there is no such behavior-preserving supervisor, then we increment state size n and redo the search. If there is a behavior-preserving supervisor of state size n, then we store the set of all behavior-preserving supervisors of state size n in stack I, obtained by using Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$. For each behavior-preserving supervisor in the stack, we check the non-attackability of the closed-loop system using Algorithm **NA**(G, S, H). If there exists a behavior-preserving supervisor in the stack so that the closed-loop system is non-attackable, we have found a resilient and behavior-preserving supervisor, which is guaranteed to be of the minimum number of states. If the closed-loop system associated with each behavior-preserving supervisor in the stack is attackable, then we increment n and redo the search. To prevent the algorithm from searching infinitely (by increasing n indefinitely), we can impose a bound on the largest n that we would like to search. Typically, we choose the largest n to be $|X|$ (that is, we would like to synthesize a resilient and behavior-preserving supervisor S' of smaller state size than that of S) or some small multiples of $|X|$. We remark that the bisection method can be used to improve the performance of the searching process.

The formula $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$ used in Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$ states the existence of an n-bounded supervisor over $\mathcal{A}$ that is behavior-preserving w.r.t. S over G. A satisfying assignment $\mathbf{M}$ of $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$ can be easily converted to an n-bounded behavior-preserving supervisor $S'^{\mathbf{M}}$ on G over $\mathcal{A}$. Only when the set of reachable states of $S'^{\mathbf{M}}$ is of cardinality n, we can then add $S'^{\mathbf{M}}$ to the stack I, in Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$, to ensure that I only has behavior-preserving supervisors of state size exactly n. This avoids repetition of non-attackability verification carried out in Step 7 of Algorithm 1, as supervisors found by solving $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$ are only guaranteed to be n-bounded.

Computing Behavior-Preserving Supervisors: We now show that the problem of computing behavior-preserving supervisors, which is given below, is equivalent to the problem of computing separating finite state automata under controllability and observability constraints. This shall allow us to borrow the technique developed for computing minimal separating

---

Algorithm 1 Algorithm for supervisor obfuscation

Input Plant $G = (Q, \Sigma, \delta, q_0)$, supervisor $S = (X, \Sigma, \xi, x_0)$, control constraint $\mathcal{A}$, attack constraint $(\Sigma_{c,A}, \Sigma_{o,A})$, damage automaton $H = (Z, \Sigma, \eta, z_0, Z_m)$.
Output Supervisor $S' = (X', \Sigma, \xi', x_0')$

1: $n := 1$
2: Compute the stack of behavior-preserving supervisors $I = $ **SUPBP**$(G, S, \mathcal{A}, n)$ of state size n
3: **If** $I = \varnothing$
4:     $n := n + 1$, **goto** step 2
5: **While** $I \neq \varnothing$
6:     Let $S' = I.pop()$
7:     Compute **NA**(G, S', H)
8:     **If** **NA**$(G, S', H) = $ TRUE
9:        **return** S'
10: $n := n + 1$, **goto** step 2

Algorithm 2 Algorithm **SUPBP**$(G, S, \mathcal{A}, n)$

Input: Plant $G = (Q, \Sigma, \delta, q_0)$, control constraint $\mathcal{A}$, supervisor $S = (X, \Sigma, \xi, x_0)$ and a positive integer n
Output: Stack I of behavior-preserving supervisors of state size n over $\mathcal{A}$.

1: Construct the boolean formula $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$
2: Let $I = \varnothing$
3: **If** $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$ is satisfiable
4:    **for** each satisfying assignment $\mathbf{M}$ of $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$
5:      Construct the supervisor $S'^{\mathbf{M}}$ that corresponds to $\mathbf{M}$
6:      If $S'^{\mathbf{M}}$ is of state size n
7:        Let $I := I.push(S'^{\mathbf{M}})$
8: **return** I

---

finite state automata to solve our problem. Essentially, for any given plant G, supervisor S and control constraint $\mathcal{A}$, we shall construct boolean formulas $\phi_n^{\overline{G\downarrow S}, \mathcal{A}}$ that state the existence of an n-bounded supervisor over $\mathcal{A}$ that is behavior-preserving w.r.t. S over G.

