# Synthesis of Sensor Deception Attacks for Systems Modeled as Probabilistic Automata

Rômulo Meira-Góes, Raymond Kwong and Stéphane Lafortune

***Abstract*— We study the security of control systems in the context of the supervisory control layer of stochastic discrete-event systems. Control systems heavily rely on correct communication between the plant and the controller. In this work, we consider that such communication is partially compromised by a malicious attacker. The attacker has the ability to modify a subset of the sensor readings and mislead the supervisor, with the goal of inducing the system into an unsafe state. We consider this problem from the attacker's viewpoint and investigate the synthesis of an attack strategy for systems modeled as probabilistic automata. Specifically, we quantify each attack strategy based on the likelihood of successfully reaching an unsafe state. The solution methodology that we develop uses techniques from the area of stochastic graph-games, specifically turn-based one-player stochastic reachability games.**

## I. INTRODUCTION

The correct behavior of control systems depends heavily on correct communication between the uncontrolled plant and the controller. Most works in the existing literature on control with robust communication focuses on random communication faults, such as delays, information loss, and so forth. However, in recent years, there has been an interest in considering that a "smart" agent, or attacker, could be responsible for communication faults. In this paper, we consider such an attack model, where the goal of the attacker is to induce the controller to steer the system to an unsafe state by altering the communications from the system sensors to the controller. Our analysis is at the supervisory control level; hence, we adopt a discrete event modeling formalism, where system operation and communications are event-based. In contrast to prior work on sensor deception attacks for Discrete Event Systems (DES), where logical models are used, we model the system as a *Probabilistic Finite-State Automaton*. This allows us to quantify, in a probabilistic sense, attack strategies.

Several works have addressed in recent years problems of security in the field of DES; see, e.g, [1], [10], [11], [12], [14], [15]. Most of these works focus on the intrusion detection problem for some classes of attacks, while some of them study the synthesis of attacks that might go unnoticed by the detectors [11], [12], [14]. The results of [11], [12], [14] are all in the context of logical DES models. To the best of our knowledge, this paper is the first one to study the synthesis of attack strategies for stochastic DES models, at the supervisory level of the control system.

Of particular relevance to this paper is the recent work in [12]; we use herein a similar framework for how attacks take place on the communication from the sensors to the controller. In [12], an attacker has compromised a subset of the sensors (i.e., observable events) and is able to delete sensor readings or insert fictitious ones in the communication channel. The problem investigated is the synthesis of stealthy *sensor deception attacks* for a known *logical* control system. In this work, we also investigate the problem of synthesis of *sensor deception attacks*. However, we study this synthesis problem for *stochastic* systems. This gives rise to a broader class of attack strategies, as compared with [12].

As a consequence of the stochastic control system model that we adopt, it is possible to quantify each attack strategy by the likelihood of reaching the unsafe region of the uncontrolled plant. In this manner, a quantitative measure is introduced in the synthesis problem of attack strategies. We investigate the synthesis of an *optimal attack strategy*, where the optimality criterion is the likelihood of reaching the unsafe region.

Our solution methodology employs results from the area of stochastic graph-games, more specifically $1\frac{1}{2}$-player reachability games [4]. First, we show how to build the "right" *game arena* that captures the interaction of the attacker and the control system. Next, we show that the solution of the $1\frac{1}{2}$-player reachability game played in the built arena can be mapped to the solution of our problem.

Our presentation is organized as follows. Section II introduces necessary background and the notation used throughout the paper. In Section III, we review the model of sensor deception attacks and then formalize the problem of synthesis of optimal attack functions. In section IV, we first define the construction of the game arena for $1\frac{1}{2}$-player reachability games and review solution methods for such games. Next, we show how to relate the two problems of: (i) $1\frac{1}{2}$-player reachability game and (ii) synthesis of an optimal attack function. Finally, we conclude the paper in Section V. Due to space limitations, several proofs have been omitted.

## II. PRELIMINARIES

### A. Supervisory Control

We consider the supervisory level of a feedback control system, where the uncontrolled system is modeled as a Deterministic Finite-State Automaton (DFA) in the discrete-event modeling formalism. A DFA is denoted by $G =$

R.M.G. and S.L. are with the Department of EECS, University of Michigan, USA. Their work was supported in part by US NSF grants CNS-1421122, CNS-1446298 and CNS-1738103. {romulo,stephane}@umich.edu

R.K. is with the ECE Department, University of Toronto, Toronto, ON M5S 3G4, Canada. His work was supported by Natural Sciences and Engineering Research Council of Canada (grant RGPIN-2015-04273). kwong@control.utoronto.ca

$(X_G, \Sigma, \delta_G, x_{0,G})$, where $X_G$ is the finite set of states, $\Sigma$ is the finite set of events, $\delta_G : X_G \times \Sigma \to X_G$ is the partial transition function and $x_{0,G}$ is the initial state. The function $\delta_G$ is extended, in the usual manner, to the domain $X_G \times \Sigma^*$. The language generated by $G$ is defined by $\mathcal{L}(G) = \{s \in \Sigma^* | \delta_G(x_{0,G}, s)!\}$, where ! means that the function is defined for these arguments. We define $\Gamma_G(x) = \{e \in \Sigma | \delta(x, e)!\}$ as the active event set of state $x \in X_G$.

