Resilience Against Sensor Deception Attacks at the Supervisory Control Layer of Cyber-Physical Systems: A Discrete Event Systems Approach

by

Rômulo Meira Góes

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2020

Doctoral Committee:

Professor Stéphane Lafortune, Chair
Assistant Professor Jean-Baptiste Jeannin
Professor Mingyan Liu
Associate Professor Necmiye Ozay
Professor Atul Prakash

Rômulo Meira Góes

romulo@umich.edu

ORCID iD: 0000-0003-3567-9685

ACKNOWLEDGMENTS

I recall in 2015 a "young me" writing about my PhD expectations... As expected by many of you, I was completely wrong about most of what I had written down. I could write an essay about all the conflicting emotions that I had during my PhD, but I will spare you from that. Instead, I sincerely express my gratitude to everyone that supported, encouraged, and inspired me; I could not thank you enough.

Almost like a motto for PhD applicants is: "Carefully choose your PhD advisor." I could not be luckier for having Stéphane Lafortune in this advisor role. His patience for my long-winded progress in the field astonishes me. He is a great mentor that taught me how to persevere in the pursuit of knowledge.

I thank Professor Necmiye Ozay for being an important mentor and teacher. She provided me insightful comments that help me throughout my graduate studies. I also would like to acknowledge my other committee members: Professor Mingyan Liu, Professor Jean-Baptiste Jeannin, and Professor Atul Prakash. In addition, I appreciate all the faculty at UM from whom I have learned a lot as well as the UM staff, specially EECS staff, that provide essential support to students.

My research collaborators are intrinsically part of this dissertation. Many thanks to Professor Raymond Kwong, Professor Karen Rudie, Professor Eunsuk Kang, Doctor Hervé Marchand, Doctor Christoforos Keroglou, Doctor Sahar Mohajerani, and Doctor Blake Rawlings for sharing their knowledge and criticisms with me. I specially appreciate Professor Karen Rudie and Doctor Hervé Marchand for their receptiveness during my stay at Queen's University and INRIA-Rennes. On that note, I would like to express my gratitude to the people involved in the Control of Discrete Event Systems group at Queen's University and the Sumo group at the INRIA-Rennes.

My time at UM would not be as joyful as it was without my friends and colleagues. Special

thanks to my colleagues at the UMDES group Xiang Yin, Yiding Ji, Andrew Wintenberg, and Shoma Matsui for insightful discussions about research. My friends Shurjo and Madan who always had their lab door open for the must needed research breaks. My running and adventure friends Ted and Jonas who were always down for long overnight drives so that we would arrive at the parks at dawn. I also have to mention Bernardo, Raquel, and Eurico, this Brazilian troupe brought sheer happiness that helped me cope with my homesickness. Of course, I express my great gratitude to my friends in Brazil. I thank my friend and roommate Wesley and our BTB cantina lunch group for our goofy and/or intellectual discussions.

Continuing on this personal note, I would like to thank my parents, Reinaldo and Solange, and my sisters, Suelen and Liz. My parents always valued education even though they had not had a chance to pursue it. Many times they sacrificed themselves so that my sisters and I would continue receiving our education. My sisters paved my path in the academic world since both were part of it way before I decided to step in it. I also want to thank my partner Rachel for being there to share my achievements and support my struggles.

Finally, I wish to acknowledge the financial support from NSF.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

AIDA All Insertion Deletion Attack

BSCP Basic Supervisory Control Problem

BAIDA Bounded All Insertion Deletion Attack

CPS Cyber-Physical Systems

DES Discrete Event Systems

DFA Deterministic Finite State Automaton

E-state Environment-state

IDA Insertion Deletion Attack

ISDA Interruptible Stealthy Deceptive Attack

MDP Markov Decision Processes

PFA Probabilistic Finite State Automaton

PTF Probabilistic Transition Function

SCT Supervisory Control Theory

SCP-AP Supervisory Control Problem with Arbitrary control Patterns

S-state Supervisor-state

USDA Unbounded Stealthy Deceptive Attack

LIST OF SYMBOLS

Σ Set of Events

G Plant - DFA

$\mathcal{L}(G)$ Language of DFA G

$En_G(x)$ Set of events enabled at state x in DFA G

$Ac(G, X)$ Accessible part of G after deleting states X

$P_{\Sigma_l \Sigma_s}$ Natural projection from events Σ_l to events in Σ_s

Σ_{ctr} Set of Controllable Events

Σ_{obs} Set of Observable Events

Γ Set of feasible control decision - $\Gamma = \{\gamma \in 2^{\Sigma} \mid \gamma \subseteq \Sigma_{uctr}\}$