Problem 3 (Existence of n-Bounded Behavior-Preserving Supervisors)**.** *Given a plant G, a control constraint $\mathcal{A} = $*

Algorithm 3 Algorithm **NA**(G, S', H)

Input: Plant $G = (Q, \Sigma, \delta, q_0)$, control constraint $\mathcal{A}$, supervisor $S' = (X', \Sigma, \xi', x_o')$, damage automaton $H = (Z, \Sigma, \eta, z_0, Z_m)$
Output: TRUE or FALSE

1: Compute the annotated supervisor S'^A of S'
2: Compute the general synchronous product $GP(G, S'^A, H)$ of G, S'^A and H
3: Compute the automaton $SUB(GP(G, S'^A, H))$ with a labeling function Lf
4: **If** for every reachable state y in $SUB(GP(G, S'^A, H))$ it holds that $Lf(y) = \varnothing$
5:     **return** TRUE
6: **else**
7:     **return** FALSE

(Σ_c, Σ_o) *and a supervisor S, determine the existence of an n-bounded supervisor S' over $\mathcal{A}$ such that $L(S\|G) = L(S'\|G)$.*

We shall now introduce the well-known notion of a separating finite state automaton [12], which works in the context of complete finite state automata.

Definition 2 (Separating Finite State Automaton). *Given any (complete) finite state automaton C and any two languages L_1, L_2, if $L_1 \subseteq L_m(C)$ and $L_m(C) \cap L_2 = \varnothing$, then C is said to be a separating finite state automaton for the pair (L_1, L_2).*

Intuitively, C is a witness of the disjointness of L_1 and L_2. We make a simple observation in Lemma 1, which relates the notion of a behavior-perserving supervisor with the notion of a separating finite state automaton. In order to relate these two notions, we need to convert the plant and the supervisor[8], to a complete finite state automaton by adding a dump state, so that a partial finite state automaton $P = (W, \Sigma, \pi, w_0)$ becomes a complete finite state automaton $\overline{P} = (W \cup \{w_d\}, \Sigma, \overline{\pi}, w_0, W)$, where the subscript d is always used with dump state and $\overline{\pi} =$

$$\pi \cup (\{w_d\} \times \Sigma \times \{w_d\}) \cup \{(w, \sigma, w_d) \mid \pi(w, \sigma) \text{ is} \\ \text{undefined}, w \in W, \sigma \in \Sigma\}$$

After the above completion process, we have $L(P) = L_m(\overline{P})$. It is also straightforward to recover P from $\overline{P}$, by removing the dump state (and the corresponding transitions). In the rest, we shall work on $\overline{G}$, $\overline{S}$ and $\overline{S'}$ instead. In particular, $\overline{G}$ and $\overline{S}$ can be obtained from G and S by the above completion process; S' is the behavior-preserving supervisor that we would like to compute and we only need to obtain $\overline{S'}$ first and then remove the unique dump state of $\overline{S'}$ to recover S'. Then, our goal is to compute $\overline{S'}$ in the following.

Based on the above discussion, we need to ensure $L_m(\overline{G}) \cap L_m(\overline{S'}) = L_m(\overline{G}) \cap L_m(\overline{S})$, which is equivalent to $L(G) \cap L(S') = L(G) \cap L(S)$. Now, we are ready to state Lemma 1.

Lemma 1. *Let $\overline{G}$, $\overline{S}$ and $\overline{S'}$ be any given complete finite state automata. $L_m(\overline{G}) \cap L_m(\overline{S'}) = L_m(\overline{G}) \cap L_m(\overline{S})$ if and only if*

1) $L_m(\overline{G}) \cap L_m(\overline{S}) \subseteq L_m(\overline{S'})$
2) $L_m(\overline{S'}) \cap (L_m(\overline{G}) \backslash L_m(\overline{S})) = \varnothing$

According to the above result and discussion, to solve Problem 3 we shall compute an $(n+1)$-bounded $\overline{S'}$ (since S' is n-bounded) that satisfies the following conditions:

1) $\overline{S'}$ separates $(L_m(\overline{G}) \cap L_m(\overline{S}), L_m(\overline{G}) \backslash L_m(\overline{S}))$;
2) The S' part of $\overline{S'}$ satisfies constraints (**C**) and (**O**).