In the context of the supervisory control theory of DES [13], system $G$ is considered as the uncontrolled system (plant) that needs to be controlled in order to satisfy given safety specifications. In order to control $G$, the event set $\Sigma$ is partitioned into two disjoint sets, $\Sigma_c$ and $\Sigma_{uc}$. These sets are, respectively, the set of controllable events and the set of uncontrollable events. The safety specifications on $G$ are enforced by a supervisor, denoted by $S$. The supervisor $S$ dynamically enables/disables controllable events to generate provably "safe" controlled behavior, denoted by $S/G$. The behavior of $S/G$ is the closed-loop prefix-closed language $\mathcal{L}(S/G)$; see, e.g., [2]. A supervisor is formally defined as a function $S : \Sigma^* \to \Gamma$, where $\Gamma = \{\gamma \subseteq \Sigma | \Sigma_{uc} \subseteq \gamma\}$ is the set of admissible control decisions. Such decisions must guarantee that the supervisor never disables an uncontrollable event. Without loss of generality, we assume that $S$ is realized by an automaton $R = (X_R, \Sigma, \delta_R, x_{0,R})$ (see, e.g., [2]). We will use both notations $S$ and $R$ interchangeably.

For any string $s \in \Sigma^*$, we use the following notation. We denote by $e_s^i$ the $i^{th}$ event of $s$ such that $s = e_s^1 e_s^2 \dots e_s^{|s|}$, where $|s|$ denotes the length of $s$. We denote by $s^i$ the $i^{th}$ prefix of $s$, namely $s^i = e_s^1 \dots e_s^i$ and $s^0 = \epsilon$. Finally, we use $\mathbb{N}$ to be the set of natural numbers, $[n]$ and $[n]^+$ to be, respectively, the set of natural numbers and the set of positive natural numbers both bounded by $n \in \mathbb{N}$.

### B. Stochastic Discrete Event Systems

We model a stochastic DES as a Probabilistic Finite-State Automaton (PFA). A PFA is denoted by $H = (X_H, \Sigma, \delta_H, P, x_{0,H})$, where $X_H$, $\Sigma$, and $x_{0,H}$ are defined as in a DFA, $P : X \times \Sigma \times X \to [0, 1]$ is the transition probability function and $\delta_H : X \times \Sigma \to X$ is the deterministic transition function defined by $P$. Given an event $e \in \Sigma$ and two states $x, y$, the function $P(x, e, y)$ specifies the probability of moving from state $x$ to state $y$ with event $e$. For simplicity, we write $P_e(x)$ when $\exists y \in X_H$ such that $P(x, e, y) > 0$. In general, for any $x \in X$ we have that $\sum_{e \in \Sigma} P_e(x) \leq 1$. However, in this paper we consider the family of nonterminating PFA, where for any $x \in X_H$, $\sum_{e \in \Sigma} P_e(x) = 1$. This assumption is without loss of generality, as any terminating PFA can be transformed to a nonterminating one, as shown in [9]. As was mentioned above, $\delta_H$ is defined by the transition probability function $P$, formally $\delta_H(x, e) = y$ if $P(x, e, y) > 0$. This transition function is assumed to be deterministic: if $P(x, e, y) > 0$ for some $x, y \in X_H$ and $e \in \Sigma$, then there does not exist $y^* \in X_H$, $y^* \neq y$, such that $P(x, e, y^*) > 0$. Using this definition, then every PFA $H$ can be associated with a corresponding DFA $G$ where $\delta_G(x, e) = \delta_H(x, e)$. In this manner, the language generated by $H$ is defined as the language generated by the associated $G$, i.e., $\mathcal{L}(H) := \mathcal{L}(G)$. For simplicity, whenever we use a DFA operator in a PFA $H$, it will mean that we are analyzing the associated DFA $G$.

Although $\mathcal{L}(H)$ is well defined, we want to define a probability space over this language. In [7], the notion of probabilistic-languages (p-languages) was introduced. Formally, $L_p(H) : \Sigma^* \to [0, 1]$ is defined as:

$$L_p(H)(\epsilon) = 1 \tag{1}$$

$$L_p(H)(se) = \begin{cases} L_p(H)(s)P(x, e, y) & \text{if } x = \delta_H(x_{0,H}, s) \\ & y = \delta_H(x_{0,H}, se) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In fact, $L_p(H)$ defines a probability measure over the $\sigma$-algebra $\mathcal{F}$ as defined in [7]. In this work, we use the same probability space as in [7]. One property from the measurable sets in $\mathcal{F}$ is that two *distinct* strings $s$ and $t$, such that no string is a prefix of the other one, generate independent measurable sets. This property implies that the probability of generating either one of these two strings is equal to $L_p(H)(s) + L_p(H)(t)$. This useful result will be exploited in both the problem formulation and the solution methodology. For more details on the probability space used in this paper, please see [7].

Supervisory control of stochastic DES was first introduced in the work of [9]. In this paper, we use the results of supervisory control of stochastic DES introduced by [8]. In [9], both the supervisor and the plant have stochastic models, namely PFA. On the other hand, the work of [8] only considers the plant to behave stochastically. The supervisor is defined to be non-probabilistic, as in the previously-described supervisory control framework. However, the control actions of the supervisor (which disable events) *do alter* the probabilistic behavior of the plant. Assume that the requirements for the stochastic controlled behavior are the same as the one presented for the non-probabilistic case, namely, safety requirements. Sufficient and necessary conditions for the existence of a supervisor for the above control problem are provided in [8]. A polynomial-time algorithm to synthesize maximally permissive supervisors is also presented.

In the presence of a supervisor $R$, the probability transition function of the plant $H$ needs to be updated accordingly. Namely, the disablement of events by $R$ increases the probability of the enabled ones. In other words, $R/H$ generates another p-language. Given a state $x \in X_H$, a state $q \in X_R$, and an event $e \in \Gamma_H(x) \cap \Gamma_R(q)$, the probability of $e$ being executed is given by the standard normalization:

$$P_e^{x,q} = \frac{P_e(x)}{\sum_{e' \in \Gamma_H(x) \cap \Gamma_R(q)} P'_e(x)} \tag{3}$$

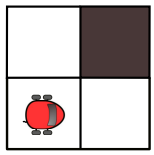
Without loss of generality, unsafe strings are considered to be those outside of the controlled language[1]. For-

[1] In the general case, behavior outside the controlled language might not be unsafe. However, the techniques here can be adapted for this case as well.