R Supervisor - DFA

Σ_{sen} Set of Events Compromised by the Attacker

Σ_{ins} Set of Insertion Events - $\Sigma_{ins} = \{ins(e) \mid e \in \Sigma_{sen}\}$

Σ_{del} Set of Deletion Events - $\Sigma_{del} = \{ins(e) \mid e \in \Sigma_{sen}\}$

Σ_{att} Set of Attacker Events - $\Sigma_{att} = \Sigma_{ins} \cup \Sigma_{del}$

Σ_{all} Set of All Events in the Attacked Plant - $\Sigma_{all} = \Sigma \cup \Sigma_{att}$

P^S Projection operator that describes how the supervisor observes Σ_{all} events

P^G Projection operator that describes how the plant executes Σ_{all} events

G_a Attacked Plant - DFA - Def. **3.3**

R_a Attacked Supervisor - DFA - Def. **3.4**

A Automaton that encodes an attack function

M Markov Decision Process

Π_M The set of all deterministic MDP policies of M

$\mathcal{A}$ Graph-game arena

ABSTRACT

Cyber-Physical Systems (CPS) are already ubiquitous in our society and include medical devices, (semi-)autonomous vehicles, and smart grids. However, their security aspects were only recently incorporated into their design process, mainly in response to catastrophic incidents caused by cyber-attacks on CPS. The Stuxnet attack that successfully damaged a nuclear facility, the Maroochy water breach that released millions of gallons of untreated water, the assault on power plants in Brazil that disrupted the distribution of energy in many cities, and the intrusion demonstration that stopped the engine of a 2014 Jeep Cherokee in the middle of a highway are examples of well-publicized cyber-attacks on CPS. There is now a critical need to provide techniques for analyzing the behavior of CPS while under attack and to synthesize attack-resilient CPS. In this dissertation, we address CPS under the influence of an important class of attacks called sensor deception attacks, in which an attacker hijacks sensor readings to inflict damage to CPS.

The formalism of regular languages and their finite-state automata representations is used to capture the dynamics of CPS and their attackers, thereby allowing us to leverage the theory of supervisory control of discrete event systems to pose our investigations. First, we focus on developing a supervisory control framework under sensor deception attacks. We focus on two questions: (1) Can we automatically find sensor deception attacks that damage a given CPS? and (2) Can we design a secure-by-construction CPS against sensor deception attacks? Answering these two questions is the main contribution of this dissertation.

In the first part of the dissertation, using techniques from the fields of graph games and Markov decision processes, we develop algorithms for synthesizing sensor deception attacks in both qualitative and quantitative settings. Graph games provide the means of synthesizing sensor deception attacks that might damage the given CPS. In a second step, equipped with stochastic information

about the CPS, we can leverage Markov decision processes to synthesize attacks with the highest likelihood of damage.

In the second part of the dissertation, we tackle the problem of designing secure-by-construction CPS. We provide two different methodologies to design such CPS, in which there exists a trade-off between flexibility on selecting different designs and computational complexity of the methods. The first method is developed based on supervisory control theory, and it provides a computationally efficient way of designing secure CPS. Alternatively, a graph-game method is presented as a second solution for this investigated problem. The graph-game method grants flexible selection of the CPS at the cost of computational complexity. The first method finds *one* robust supervisor, whereas the second method provides a structure in which *all* robust supervisors are included.

Overall, this dissertation provides a comprehensive set of algorithmic techniques to analyze and mitigate sensor deception attacks at the supervisory layer of cyber-physical control systems.