We recall that $G = (Q, \Sigma, \delta, q_0)$ and $S = (X, \Sigma, \xi, x_0)$. It is straightforward to encode in a "generalized" complete finite state automaton the two languages $L_m(\overline{G}) \cap L_m(\overline{S})$ and $L_m(\overline{G}) \backslash L_m(\overline{S})$, by using two different kinds of markings. Let $\overline{G \downarrow S} = (Y, \Sigma, \rho, y_0, Y_A, Y_B)$ denote such a generalized (complete) finite state automaton with two kinds of marking states Y_A and Y_B. The set $Y_A \subseteq Y$ of marking states is used to recognize $L_m(\overline{G}) \cap L_m(\overline{S})$ and the set $Y_B \subseteq Y$ of marking states is used to recognize $L_m(\overline{G}) \backslash L_m(\overline{S})$. $\overline{G \downarrow S}$

---

[8]We assume the non-degenerate and practically relevant case where both supervisors and plants are strictly partial finite state automata. The degenerate case can also be treated in a similar fashion.

can be obtained by computing the synchronous product of $\overline{G}$ and $\overline{S}$, with $Y_A = Q \times X$ and $Y_B = Q \times \{x_d\}$.

We adopt the technique of [12] and provide a polynomial-time reduction from Problem 3 to the SAT problem; it then follows that a SAT solver could be deployed to solve Problem 3. In the high level, the idea of the reduction proceeds as follows: for any given instance of Problem 3 with input $\overline{G}$, $\overline{S}$ and control constraint $\mathcal{A}$, we produce a propositional formula $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ such that $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ is satisfiable if and only if there exists an $(n+1)$-bounded $\overline{S'}$ such that $\overline{S'}$ separates $(L_m(\overline{G}) \cap L_m(\overline{S}), L_m(\overline{G}) \backslash L_m(\overline{S}))$. Moreover, each model of the formula $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ can be directly translated to an $(n+1)$-bounded $\overline{S'}$ (correspondingly, an n-bounded S') that solves the given instance.

Let $S' = (X', \Sigma, \xi', x_0')$ be an n-bounded finite state supervisor over $\mathcal{A}$, where $X' := \{x_0', x_1', \ldots, x_{n-1}'\}$ consists of n states, $x_0' \in X'$ is the initial state; the partial transition function $\xi' : X' \times \Sigma \longrightarrow X'$ is the only parameter that needs to be determined to ensure that S' is a solution of the given instance, if a solution exists. Then, we know that $\overline{S'}$ is given by the 5-tuple $(\{x_0', x_1', \ldots, x_{n-1}', x_d'\}, \Sigma, \overline{\xi'}, x_0', \{x_0', x_1', \ldots, x_{n-1}'\})$ and we need to determine $\overline{\xi'}$. There are constraints on $\overline{\xi'}$ that can be used to reduce the search space of $\overline{S'}$. First of all, $\overline{\xi'}(x_d', \sigma) = x_d'$ for any $\sigma \in \Sigma$. Secondly, our setting assumes normality on the supervisors. For a normal supervisor S' w.r.t. control constraint (Σ_c, Σ_o), the observability constraint is then reduced to: for any state $x' \in X'$ and any unobservable event $\sigma \in \Sigma_{uo}$, $\xi'(x', \sigma) = x'$. This translates to: for any state $x' \in X'$ and any unobservable event $\sigma \in \Sigma_{uo}$, $\overline{\xi'}(x', \sigma) = x'$. For convenience, we let $x_n' = x_d'$. We introduce the boolean variables $t_{x_i', \sigma, x_j'}$, where $x_i', x_j' \in X' \cup \{x_n'\}$, and $\sigma \in \Sigma$, in the encoding of $\overline{S'}$ with the interpretation that $t_{x_i', \sigma, x_j'}$ is true if and only if $\overline{\xi'}(x_i', \sigma) = x_j'$.

We encode the fact that $\overline{\xi'}$ is a transition function using the following constraints.