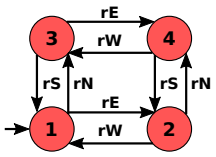
mally, unsafe strings are defined by the set $\mathcal{L}(H)\backslash\mathcal{L}(R/H)$. However, measuring this set would be challenging in our probability space. For this reason, let $U_{uns} = \{s \in \mathcal{L}(H)\backslash\mathcal{L}(R/H) | s^{|s|-1} \in \mathcal{L}(R/H)\}$ be the set with the *shortest* distinct strings that belong to $\mathcal{L}(H)\backslash\mathcal{L}(R/H)$. From now on, $U_{uns}$ is considered to be the set of unsafe strings. We define the set of unsafe state pairs for the controlled system $R/H$ by $X_{uns} = \{(x_1, x_2) \in X_H \times X_R | \nexists s \in \mathcal{L}(R/H)$ s.t. $x_1 = \delta_H(x_{0,H}, s) \wedge x_2 = \delta_R(x_{0,R}, s)\}$.

We use the following example as a running illustrative example throughout the paper.

**Example II.1.** Consider a robot that is navigating an area that has been partitioned as a grid with 2 rows and 2 columns, as shown in Figure 1(a). The robot moves freely in the region. However, it needs to do its moves without colliding with any obstacle. In our example, it needs to avoid the gray region.



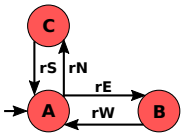
(a) Example robot in a 2 by 2 grid. The gray square represents an obstacle



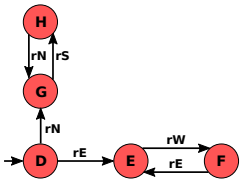

(b) General model of robot in a 2 by 2 grid

Fig. 1: Robot example

The robot is modeled as the PFA $H$ shown in Figure 1(b). Every event is controllable and the probability transition function is defined as $P(x, e, y) = \frac{1}{2}$ for the transitions defined by the model in Figure 1(b). We show two supervisors in Figure 2 that guarantee the safety of the robot. The supervisor is deterministic but its presence alters the behavior of $H$. For example, in the controlled behavior $R_1/H$, the probability of event $rS$ to occur at state 3 given that the supervisor is at state $C$ is $P_e^{3,C} = 1$.




(a) Supervisor $R_1$




(b) Supervisor $R_2$

Fig. 2: Supervisors for robot

### C. Stochastic Graph-Games

Two-player graph-games are an important field of study in several areas of computer science, and they have been thoroughly studied within these various fields; see [3] for details. In this work, we are interested in *perfect information turn-based one-player stochastic reachability games* ($1\frac{1}{2}$-player reachability games and also known as Markov Decision Processes). These games are defined by two players, namely Player 1 and Player Random, that play for an infinite number of rounds [4].

In these games, each vertex is assigned to a specific player, which at these vertices selects the successor state. Player random chooses its successor according to a probability distribution, hence the name. On the other hand, player 1 selects the successor state based on a specific goal. For these games, the goal of player 1 is described by a reachability condition. Consequently, the path generated by the infinite number of rounds played by both players is either winning or losing for player 1.

Formally, $1\frac{1}{2}$-player reachability games are defined over a directed graph $\mathcal{G} = (V, E)$ with $V = V_1 \cup V_r$, and a probability transition function $\lambda : V_r \times V \to [0, 1]$ with the following properties. The vertex set is partitioned as $V_1, V_r$ called "player 1" and "player random" vertices, respectively. At vertices $v \in V_1$, player 1 selects the successor vertex based on the edge set $E$. At a random vertex $v$, the successor vertex is chosen according to $\lambda$, namely, an edge $(v, u) \in E$ is selected with probability $\lambda(v, u)$ for $u \in V$. Note that, for any vertex $v \in V_R$ and any vertex $u \in V$, $(v, u) \in E$ if and only if $\lambda(v, u) > 0$. The pair $\mathcal{G}$ and $\lambda$ is also called the *arena* of the game. To complete the problem formulation, the set $Obj \subset V$ is defined to be the reachability objective for player 1. In other words, player 1 wins the game if it reaches any vertex in $Obj$.

We use the same terminology used in the literature of graph-games. A path or a run $\rho$ of the game $\mathcal{G}$ is an infinite sequence $\rho = v^0 v^1 \ldots$ such that for any $i \in \mathbb{N}$, $v^i \in V$ and $(v^i, v^{i+1}) \in E$. The set $Pref_v(\mathcal{G})$ is the set of all finite-length prefixes of runs in $\mathcal{G}$, starting from $v \in V$. Strategies in graph-games normally require memory and they are randomized. A deterministic (or pure) strategy is defined by a map $\sigma : V^* V_1 \to V$. However, it is known that for $1\frac{1}{2}$-player reachability games, pure and memoryless strategies suffice [4], e.g., $\sigma : V_1 \to V$.

Once a strategy $\sigma$ for player 1 is fixed, the path $\rho_v^\sigma$ results in a random walk in $\mathcal{G}$, and the strategy can be evaluated (policy evaluation). The game becomes a Markov chain and the probability of winning the game starting from any vertex is equal to the first passage probability from the initial vertex to the set $Obj$. We define by $Pref_v^\sigma(\mathcal{G})$ all finite-length prefixes of paths $\rho_v^\sigma$; formally: $Pref_v^\sigma = \{\rho = v^0 \ldots v^n | v^0 = v \wedge (\forall i \in [n] \Rightarrow (v^{i+1} = \sigma(v^0 \ldots v^i) \vee (v^i \in V_r \wedge (v^i, v^{i+1}) \in E)))\}$. The work of [5] provides polynomial-time algorithms to solve $1\frac{1}{2}$-player reachability games. For more details on stochastic graph-games, see [3].