CHAPTER I
Introduction

1.1 Motivation

The growth of successful cyber-attacks on key elements of our society's infrastructure has recently become a concern, especially to engineers. The Stuxnet attack that successfully caused damage to a nuclear facility [1], the Maroochy water breach that released millions of gallons of untreated water [2], the assault on power plants in Brazil that disrupted the distribution of energy in many Brazilian cities [3], and the intrusion demonstration that stopped the engine of the 2014 Jeep Cherokee in the middle of a highway [4] are examples of well-publicized cyber-attacks on physical infrastructure. All of these examples share a similar feature: they are modern engineering systems composed of the interaction of physical elements - such as power plants, vehicles, or medical devices - with a computational infrastructure that controls them. These systems are called *cyber-physical systems* (CPS).

CPS are found across diverse areas of society, from power plants to transportation systems, from smart homes to medical devices. These systems must be reliable, robust and secure against both benign malfunction as well as malicious planned attacks. However, the *security* aspects of their design have often been treated as an afterthought; and this approach has revealed its flaws considering the number of successful attacks our society has experienced. According to the Council of Economic Advisers, cyber-attacks were estimated to have cost between $\$57$ billion and $\$109$ billion to the U.S. economy in 2016 [5]. There is now an urgency to develop techniques for understanding and designing attack-resilient CPS. These techniques will need to take into account diverse areas of application and *ad hoc* approaches to their development should be avoided.

This dissertation proposes to tackle these problems by developing a novel methodology for understanding a large class of CPS attacks and then designing attack-resilient CPS. This methodology has the following features: (i) it considers and handles systems with imperfect information (due to sensor limitations) and limited control of actuators (due to system disturbances) (ii) it models sensor deception attacks, an important class of attacks for CPS, one in which an attacker hijacks the information sent to the computational infrastructure; (iii) it considers the attacker's perspective by providing methods to synthesize successful attacks for fixed controllers; (iv) it provides a formal model-based approach for designing attack-resilient controllers; (v) the methods are general and applicable to a large class of CPS, specifically systems with safety-critical requirements whose behavior is suitably model as event-driven systems (e.g., power plants, autonomous vehicles, and transportation systems).

1.2 Overview of attacks in cyber-physical systems

In control engineering, the area of discrete event systems (DES) is known for its ability in modeling the high-level behavior of complex systems. The operation of these systems is modeled in an event-based manner and only important details are considered in analyzing the system's operation. This dissertation uses *supervisory control theory* (SCT) from the field of discrete event systems to analyze CPS. We assume that CPS has been abstracted into an even-driven model, where we employ our methodology.

Supervisory control provides a formal model-based framework for designing correct-by-construction controllers for complex event-driven systems. Despite these benefits, this framework does not encompass security aspects. We do a brief literature review to show some of the existing related works on cyber-security of CPS in the field of DES. A more comprehensive, although incomplete, literature review is described in [6].

1.2.1 Modeling attacks

Previously, some efforts were made in classification and modeling of cyber-attacks assuming certain intelligence on the part of the attacker [7, 8]. Our focus is on a special type of attacks called *deception attacks*. These attacks are characterized by some type of manipulation of the sensor measurements received by the supervisor and/or of the actuator commands sent by the supervisor. Figure 1.1 pictorially defines the concept of deception attacks, where an attacker compromises the communication channels between the supervisor and the plant.

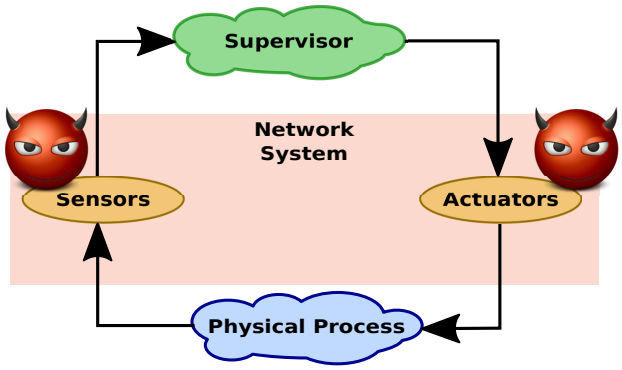


Figure 1.1: Concept of deception attacks on CPS

Deception attacks were first introduced in the DES field by Carvalho et al. [9], where they define and study actuator enablement deception attacks. This class of attacks is literally defined as its name; the attacker *enables* actuator commands that were previously disabled by the supervisor. Their attack model is intertwined with the physical dynamics, which prevents a clear analysis of the attacker's strategies. In this manner, the attacker is not pictured as a "smart" agent by this work. Later on, the authors extended their work to encompass full deception attacks [10, 11, 12], i.e., both sensor and actuator deception attacks. However, they followed similar modeling techniques.