1) $\neg t_{x_i', \sigma, x_j'} \vee \neg t_{x_i', \sigma, x_k'}$ for each $i \in [0, n-1]$, each $\sigma \in \Sigma_o$ and each $j \neq k \in [0, n]$
2) $\bigvee_{j \in [0,n]} t_{x_i', \sigma, x_j'}$ for each $i \in [0, n-1]$ and each $\sigma \in \Sigma_o$

We remark that Constraints (1) are imposed to ensure that $\overline{\xi'}$ is a partial function, and Constraints (2) are imposed to ensure that $\overline{\xi'}$ is total. Together, they will ensure that $\overline{\xi'}$ is a transition function and thus $\overline{S'}$ is a complete finite state automaton. Then,

$$\phi_n^{fsa} = \bigwedge_{i \in [0,n-1], \sigma \in \Sigma_o, j \neq k \in [0,n]} (\neg t_{x_i', \sigma, x_j'} \vee \neg t_{x_i', \sigma, x_k'}) \wedge \\ \bigwedge_{i \in [0,n-1], \sigma \in \Sigma_o} (\bigvee_{j \in [0,n]} t_{x_i', \sigma, x_j'})$$

shall denote the resultant formula after combining Constraints (1) and (2).

With the above constraints, we can now encode the fact that S' is a finite state supervisor over $\mathcal{A} = (\Sigma_c, \Sigma_o)$ using the following extra constraints.

3) $\bigvee_{j \in [0,n-1]} t_{x_i', \sigma, x_j'}$ for each $i \in [0, n-1]$ and each $\sigma \in \Sigma_{uc} - \Sigma_{uo}$

In particular, Constraint (3) is imposed to ensure controllability (**C**). We do not need to care about unobservable events $\sigma \in \Sigma_{uo}$, which surely lead to self-loops, and we do not need to care about observability, which has been taken care

of. We shall note that the range of the index j in (3), does not contain n, since x'_n is the dump state and thus not relevant for controllability **(C)**. Then,

$$\phi_n^{con} = \bigwedge_{i \in [0,n-1], \sigma \in \Sigma_{uc} - \Sigma_{uo}} (\bigvee_{j \in [0,n-1]} t_{x'_i, \sigma, x'_j})$$

shall denote the resultant formula ensuring controllability.

The above constraints only guarantee that S' is a finite state supervisor over $\mathcal{A}$, and we still need to encode the fact that $\overline{S'}$ is a separating finite state automaton for $(L_m(\overline{G}) \cap L_m(\overline{S}), L_m(\overline{G}) \backslash L_m(\overline{S}))$. That is, we need to encode the fact that $L_m(\overline{G}) \cap L_m(\overline{S}) \subseteq L_m(\overline{S'})$, $L_m(\overline{S'}) \cap (L_m(\overline{G}) \backslash L_m(\overline{S})) = \varnothing$. We recall that

$$\overline{G \downarrow S} = (Y, \Sigma, \rho, y_0, Y_A, Y_B),$$

where Y_A, Y_B are used to recognize the two different kinds of languages $L_m(\overline{G}) \cap L_m(\overline{S})$ and $L_m(\overline{G}) \backslash L_m(\overline{S})$, respectively. The remaining constraints are formulated based on $\overline{S'}$ and $\overline{G \downarrow S}$.

Language inclusion and emptiness of language intersection can be checked by using the synchronous product operation, we only need to track the reachable states in the synchronous product of $\overline{S'}$ and $\overline{G \downarrow S}$. We now introduce, as in [12], auxiliary Boolean variables $r_{x', y}$, where $x' \in X' \cup \{x'_n\}$ and $y \in Y$, with the interpretation that if state (x', y) is reachable from the initial state (x'_0, y_0) in the synchronization of $\overline{S'}$ and $\overline{G \downarrow S}$, then $r_{x', y}$ is True. We have the following constraints.

5) $r_{x'_0, y_0}$
6) $r_{x'_i, y_1} \wedge t_{x'_i, \sigma, x'_j} \implies r_{x'_j, y_2}$, for each $i, j \in [0, n]$, each $y_1 \in Y$, each $\sigma \in \Sigma$ and for each $y_2 \in Y$ such that $y_2 = \rho(y_1, \sigma)$
7) $\neg r_{x'_n, y}$, for each $y \in Y_A$
8) $\bigwedge_{i \in [0, n-1]} \neg r_{x'_i, y}$, for each $y \in Y_B$

In particular, Constraints (5) and (6) are used to propogate the constraints on $r_{x', y}$, based on the synchronous product construction and the inductive definition of reachability. Based on Constraints (5) and (6), Constraints (7) are used to ensure $L_m(\overline{G}) \cap L_m(\overline{S}) \subseteq L_m(\overline{S'})$; Constraints (8) are used to ensure $L_m(\overline{S'}) \cap (L_m(\overline{G}) \backslash L_m(\overline{S})) = \varnothing$.