## III. PROBLEM DESCRIPTION

We consider an uncontrolled plant described by PFA $H$ that is being controlled by a supervisor modeled by DFA $R$. The controlled system $R/H$ is assumed to be safe, i.e., the supervisor $R$ was constructed such that it provably prevents $H$ from generating any existent unsafe behavior. For instance, $R$ could have been designed using the methods presented in [8].

The correct operation of the system $R/H$ relies heavily on the correct bidirectional communication channel between the plant and the supervisor. Our goal is to investigate the

performance of $R/H$ when this assumption is violated. More specifically, this violation is due to a malicious attacker. Even though the attacker could undermine both directions of this communication, we only investigate corruption in the direction from plant to supervisor, i.e., compromised sensors.

As in [12], the attacker is capable of manipulating a subset of the events. We define this set as $\Sigma_a \subseteq \Sigma$, and it is called the *compromised* event set. Any event in $\Sigma_a$ can be manipulated by the attacker, where "manipulate" means ability to insert fictitious events in the channel or to erase events from the channel. Consequently, the supervisor might receive corrupted information from the plant. Under this unreliable communication model, the supervisor $R$ may not guarantee the safety requirement for the controlled system. This possible unsafe behavior is the focus of our paper.

The previous work [12] investigated if an attacker could modify, without being detected, the information sent from the plant to the supervisor while causing the plant to reach an unsafe state. If such modifications exist, the attacker has an attack strategy to reach its goal. In [12], both the plant and the supervisor were modeled as DFA, resulting in an analysis of terms of all possible behaviors, without quantification. In this paper, the plant is modeled as a PFA, as our objective is to obtain quantitative results for attack functions. Namely, we wish to investigate the likelihood of an attack strategy reaching the unsafe region of the plant. This probabilistic viewpoint is more general than the logical viewpoint considered in [12].

First, let us formally define the model of an attacker.

**Definition III.1.** An attacker that hijacks the events in $\Sigma_a$ in the communication channel between the plant and the supervisor is defined as a map $A : \Sigma^* \to \Sigma^*$. The map $A$ must satisfy the following conditions for any $s \in \Sigma^*$:

1) If $s = \epsilon$, then $A(s) \in \Sigma_a^*$
2) If $e_s^{|s|} \in \Sigma_a$, then $A(s) \in \Sigma_a^*$
3) If $e_s^{|s|} \in \Sigma \backslash \Sigma_a$, then $A(s) \in \{e_s^{|s|}\}\Sigma_a^*$

The attack function $A$ only defines the attack strategy for the last event. In other words, it just expresses that the last event $e_s^{|s|}$ is substituted by $A(s)$. For convenience, we define the function $\hat{A}$ that recursively concatenates these modifications for any string $s \in \Sigma^*$. For any $s \in \Sigma^*$, we define $\hat{A}(s) = \hat{A}(s^{|s|-1})A(s)$.

The introduction of the attack function $A$ in the communication channel between the plant and the supervisor generates a new controlled behavior. In fact, the supervisor $S$ and the attack function $A$ can be combined to form a new supervisor, denoted by $S_A$, where $S_A(s) = S \circ \hat{A}(s)$ is the resulting control action, under attack, after string $s$ has been executed by the system. Given $S_A$, we then obtain $\mathcal{L}(S_A/H)$ as the new controlled language, which is defined as follows:

1) $\epsilon \in \mathcal{L}(S_A/H)$
2) $s \in \mathcal{L}(S_A/H) \wedge se \in \mathcal{L}(H) \wedge e \in S_A(s) \Leftrightarrow se \in \mathcal{L}(S_A/H)$

*Remark 1:* $S_A/H$ generates a p-language in the same manner as $S/H$.

*Remark 2*: In the definition of the language of $S_A/H$, the attacker completes its string modification without any interruption of the plant $H$. In other words, the plant $H$ does not execute any event in the middle of the attacker editions.

Next, we introduce the notions of *complete* and *consistent* attack function. Intuitively, an attack function is complete if it is defined for every string in the new controlled behavior $\mathcal{L}(S_A/H)$. This means that the attacker always "knows" what to do next. An attack function is consistent if its modifications are consistent with the supervisor $S$, i.e., the supervisor receives an event that is possible in its current state. In this work, we only consider complete and consistent attack functions. The formal definitions of these notions are given next.

**Definition III.2.** An attack function $A$ is *complete* if for any $s$ in $\mathcal{L}(S_A/H)$, we have that $A(s)$ is defined.
An attack function $A$ is *consistent* if for any $e \in \Sigma$, $s$ in $\mathcal{L}(S_A/H)$ s.t. $se \in \mathcal{L}(S_A/H)$ with $A(se) = t$, then $e_t^{i+1} \in S(\hat{A}(s)t^i)$ for all $i \in |t| - 1$.

Since the controlled behavior under the influence of attack function $A$ is well defined by $\mathcal{L}(S_A/H)$, we can define the objective of the attacker based on this language. The attack function is successful if $\exists s \in U_{uns}$ such that $s \in \mathcal{L}(S_A/H)$. Moreover, an attack function $A$ is quantified by the probability of generating the strings $s \in U_{uns} \cap \mathcal{L}(S_A/H)$, since $L_p(S_A/H)$ is well defined. We define the *winning level of* $A$ to be the probability that $S_A/H$ generates unsafe strings. Formally,

$$win_A = \sum_{s \in U_{uns}} L_p(S_A/H)(s) \tag{4}$$

*Remark:* Every string in $U_{uns}$ is distinct, consequently $win_A \leq 1$.

The definition of the value $win_A$ makes it possible to compare attack functions. For example, given two attack functions $A$ and $A'$, if $win_A \geq win_{A'}$, then it means that strategy $A$ is more likely to reach the unsafe region of $H$ than $A'$. A natural question to ask is if there exists an attack function that is more likely to reach the unsafe region than any other attack function. Formally, the problem is posed as follows.