The works of Su and Wakaiki et al. in [13, 14] introduced a more sophisticated model for sensor deception attacks. An attacker is seen as an agent that reacts to sensor readings, differently than the model presented in [11], where the attacker is a passive entity.

In [13], an attacker is modeled by an input-output automaton that has input and output languages. Namely, the input language is the language generated by the sensor readings while the

output language is the language generated by the attacker to confuse the supervisor. In this model, three key assumptions are made: the attacker can only modify a subset of the sensor readings; modifications are bounded; and the attacker finishes its modifications before the physical process executes a new event.

On the other hand, Wakaiki et al. models sensor deception attacks as a function instead of input-output automaton [14]. This function has languages as domain and co-domain in a similar manner as the input and output languages in [13]. However, Wakaiki et al. impose different constraints to their attack model compared to the ones in [13], which leads to a different closed-loop behavior.

Recently, the work in [15] modeled sensor and actuator deception attacks as input-output automaton similar as [13]. Differently than [13], they do not impose any bound constraint over the attacker. However, their supervisory control framework differs from the standard framework. They assume that the supervisor can *actively* change the state of the physical process. In the standard supervisory control framework, the supervisor cannot change the state of the plant.

Actuator deception attacks were introduced by [16, 17, 18]. The models of actuator attacks in these works are similar to the previously mentioned methodologies for sensor deceptions attacks.

1.2.2 Intrusion detection of deception attacks

Detecting and mitigating deception attacks become an essential part of CPS. For this reason, CPS are enhanced with intrusion detection modules that are constructed to detect specific types of attacks. These modules provide additional information for supervisors in order to mitigate attacks [12, 19].

Intrusion detection of deception attacks has been studied in DES since the work in [20]. In [20], the authors focus on deciding when it is possible to detect actuator deception attacks such that a supervisor can satisfy a specification both in normal operation and after an attack.

Similarly, the work in [9] focuses on actuator deception attacks. It provides conditions for the detection and mitigation of these attacks. The work of [9] is closest to the work in [21], where after detecting an attack the system reconfigures the control law in order to mitigate it. Extension

of their work is found in [10, 11, 12], where it fully encompasses deception attacks. Recently, a stochastic notion of intrusion detection is considered in [22].

1.2.3 Synthesis of deception attacks

Intrusion detection modules are not always capable of detecting deception attacks. Therefore, considering the attacker's perspective provides a way of characterizing vulnerabilities in feedback control systems. In other words, studying systematic methods to synthesize deception attacks that are undetectable by intrusion detection modules is of great importance.

The work in [13] studied synthesis of bounded sensor deception stealthy attack strategies. They are considered stealthy since they are not detected by a given detection mechanism. However, [13] only focuses on bounded sensor deception attacks.

The work of [23] also investigated the synthesis of sensor deception stealthy attacks. Nevertheless, they studied this problem in an open-loop context instead of the closed-loop context shown in Fig. 1.1.

Similarly, the work in [16] investigated the synthesis of actuator enablement attack strategies. In that work, the authors assume that actuator events are observable, which simplifies the problem.

1.2.4 Synthesis of supervisors resilient against deception attacks

Both of the previous topics assumed that a feedback control system already exists and it is fixed. However, if the feedback control system is being designed, i.e., it was not implemented, then could we design a feedback control system robust against deception attacks? Namely, we wish to design a feedback control system that does not necessarily need an intrusion detection module since it is provably robust by design.

In [14, 13], authors studied the synthesis of robust supervisors against sensor deception attacks. Wakaiki et al. provide necessary and sufficient conditions for the existence of a supervisor that exactly achieves a specification under sensor deception attacks [14]. On the other hand, Su describes a synthesis procedure that obtains the supremal normal sublanguage robust against sen-

sor deception attacks [13]. However, the solution provided by [13] only calculates the supremal normal sublanguage and it has triple exponential complexity.