Let the conjunction of Constraints (5), (6), (7) and (8) be denoted by formula ϕ_n^{sep}, which enforces the separating finite state automaton constraint. We have

$$\phi_n^{sep} = r_{x'_0, y_0} \wedge \bigwedge_{i, j \in [0,n], y_1 \in Y, \sigma \in \Sigma, y_2 = \rho(y_1, \sigma)} (\neg r_{x'_i, y_1} \vee \neg t_{x'_i, \sigma, x'_j} \vee r_{x'_j, y_2}) \wedge \bigwedge_{y \in Y_A} \neg r_{x'_n, y} \wedge \bigwedge_{y \in Y_B, i \in [0, n-1]} \neg r_{x'_i, y},$$

which has been converted to CNF.

Let $\phi_n^{\overline{G \downarrow S}, \mathcal{A}} := \phi_n^{dfa} \wedge \phi_n^{con} \wedge \phi_n^{sep}$. It is clear that $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ is in CNF. We note that Constraints (3) implies Constraints (2) on the part where $\sigma \in \Sigma_o \cap \Sigma_{uc}$, making Constraint (2) partly redundant. By solving $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ using a SAT solver [17], we can get $\overline{S'}$ by interpreting the Boolean variables $t_{x'_i, \sigma, x'_j}$ on $\overline{\xi'}$, which can be used to recover S'. We have the following result.

Theorem 2. *The formula $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ is correct. That is, there exists an n-bounded finite state supervisor S' that solves the instance with plant G, supervisor S and control constraint $\mathcal{A}$ if and only if $\phi_n^{\overline{G \downarrow S}, \mathcal{A}}$ is satisfiable.*

After obtaining S', we shall perform the non-attackability verification. This is explained in the next subsection (see [11] for more details).

Attackability Verification: We here recall the steps used in the algorithm for verifying non-attackability, i.e., Algorithm 3.

1. Annotation of the Supervisor

Given the supervisor $S = (X, \Sigma, \zeta, x_0)$, we produce the annotated supervisor

$$S^A = (X, \Sigma_o \times \Gamma \cup \Sigma_{uo}, \zeta^A, x_0),$$

where $\zeta^A : X \times (\Sigma_o \times \Gamma \cup \Sigma_{uo}) \longrightarrow X$ is the partial transition function, which is defined as follows:

1) For any $x, x' \in X$, $\sigma \in \Sigma_o$, $\gamma \in \Gamma$, $\zeta_A(x, (\sigma, \gamma)) = x'$ iff $\zeta(x, \sigma) = x'$ and $\gamma = \{\sigma \in \Sigma \mid \zeta(x', \sigma)!\}$.
2) For any $x, x' \in X$, $\sigma \in \Sigma_{uo}$, $\zeta_A(x, \sigma) = x'$ iff $\zeta(x, \sigma) = x'$.

2. Generalized Synchronous Product:

Given the plant $G = (Q, \Sigma, \delta, q_0)$, the annotated supervisor $S^A = (X, \Sigma_o \times \Gamma \cup \Sigma_{uo}, \zeta^A, x_0)$ and the damage automaton $H = (Z, \Sigma, \eta, z_0)$, the generalized synchronous product $GP(G, S^A, H)$ of G, S^A and H is given by

$$(Q \times X \times Z \cup \{\bot, \top\}, \Sigma^{GP}, \delta^{GP}, (q_0, x_0, z_0)),$$

Where, the state $\bot$ indicates a failed attack, while the state $\top$ indicates a successful attack. $\bot, \top$ are two new states that are different from the states in $Q \times X \times Z$, $\Sigma^{GP} = \Sigma_o \times ((\Sigma_{o,A} \cup \{\epsilon\}) \times \Gamma) \cup \Sigma_{uo} \times \{\epsilon\} \cup \Sigma_{c,A}$ and the partial transition function

$$\delta^{GP} : (Q \times X \times Z \cup \{\bot, \top\}) \times \Sigma^{GP} \longrightarrow Q \times X \times Z \cup \{\bot, \top\}$$

is defined as follows.