**Problem III.1.** [Optimal Attack Function Synthesis Problem] Given a plant modeled as PFA $H$, a supervisor modeled as DFA $R$, and the set of compromised events $\Sigma_a \subseteq \Sigma$, synthesize a complete and consistent attack function $A^*$, if one exists, such that for any other complete and consistent attack function $A$, the following holds:

$$win_{A^*} \geq win_A \tag{5}$$

## IV. SOLUTION

The solution of Problem **III.1** is obtained by relating it to the $1\frac{1}{2}$-player reachability graph-game problem. We start by showing the construction of the game arena and its reachability objective using $H$, $R$, and $\Sigma_a$. Next, we review one technique to solve the reachability game. Finally, we

map the solutions of these two problems to each other in the third part of this section.

### A. Construction of the Arena

In the problem definition, we assume that $H$ and $R$ are given as a PFA and a DFA, respectively. For this reason, the vertices of player random are constructed based on these two models. On the other hand, player 1 represents the attack function. Therefore, its vertices are defined based on $H$, $R$ and $\Sigma_a$. The arena for the $1\frac{1}{2}$-player reachability graph-game is defined as follows.

**Definition IV.3.** Given plant $H$, supervisor $R$, and compromised event set $\Sigma_a$, the game arena $\mathcal{A} = (\mathcal{G}, \lambda)$ is constructed in the following manner.

- $V_1 \subseteq X_H \times X_R \times (\Sigma \cup \{\epsilon\} \cup \{\epsilon_e | e \in \Sigma_a\})$ and $V_r \subseteq X_H \times X_R$. For a vertex $v$, we denote by $v_H$, $v_R$ and $v_e$ the plant state, the supervisor state, and the last event received by the supervisor, i.e., $v = (v_H, v_R, v_e)$ if $v \in V_1$ or $v = (v_H, v_R)$ if $v \in V_r$. The reason for introducing the set $\{\epsilon_e | e \in \Sigma_a\}$ is to differentiate vertices in $V$. (This will become clear once the set $E$ is defined.) Moreover, we treat the elements in this set as empty strings when constructing strings of events.
- The set $E$ is defined as follows:
  1) For any $v, u \in V_1$ s.t. $(v_H, v_R) \notin X_{uns}$, then $(v, u) \in E$ if $\exists e \in \Sigma_a$ s.t. $u_H = v_H$, $u_R = \delta_S(v_R, e)$ and $u_e = e$.
  2) For any $v \in V_1$ s.t. $(v_H, v_R) \notin X_{uns}$ and $u \in V_r$, then $(v, u) \in E$ with $u = (v_H, v_R)$ if $\exists e \in \Gamma_H(v_H) \cap \Gamma_S(v_R)$.
  3) For any $v \in V_r$ and $u \in V_1$, then $(v, u) \in E$ if $\exists e \in \Gamma_H(v_H) \cap \Gamma_S(v_R)$ where $u = (\delta_H(v_H, e), v_R, \epsilon_e)$ if $e \in \Sigma_a$ and $(u_H, u_R) \notin X_{uns}$, or $u = (\delta_H(v_H, e), \delta_S(v_R, e), e)$ otherwise.
- Post-Processing:
  1) If $v \in V_1$ and $\nexists u \in V$ s.t. $(v, u) \in E$, then $(v, u) \in E$ with $u = (v_H, v_R, \epsilon)$
- The probability function $\lambda$ is defined for any $(v, u) \in E$ s.t. $v \in V_r$ and $u \in V_1$ as $\lambda(v, u) = P_e^{v_H, v_R}$ where $e \in \Sigma$ and $u_H = \delta_H(v_H, e)$.

The rules defining the set $E$ for arena $\mathcal{A}$ capture the interaction between the controlled system and the attack function. Rule 1 captures player 1 sending information to supervisor $R$; for this reason only states in $R$ are updated. Rule 2 says that player 1, at that moment, does not want to share any more information with $R$; it wants to wait for the response of plant $H$. Moreover, rules 1 and 2 enforce that states in $X_{uns}$ are absorbing, e.g., no transitions are defined after these states are entered. Rule 3 is related to the execution of events in $H$. The post-processing on $E$ ensures that no deadlock is present in $\mathcal{G}$.

In order to complete the construction of the game, we must specify the reachability objective for player 1. This objective must be aligned with the objective of the attack function in Problem **III.1**. Formally, player 1 needs to reach any state in $Obj = \{v \in V_1 | (v_H, v_R) \in X_{uns}\}$. In this manner, the reachability game is well defined by $\mathcal{A}$ and the set $Obj$; in this regard, we set $v_0 = (x_{0,H}, x_{0,R}, \epsilon)$ to be the initial vertex. In order words, we only analyze runs of $\mathcal{A}$ that start from vertex $v_0$.

**Example IV.2.** In order to illustrate the construction of $\mathcal{A}$, recall the plant $H$ and the supervisor $R_1$ shown in Ex. **II.1**. The compromised event set for this example is $\Sigma_a = \{rE\}$. Figure 3 represents $\mathcal{A}$ for the controlled system $R_1/H$. In Figure 3, the vertices in $V_1$ are represented by rectangles, while vertices in $V_r$ are identified by diamonds. The function $\lambda$ is defined by the values in the edges between a random vertex and a player-1 vertex, i.e., $\lambda((3, C), (1, A, rS)) = 1$.

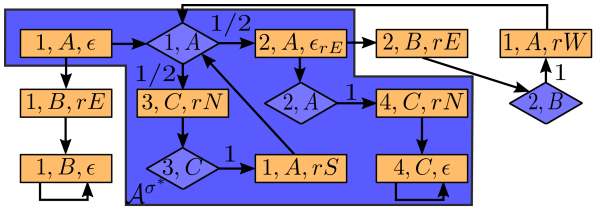



Fig. 3: Arena for controlled system $R_1/H$ and $\Sigma_a = \{rE\}$

### B. Solution of $1\frac{1}{2}$-player reachability graph-games

We briefly review one technique to solve $1\frac{1}{2}$-player reachability games. The result is attributed to [6] and shown in [5]. The optimal winning value for each vertex is calculated by solving a linear programming (LP) problem [5].