The works of [18, 17] analyze the synthesis of supervisors robust against actuator attacks. In [17], the robustness property is enforced using obfuscation methods to enforce robustness. The system output is obfuscated so as to deceive the attacker. Other cybersecurity problems, such as the opacity enforcement problem [24], successfully employed this obfuscation technique. The work of [18] enforces robustness against bounded actuator attacks using supervisory control techniques.

1.3 Organization and main contributions

The main contributions and the organization of this dissertation are summarized as follows.

Chapter II: Supervisory control theory

This chapter reviews basic notions and operations in DES such as: deterministic finite automata, parallel composition and observer automaton. We also describe the standard supervisory control framework.

Chapter III: Supervisory control theory under sensor deception attacks [25, 26, 27, 28]

In this chapter, we first provide a general model for sensor deception attacks in the supervisory control framework. Sensor readings in DES are modeled by possible sequences of events over an "alphabet", i.e., a language. Thus, a sensor deception attack is defined as the *manipulation* of these sequences of events. This attack model is general and applicable to different attack goals, such as "reach an unsafe state" or "denial of service".

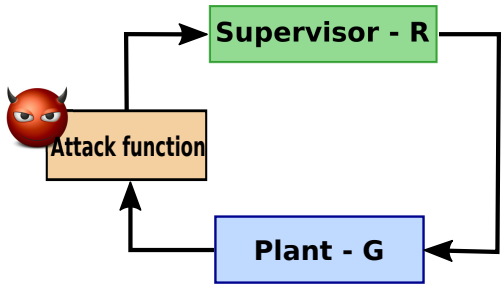


Figure 1.2: Sensor deception attacks in the supervisory control framework

The second task of this chapter is to include this attack model into the supervisory control framework, as depicted in Fig. 1.2. Namely, we provide a way to analyze the closed-loop behavior of the controlled system under sensor deception attack. In other words, we precisely capture the language generated by the controlled system under attack. The subsequent chapters are developed based on this newly described supervisory control framework.

Chapter IV: Synthesis of stealthy sensor deception attacks for supervisory control [25, 26]

Many complex systems already have existing supervisors in place; however, these supervisors were designed without taking into account sensor deception attacks. Our main goal in this chapter is to answer three questions: When does a sensor deception attack cause damage to the system? How does this attack cause damage? Could this attack hide its actions? These questions enlighten the vulnerabilities of the closed-loop system.

To answer these questions, we put ourselves into the attacker's shoes. The attacker searches for ways to cause damage to the plant while concealing its actions. Namely, the behavior *generated* by the controlled system under attack can reach the critical region (damage) of the uncontrolled system, as shown in Figure 1.3. The controlled behavior without attacks provides the baseline for the stealthiness of the attacker. As long as the system *observed* behavior is within this baseline, the attacker conceals its presence, i.e., the controlled system seems to be operating in normal conditions.

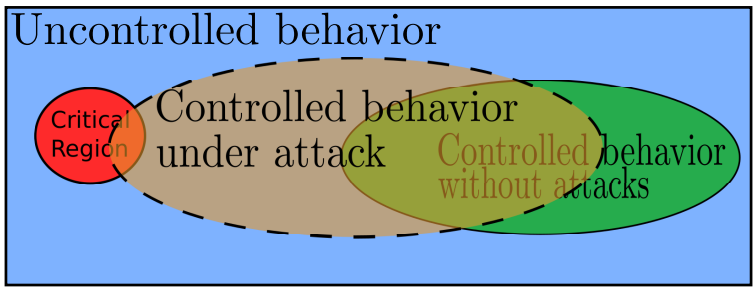


Figure 1.3: Controlled system under attack

We study three specific types of attacks that are based on the interaction between the attacker and the controlled system. The methodology developed to synthesize these attacks is inspired by the work in [24, 29, 30]. As in these works, we employ a discrete structure to model the

game-like interaction between the supervisor and the environment. This game-theoretical approach provides a structure for each attack class that incorporates *all successful stealthy attacks*. Different stealthy attack strategies can be extracted from these structure. By providing a general synthesis framework, our goal is to allow CPS engineers to detect and address potential vulnerabilities in their control systems.