1) For any $q, q' \in Q$, $x, x' \in X$, $z, z' \in Z$, $\gamma \in \Gamma$, $\sigma \in \Sigma_{o,A}$, $\delta^{GP}((q, x, z), (\sigma, (\sigma, \gamma))) = (q', x', z')$ iff $\delta(q, \sigma) = q'$, $\zeta^A(x, (\sigma, \gamma)) = x'$ and $\eta(z, \sigma) = z'$.
2) For any $q, q' \in Q$, $x, x' \in X$, $z, z' \in Z$, $\gamma \in \Gamma$, $\sigma \in \Sigma_o - \Sigma_{o,A}$, $\delta^{GP}((q, x, z), (\sigma, (\epsilon, \gamma))) = (q', x', z')$ iff $\delta(q, \sigma) = q'$, $\zeta^A(x, (\sigma, \gamma)) = x'$ and $\eta(z, \sigma) = z'$.
3) For any $q, q' \in Q$, $x, x' \in X$, $z, z' \in Z$, $\sigma \in \Sigma_{uo}$, $\delta^{GP}((q, x, z), (\sigma, \epsilon)) = (q', x', z')$ iff $\delta(q, \sigma) = q'$, $\zeta^A(x, \sigma) = x'$ and $\eta(z, \sigma) = z'$.
4) For any $q \in Q$, $x \in X$, $z \in Z$, $\sigma \in \Sigma_{c,A}$, $\delta^{GP}((q, x, z), \sigma) = \top$ iff $\delta(q, \sigma)!$, $\neg\zeta(x, \sigma)!$ and $\eta(z, \sigma) \in Z_m$.
5) For any $q \in Q$, $x \in X$, $z \in Z$, $\sigma \in \Sigma_{c,A}$, $\delta^{GP}((q, x, z), \sigma) = \bot$ iff $\delta(q, \sigma)!$, $\neg\zeta(x, \sigma)!$ and $\eta(z, \sigma) \notin Z_m$.

3. Subset Construction w.r.t. the Attacker's Observation Alphabet:

Given $GP(G, S^A, H)$, we shall now perform a subset construction on $GP(G, S^A, H)$, with respect to the attacker's observation alphabet.

Let $GPS(G, S^A, H)$ denote the sub-automaton of $GP(G, S^A, H)$ with state space restricted to $Q \times X \times Z$. The alphabet of $GPS(G, S^A, H)$ is $\Sigma^{GPS} = \Sigma_o \times ((\Sigma_{o,A} \cup \{\epsilon\}) \times \Gamma) \cup \Sigma_{uo} \times \{\epsilon\}$. Let $GPS^2(G, S^A, H)$ denote the automaton that is obtained from $GPS(G, S^A, H)$ by projecting out the first component of the event of each transition. Let

$SUB(GP(G, S^A, H))$ denote the the automaton that is obtained from $GPS^2(G, S^A, H)$ by determinization. In addition, we augment $SUB(GP(G, S^A, H))$ with a labeling function $Lf : Y \mapsto 2^{\Sigma_{c,A}}$, which is defined such that for any $y \in Y$ and any $\sigma \in \Sigma_{c,A}$, $\sigma \in Lf(y)$ iff there exists some $v \in y$ such that, 1) $\delta^{GP}(v, \sigma) = \top$ and 2) for any $v' \in y$, $v' \neq v$ implies $\delta^{GP}(v', \sigma) \neq \bot$.

We here recall the following result [11], which states that Algorithm **NA**(G, S', H) is correct.

Theorem 3. *Let the plant G and the normal supervisor S be given. Then, (G, S) is unattackable with respect to $(\Sigma_{c,A}, \Sigma_{o,A})$ and L_{dmg} iff for every reachable state y in $SUB(GP(G, S^A, H))$ it holds that $Lf(y) = \varnothing$.*

IV. Conclusion

In this paper, we have proposed and addressed the problem of supervisor obfuscation in actuator enablement attack scenario, in the setting where the attacker is able to eavesdrop the control commands issued by the supervisor and under a normality assumption on the actuator attackers and supervisors.