In order to introduce the LP problem, some notation is necessary. For any $v \in V$, define $val(v)$ to be the probability that player 1 wins the game starting in $v$. The optimal value of winning for each vertex in $V$ is the optimal solution to the following LP problem. (Adapted from [5].)

$$\begin{aligned}
&\text{minimize} \sum_{v \in V} val(v), \text{ subject to} \\
&val(v) \geq val(u) && \text{if } v \in V_1 \wedge (v, u) \in E \\
&val(v) = \sum_{(v,u) \in E} \lambda(v, u) val(u) && \text{if } v \in V_r \\
&val(v) \geq 0 && v \in V \\
&val(v) = 1 && v \in Obj
\end{aligned} \tag{6}$$

*Remark 1:* The optimal winning strategy is defined based on the optimal winning values of each vertex, i.e., memoryless strategy. It is defined by simply selecting at each vertex $v \in V_1$ any neighbor with the largest optimal winning value. Next, we show the optimal winning values and the optimal winning strategy for the game $\mathcal{A}$ constructed in Ex. **IV.2**.

*Remark 2:* Although the formulation of the LP may appear counter-intuitive at first (since the LP minimizes over the winning values and the attacker wants to maximize the winning values), it does find the optimal winning values for each vertex. Specifically, examining the LP constraints reveals that minimizing over the sum of $val(v)$ does return the maximal winning values.

**Example IV.3.** In Ex. **IV.2**, we presented the arena $\mathcal{A}$ for the controlled system $R_1/H$ and $\Sigma_a = \{rE\}$. Solving the LP problem formulated as in Equation (6) for arena $\mathcal{A}$ and $Obj$ returns the optimal value $val(1, A, \epsilon) = 1$. Moreover, the optimal value is $val(v) = 1$ for $v \in V \backslash \{(1, B, rE), (1, B, \epsilon)\}$ and $val(v) = 0$ for $v \in \{(1, B, rE), (1, B, \epsilon)\}$. One optimal strategy is defined as $\sigma^*((1, A, \epsilon)) = (1, A)$, $\sigma^*((3, C, rN)) = (3, C)$, $\sigma^*((1, A, rS)) = (1, A)$, $\sigma^*((2, A, \epsilon_{rE})) = (2, A)$, $\sigma^*((4, C, rN)) = (4, C, \epsilon)$, and $\sigma^*((4, C, \epsilon)) = (4, C, \epsilon)$. This strategy limits $\mathcal{A}$ within the blue shaded region in Figure 3.

Similarly, we can solve the problem for system $R_2/H$ and $\Sigma_a = \{rE\}$. Solving the LP problem for this controlled system returns the optimal value $val(1, D, \epsilon) = 1/2$.

### C. Solution of the Optimal Attack Function Synthesis Prob.

The definition of winning level is directly related to an attack function $A$. In other words, once an attack function $A$ is fixed, we obtain the language $\mathcal{L}(S_A/H)$ and consequently the value of $win_A$. The same idea applies to the game played in $\mathcal{A}$; once the strategy of player 1 is fixed, we can calculate the probability of winning the game at any vertex. The construction of $\mathcal{A}$ and $Obj$ ties these two problems together, as we formalize in this section. First, we show that any finite run in $\mathcal{A}$ starting from $v_0$ is related to a string in $H$.

**Proposition IV.1.** For every run $\rho \in Pref_{v_0}(\mathcal{A})$ with $\rho = v^0 \dots v^n$, there exists $s \in \mathcal{L}(H)$ s.t. $\delta_H(x_{0,H}, s) = v^n_H$.

The second proposition is also related to the construction of $\mathcal{A}$. While the first proposition relates $\mathcal{A}$ with $H$, it would be interesting to have a converse proposition. Nonetheless, this is not possible since $\mathcal{A}$ is constructed based on the compromised event set $\Sigma_a$. For this reason, the proposition below ties the relation between the language $\mathcal{L}(S_A/H)$ for any complete and consistent attack function $A$ with finite runs in $\mathcal{A}$.

**Proposition IV.2.** Given any complete and consistent attack function $A$, for any $s \in \mathcal{L}(S_A/H)$ and $t = \hat{A}(s)$, there exists a unique $\rho \in Pref_{v_0}(\mathcal{A})$ with $\rho = v^0 u^1 v^1 \dots u^{|s|} v^{|s|}$ s.t. $t = t_0 t_1 \dots t_{|s|}$, where $v^i \in V_1^*$ is defined for all $i \in [|s|]$ and $k_i \in \mathbb{N}$ as $v^i = v^{i0} \dots v^{ik_i}$, $u^i \in V_r$, $t_i = v_e^{i0} \dots v_e^{ik_i}$ and $v_e^{nk_n} \neq \epsilon$.

Prop. **IV.2** relates any pair of strings $(s, \hat{A}(s))$, given an attack function $A$, to a unique finite run in $\mathcal{A}$. If we eliminate the infinite runs that were added in the post-processing of $E$ in Def. **IV.3**, then there exists a one-to-one map between pairs of strings and finite runs. Consequently, attack functions can be related to strategies for the game $\mathcal{A}$ by a one-to-one map. Note that, the runs related to $(s, \hat{A}(s))$ follow the pattern described in Prop. **IV.2**, e.g., $\rho = v^0 u^1 v^1 \dots u^n v^n$. Thus, the infinite runs are either because $v^n$ generates a cycle within vertices in $V_1$ or $\rho = v^0 u^1 v^1 \dots u^j v^j \dots u^j v^j$. The proof of Prop. **IV.2** gives us a way to convert any attack function to a strategy, but it is unclear if the converse is true. Lemma **IV.1** below formalizes this one-to-one relation between attack functions and strategies. First, we provide more intuition about this relation since it is a key concept for the results of this section.