Chapter V: Synthesis of optimal sensor deception attacks for stochastic supervisory control [31, 32]

Chapter IV provides qualitative results about synthesizing attack strategies, i.e., an attack strategy is either successful or not. In this chapter, we investigate synthesizing sensor deception attacks in a quantitative manner, assuming a stochastic controlled system. The stochasticity of the controlled system allows a quantitative analysis of the attack strategies. This gives rise to a broader class of attack strategies, as compared with our previous results.

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 an unsafe state of the uncontrolled plant. First, we investigate the synthesis of an *attack function* that generates the maximum likelihood of reaching an unsafe state. The second problem investigated is the *synthesis of attack functions that satisfy multiple objectives* (multi-objective). Namely, the attack function must reach an unsafe state with maximum probability while minimizing a cost function based on the attacker sensor modifications.

Our solution methodology employs results from the area of stochastic control systems, more specifically Markov Decision Processes (MDPs). First, we show how to build the "right" MDP that captures the interaction of the attacker and the control system. Next, we show that the solution of the two investigated problems is reducible to known problems in the MDP literature.

Chapter VI: Synthesis of supervisors robust against sensor deception attacks [27, 28]

The previous two chapters focus on the synthesis of sensor deception attacks. They consider the attacker's perspective given a fixed and known supervisor. Although the results in the previous chapters show the possible vulnerabilities of closed-loop systems, they do not provide means of "fixing" them. To "fix" any vulnerability, we must redirect our attention from the attacker to the

supervisor. In Chapter VI, we consider the "dual" problem of synthesizing a supervisor that will be robust against sensor deception attacks; thus, this work focuses on *defense* strategies. Pictorially, the behavior generated by the feedback control system under attack should never intersect the critical region, as shown in Figure 1.4.

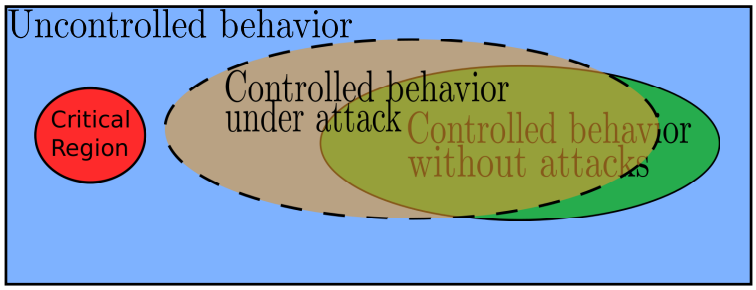


Figure 1.4: Robust controlled system under attack

We present two different methodologies to solve this problem: one uses graph-game methods together with supervisory control methods whereas the other uses only supervisory control methods. There exists a trade-off between these two methodologies. While the method relying on supervisory control techniques is computationally more efficient than the graph-game technique, it does not provide flexibility on the selection of the robust supervisor. This flexibility on choosing a robust supervisor is feasible with the graph-game technique but the price for it is a more computationally "expensive" method.

Chapter VII: Conclusion

In this chapter, we summarize the contributions of this dissertation. Moreover, we present promising directions for future work.

CHAPTER II

Supervisory control theory

2.1 Automata models and relevant operations

2.1.1 Automata models

To study the behavior of the DES, we consider the theories of languages and automata. We consider that the system under examination is described by a language over a set of events. Each event define a specific "change" of this system and the concatenation of events, a string, specifies how the system evolves. For example, consider a simple road intersection where we define the events as C_{int} to denote that a car entered the intersection and C_{out} the exiting of the car. A possible string is $C_{int}C_{out}$, where a car entered and exited the intersection.

Formally, let Σ be a finite set of events. A language is defined as a *finite* set of strings over Σ. The symbol ϵ expresses the empty string, i.e., a string with no event. We denote by Σ^* the set of all finite strings over Σ. Therefore, a language L over Σ is a subset of Σ^*, i.e., $L \subseteq \Sigma^*$.

For any string $s \in \Sigma^*$, we use the following notation. The prefix closure of s is the set $pre(s) := \{t \in \Sigma^* : \exists u \in \Sigma^*.tu = s\}$. With an abuse of notation, we use $pre(L)$ to denote the prefix closure of language $L \subseteq \Sigma^*$. We denote by $s[i]$ the i^{th} event of s such that $s := s[1]s[2]\dots s[|s|]$, where $|s|$ denotes the length of s. We denote by s^i the i^{th} prefix of s, namely $s^i := s[1]\dots s[i]$ 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 $n \in \mathbb{N}$.