The algorithm proposed in this paper for solving the supervisor obfuscation problem relies on an SAT solver for computing behavior-preserving supervisors; a behavior-preserving supervisor is accepted as a solution of the problem only when it passes the non-attackability verification. The method used in this paper enumerates each possible behavior-preserving supervisor and then verifies its non-attackability. Although SAT solver is relatively efficient in computing behavior-preserving supervisors, the enumeration process is still causing massive degradation on the performance of the overall algorithm. In future work, we intend to develop more efficient algorithms that can be used for directly searching for obfuscated supervisors.

References

[1] L. K. Carvalho, Y. C. Wu, R. Kwong, S. Lafortune, "Detection and prevention of actuator enablement attacks in supervisory control systems", International Workshop on Discrete Event Systems, pp. 298-305, 2016.

[2] R. Su, "Supervisor synthesis to thwart cyber-attack with bounded sensor reading alterations", Automatica, accepted, 2018.

[3] R. M. Goes, E. Kang, R. Kwong, S. Lafortune, "Stealthy deception attacks for cyber-physical systems", Conference on Decision and Control, pp: 4224-4230, 2017.

[4] R. Lanotte, M. Merro, R. Muradore, L. Vigano, "A formal approach to cyber-physical attacks", IEEE Computer Security Foundations Symposium, pp: 436-450, 2017.

[5] A. Jones, Z. Kong, C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach", Conference on Decision and Control, pp: 848-853, 2014.

[6] M. Wakaiki, P. Tabuada, J. P. Hespanha, "Supervisory control of discrete-event systems under attacks", arXiv:1701.00881v1, 2017.

[7] P. M. Lima, M. V. S. Alves, L. K. Carvalho, M. V. Moreira, "Security against network attacks in supervisory control systems", IFAC: 12333-12338, 2017.

[8] R. Rajkumar, I. Lee, L. Sha, J. Stankovic, "Cyber-physical systems: The next computing revolution", Design Automation Conference, pp: 731-736, 2010.

[9] Y. Mo, H. J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, B. Sinopoli, "Cyberphysical security of a smart grid infrastructure", Proceedings of the IEEE, pp: 195-209, 2012.

[10] D. Kushner, "The real story of stuxnet", IEEE Spectrum 53, No. 3, 48, 2013.

[11] Lin, L., Thuijsman, S., Zhu, Y., Ware, S., Su, R., & Reniers, M. (2018). Synthesis of Successful Actuator Attackers on Supervisors. arXiv preprint arXiv:1807.06720.

[12] D. Neider. "Computing minimal separating DFAs and regular invariants using SAT and SMT solvers", ATVA, pp. 354-369, 2012.

[13] A. F. Vaz, W. M. Wonham. "On supervisor reduction in discrete-event systems", International Journal of Control, 44(2), pp: 475-491, 1986.

[14] R. Su, W. M. Wonham. "Supervisor reduction for discrete-event systems", Discrete Event Dynamic Systems, pp: 14-31, 2004.

[15] R. Su. "What information really matters in supervisor reduction?", arXiv:1608.04104.

[16] Y. Chen, A. Farzan, E. M. Clarke, Y. Tsay, B. Wang. "Learning minimal separating DFA's for compositional verification", TACAS, pp. 31-45, 2009.

[17] Biere, A., Heule, M. and van Maaren, H. eds., 2009. Handbook of satisfiability (Vol. 185). IOS press.

[18] W. M. Wonham, K. Cai, *Supervisory Control of Discrete-Event* Systems, Monograph Series Communications and Control Engineering, Springer, 2018, in press.

[19] J. E. Hopcroft, J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Massachusetts, 1979.

[20] O. Grumberg, A. Schuster, A. Yadgar. "Memory efficient all-solutions SAT solver and its application for reachability analysis", FMCAD, pp. 275-289, 2004.

[21] T. Toda, T. Soh. "Implementing efficient all solutions SAT solvers", arXiv:1510.00523v1, 2015.

[22] Lin, F. (2011). Opacity of discrete event systems and its applications. Automatica, 47(3), 496-503.

[23] A. Bergeron, "A unified approach to control problems in discrete event processes",RAIRO-Theoretical Informatics and Applications, 27(6): 555-573,1993.