Figure 3 helps us understand this one-to-one map for memoryless strategies. Assume that strategy $\sigma$ is memoryless, then $\mathcal{A}^\sigma$ is the substructure of $\mathcal{A}$ when the game is limited by strategy $\sigma$. In Figure 3, $\mathcal{A}^{\sigma^*}$ is the substructure generated by strategy $\sigma^*$ as it was defined in Ex. **IV.3**. We want to show that we can define an attack function based on $\sigma^*$, namely $A_{\sigma^*}$.

Let us select the run $\rho_1 = v^{00} u^1 = (1, A, \epsilon)(1, A)$, since $\sigma^*(\rho_1) \in V_r$. The run $\rho$ has only one vertex, thus neither $H$ nor $R$ have evolved. For this reason, the pair of strings $(\epsilon, \epsilon)$ is associated with this run. Consequently, we set $A_{\sigma^*}(\epsilon) = \epsilon$. Next, select the run $\rho_2 = v^{00} u^1 v^{10} u^2 = (1, A, \epsilon)(1, A)(2, A, \epsilon_{rE})(2, A)$. Plant $H$ has evolved from state 1 to state 2 in $\rho_2$, for this reason the string $rE$ was executed in $H$. On the other hand, the supervisor $R$ did not change its state in $\rho_2$, which means that the supervisor observed string $\epsilon$, i.e., it remains in state A. Consequently, the pair of strings $(rE, \epsilon)$ is associated with $\rho_2$. In this case, we set $\hat{A}_{\sigma^*}(rE) = \epsilon$. If we continue this process, $A_{\sigma^*}$ will be a well-defined complete and consistent attack function.

We just argued that from a memoryless strategy, one can construct an attack function, and the proof of Prop. **IV.2** shows the converse statement. Lemma **IV.1** formalizes this claim. Moreover, Lemma **IV.1** does not make the assumption of memoryless strategies.

**Lemma IV.1.** Any strategy $\sigma$ for the game $\mathcal{A}$ can be written as a complete and consistent attack function $A_\sigma$. Conversely, every complete and consistent attack function $A$ can be written as a strategy $\sigma_A$ for the game played in $\mathcal{A}$.

*Proof.* We first show how to construct a complete and consistent attack function $A_\sigma$ given a strategy $\sigma$ for the game played in $\mathcal{A}$. The second part of the proof, to construct a strategy $\sigma_A$ given an attack function $A$, follows from the proof of Proposition **IV.2**.

We want to show that for any strategy $\sigma : V^* V_1 \to V$ we can define a complete and consistent attack function $A_\sigma$. Since the strategy is fixed, we only analyze runs that are generated by $\mathcal{A}$ restricted by the strategy $\sigma$ ($\mathcal{A}^\sigma$), e.g. runs that generate prefixes in $Pref^\sigma_{v_0}(\mathcal{A})$. More specifically, to generate $A_\sigma$ only two types of runs are analyzed. These type of runs are related to the case where for a specific $s$, $A_\sigma(s)$ is defined to be an arbitrarily long string, or a finite string.

Let us use the same notation as in Proposition **IV.1** to describe these runs, e.g. $v^i \in V_1^*$ and $u^i \in V_r$ for $i \in \mathbb{N}$. The first case is the infinite runs that are described in the following form.

$$\rho_1 = v^0 u^1 \dots u^n v^n \tag{7}$$

Thus, the sequence of vertices in $v^n$ form a cycle. In other words, after a finite prefix the run remains only in $V_1$ vertices. For example, if we define $\sigma$ for the arena in Figure 3 as $\sigma((1, A, \epsilon)) = (1, B, rE)$, $\sigma((1, B, rE)) = (1, B, \epsilon)$ and $\sigma((1, B, \epsilon)) = (1, B, \epsilon)$, then

run $(1, A, \epsilon)(1, B, rE)(1, B, \epsilon)(1, B, \epsilon)\ldots$ belongs to $\mathcal{A}^\sigma$ and it follows the pattern as in Equation 7.

The second case is the infinite runs that are incrementally defined by the following prefixes.

$$\rho_2 = v^0 u^1 v^1 \ldots u^n v^n u^{n+1} \tag{8}$$

Therefore, it is all the runs that alternate between a finite sequence of vertices in $V_1$ and one vertex in $V_r$.

We build $A_\sigma$ incrementally, starting with $i = 0$. For the initial condition, if there exists an infinite run $\rho_1 = v^0 = v^{00}v^{01}\ldots$, then we define $A_\sigma(\epsilon) = v_e^{00}v_e^{01}\ldots$. Otherwise, we select the run $\rho_2 = v^0u^1$ and define $A_\sigma(\epsilon) = v_e^{00}\ldots v_e^{0k_0}$. Note that, these two cases are mutually exclusive and $A_\sigma(\epsilon)$ is only defined once. If the run $\rho_1 = v^0 = v^{00}v^{01}\ldots$ exists in $\mathcal{A}^\sigma$ then the run $\rho_2 = v^0u^1$ will not exist.

For $i = 1$. In this case, we first select all the infinite runs $\rho_1 = v^0u^1v^1$ as in Equation (7). The run $\rho_1$ is related with the pair of strings $(s, t)$, where $s = e_s^1$ with $v_H^{10} = \delta_H(u_H^1, e_s^1)$ and $t = t_0t_1$ with $t_0 = v_e^{00}\ldots v_e^{0k_0}$ and $t_1 = v_e^{10}v_e^{11}\ldots$. Thus, we define $A_\sigma(s) = t_1$.