Although languages, as defined, can describe systems of interest, it is cumbersome to easily define and manipulate them. For this reason, we use the modeling formalism of automata to rep-

resent and manipulate languages. This assumption is not without loss of generality since there exists languages that *cannot* be represented as automata [33, 34]. Languages that can be represented as automata are denoted as *regular languages* [33]. Hereafter, the system of interest, e.g., cyber-physical systems, is modeled as a deterministic finite-state automaton.

Definition 2.1. *A deterministic finite-state automaton (DFA) G is defined as a tuple*

$$G := (X_G, \Sigma_G, \delta_G, x_{0,G})$$

where X_G is the finite set of states; Σ_G is the finite set of events; $\delta_G : X_G \times \Sigma_G \to X_G$ is the partial transition function; $x_{0,G} \in X_G$ is the initial state.

Remark 2.1. *Definition 2.1 differs from the standard definition of DFA [33]. We do not define the set of final states as in the standard DFA definition. Every state in G is final, which limits our DFA definition to only describe prefix-closed regular languages. In DES, the set of final states describes liveness properties, which are not used in this dissertation. Furthermore, we allow the transition function δ_G to be partially defined (incomplete).*

The function δ_G is extended in the usual manner to domain $X_G \times \Sigma_G^*$. The language generated by G is defined as $\mathcal{L}(G) = \{s \in \Sigma_G^* | \delta_G(x_{0,G}, s)!\}$, where ! means "is defined". The enabled function $En_G : X_G \to \Sigma_G$ is defined as the set of events enabled at a given state in X_G.

$$En_G(x) := \{e \in \Sigma_G \mid \delta_G(x, e)!\} \tag{2.1}$$

This function is extended to a set of state $X \subseteq X_G$, i.e., $En_G(x) := \cup_{x \in X} En_G(x)$.

By an abuse of notation, the function δ_G is also extended to domain $2^{X_G} \times \Sigma_G$.

$$\delta_G(X, e) := \begin{cases} \bigcup_{x \in X} \delta_G(x, e) & \text{if } e \in En_G(X) \\ \emptyset & \text{otherwise} \end{cases} \tag{2.2}$$

Example 2.1. *Figure 2.1 depicts an automaton with 9 states and 4 events. The set of events is $\Sigma_G =$*

$\{B_{int}, B_{out}, R_{int}, R_{out}\}$. *We also provide examples for the operators* En_G *and* δ_G*:* $En_G(1) = \{B_{int}, R_{int}\}$ *and* $\delta_G(\{1,2\}, R_{int}) = \{4,5\}$.

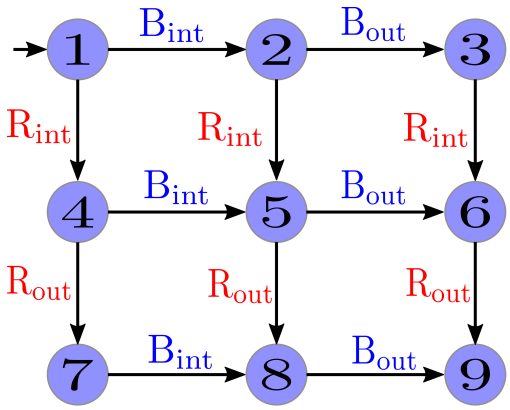


Figure 2.1: Automaton G

2.1.2 Accessible part

The accessible procedure removes states from G that are unreachable from $x_{0,G}$. Namely, it produces an automaton where every state in it is *accessible*/reachable from $x_{0,G}$, i.e., there is a directed path from $x_{0,G}$ to every state. Namely, it removes states from $x \in X_G$ such that there does not exist $s \in \Sigma_G^*$ such that $\delta_G(x_{0,G}, s) = x$. This operation does not alter the language of the automaton of interest as we can note from the definition of $\mathcal{L}(G)$.

We generalize the accessible procedure to remove states from a given set $X \subseteq X_G$. Namely, the generalized accessible procedure removes states X and computes the accessible part of the remaining states in G.