Similarly, we select all finite runs $\rho_2 = v^0u^1v^1u^2$ as in Equation (8). The run $\rho_2$ is related with the pair of strings $(s, t)$, where $s = e_s^1$ with $v_H^{10} = \delta_H(u_H^1, e_s^1)$ and $t = t_0t_1$ with $t_j = v_e^{j0}\ldots v_e^{jk_j}$ for $j \in [1]$. Thus, we define $A_\sigma(s) = t_1$. The difference between the two runs is that $t_1$ is an arbitrarily long string for the first type of runs. On the other hand, $t_1$ is a finite string for the second type of runs.

Let us generalize $A_\sigma$ for any $i \in \mathbb{N}$. First select all the infinite runs $\rho_1 = v^0u^1v^1\ldots u^iv^i$ as in Equation (7). The run $\rho_1$ is related with the pair of strings $(s, t)$, where $s = e_s^1\ldots e_s^i$ with $v_H^{j0} = \delta_H(u_H^j, e_s^j)$ for $j \in [i]^+$ and $t = t_0t_1\ldots t_i$ with $t_j = v_e^{j0}\ldots v_e^{jk_0}$ for any $j \in [i]$ and $t_i = v_e^{i0}v_e^{i1}\ldots$. Thus, we define $A_\sigma(s) = t_i$. Similarly, we select all the finite runs $\rho_2 = v^0u^1v^1\ldots u^iv^i$ as in Equation (8). The run $\rho_2$ is related with the pair of strings $(s, t)$, where $s = e_s^1\ldots e_s^i$ with $v_H^{j0} = \delta_H(u_H^j, e_s^j)$ for $j \in [i]^+$ and $t = t_0\ldots t_i$ with $t_j = v_e^{j0}\ldots v_e^{jk_j}$ for $j \in [i]$. Thus, we define $A_\sigma(s) = t_i$.

Each run in the set of runs in $\mathcal{A}^\sigma$ and that follows the pattern of Equation (7) or (8) has a one-to-one map to a string $s$ that is executed in $\mathcal{L}(H)$. For this reason, $A_\sigma(s)$ is a well defined map. Note that $A_\sigma$ is complete and consistent by construction. Moreover, $A_\sigma$ is uniquely defined.

*Remark:* Proposition **IV.1** says that there exists a string in $\mathcal{L}(H)$ for every finite run in $\mathcal{A}$. The strategy $\sigma$ induces a set of runs in $\mathcal{A}$, which is normally a subset of all runs generated by $\mathcal{A}$. It is only for these runs that we define $A_\sigma$. □

The second lemma is a direct consequence of Lemma **IV.1**.

**Lemma IV.2.** Given a strategy $\sigma$ for the game $\mathcal{A}$ and objective $Obj$, then $win_{A_\sigma} = val_\sigma(v_0)$, where $v_0$ is the initial vertex of $\mathcal{A}$ and $val_\sigma(v_0)$ is the value of winning the game with strategy $\sigma$. Conversely, given a complete and consistent attack function $A$, then $win_A = val_{\sigma_A}(v_0)$.

Finally, we are able to state the main theorem of the paper. The theorem ties the solution of the $1\frac{1}{2}$-player reachability graph-game defined by $\mathcal{A}$ and the set $Obj$ with the solution of Problem **III.1**.

**Theorem IV.1.** Consider the $1\frac{1}{2}$-player reachability graph-game defined by $\mathcal{A}$ and the set $Obj$. If $\sigma$ is an optimal strategy for the $1\frac{1}{2}$-player reachability graph-game, then $A_\sigma$ is a solution for Problem **III.1**.

## V. CONCLUSION

We have considered the problem of synthesis of attack functions for sensor deception attacks at the supervisory level of feedback control systems, where the system is modeled as a probabilistic finite-state automaton controlled by a given deterministic supervisor. Given the stochastic nature of the system model, attack functions were quantified by the likelihood of reaching the unsafe region. We leveraged techniques from supervisory control of discrete event systems to formulate the problem, and techniques from stochastic graph-games to solve it. In the future, it would be of interest to investigate defense measures for the supervisor to prevent such attacks from succeeding.

## REFERENCES


[1] L. K. Carvalho, Y. C. Wu, R. Kwong, and S. Lafortune. Detection and mitigation of classes of attacks in supervisory control systems. *Automatica*, 2018. To appear.

[2] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2008.

[3] K. Chatterjee and T. A. Henzinger. A survey of stochastic $\omega$-regular games. *Journal of Computer and System Sciences*, 78(2):394 – 413, 2012.

[4] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.

[5] A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[6] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Inc., Orlando, FL, USA, 1970.

[7] V. K. Garg, R. Kumar, and S. I. Marcus. A probabilistic language formalism for stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 44(2):280–293, Feb 1999.

[8] R. Kumar and V. K. Garg. Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, 46(4):593–606, Apr 2001.

[9] M. Lawford and W. M. Wonham. Supervisory control of probabilistic discrete event systems. In *Proceedings of 36th Midwest Symposium on Circuits and Systems*, pages 327–331, Aug 1993.

[10] P. M. Lima, L. K. Carvalho, and M. V. Moreira. Detectable and undetectable network attack security of cyber-physical systems. In *14th IFAC Workshop on Discrete Event Systems WODES 2018*, volume 51, pages 179 – 185, 2018.

[11] L. Lin, S. Thuijsman, Y. Zhu, S. Ware, R. Su, and M. Reniers. Synthesis of Successful Actuator Attackers on Supervisors. *ArXiv e-prints - http://arxiv.org/abs/1807.06720*, July 2018.

[12] R. Meira-Góes, E. Kang, R. Kwong, and S. Lafortune. Stealthy deception attacks for cyber-physical systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4224–4230, Dec 2017.

[13] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[14] R. Su. Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. *Automatica*, 94:35 – 44, 2018.

[15] D. Thorsley and D. Teneketzis. Intrusion detection in controlled discrete event systems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6047–6054, Dec 2006.