Definition 2.2. *Given* G *and* $X \subseteq X_G$*, we denote by* $Ac(G, X)$ *to be automaton with the accessible part of* G *after deleting* X*. Formally,* $Ac(G, X) := (X_{ac}, \Sigma_G, \delta_{ac}, x_{0,G})$ *where*

$$X_{ac} = \{x \in X_G \setminus X \mid \exists s \in \Sigma_G^*.\ \delta_G(x_{0,G}, s) = x \wedge \forall i \in [|s|].\ \delta_G(x_{0,G}, s^i) \notin X\}$$

$$\delta_{ac} = \delta_G \mid_{X_{ac} \times \Sigma_G \to X_{ac}}$$

The notation $Ac(G)$ is used when $X = \emptyset$.

Example 2.2. *Based on Fig. 2.1, Figure 2.2 depicts automaton $Ac(G, \{2, 5\})$.*

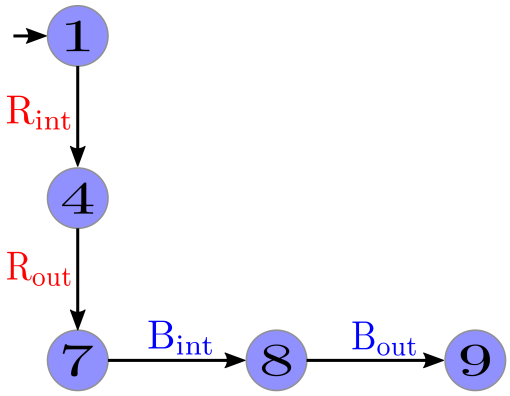


Figure 2.2: $Ac(G, \{2, 5\})$

2.1.3 Parallel composition

Parallel composition describes the interconnectivity of a set of automata. It describes the joint behavior of automata that operate concurrently. The coupling is performed based on the set of events of each automaton. Events that only belong to one automaton are ignored by the other automata, i.e., they executed asynchronously. On the other hand, events that belong to more than one automaton must be executed simultaneously. In summary, "shared" events occur synchronously while "private" events act asynchronously.

Definition 2.3. *Given automata G_1, G_2, the parallel composition of G_1 and G_2 is the automaton $G_1||G_2 := Ac((X_{G_1} \times X_{G_2}, \Sigma_{G_1} \cup \Sigma_{G_2}, \delta_{G_1||G_2}, (x_{0,G_1}, x_{0,G_2})))$ where*

$$\delta_{G_1||G_2}((x_1, x_2), e) := \begin{cases} (\delta_{G_1}(x_1, e), \delta_{G_2}(x_2, e)) & \text{if } e \in En_{G_1}(x_1) \cap En_{G_2}(x_2) \\ (\delta_{G_1}(x_1, e), x_2) & \text{if } e \in En_{G_1}(x_1) \setminus \Sigma_{G_2} \\ (x_1, \delta_{G_2}(x_2, e)) & \text{if } e \in En_{G_2}(x_2) \setminus \Sigma_{G_1} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

2.1.4 Projection map

There might be events in the system of interest that cannot be "sensed/observed" due to limited sensing capabilities. For example, the event C_{out} could be unobservable if there is no sensor to observe a car exiting the intersection. For this reason, we define a *projection* map for strings that projects strings from a larger set of events, Σ_l, to a smaller set of events, Σ_s. Intuitively, the projection map erases the events *not in* Σ_s from a given string.

Definition 2.4. *Given the set of events Σ_l and $\Sigma_s \subseteq \Sigma_l$, the projection function $P_{\Sigma_l\Sigma_s} : \Sigma_l^* \to \Sigma_s^*$ is defined as:*

$$P_{\Sigma_l\Sigma_s}(\epsilon) := \epsilon$$

$$P_{\Sigma_l\Sigma_s}(e) := \begin{cases} e & \text{if } e \in \Sigma_s \\ \epsilon & \text{if } e \in \Sigma_l \setminus \Sigma_s \end{cases}$$

$$P_{\Sigma_l\Sigma_s}(se) := P_{\Sigma_l\Sigma_s}(s)P_{\Sigma_l\Sigma_s}(e)$$

The inverse projection $P^{-1}_{\Sigma_l\Sigma_s} : \Sigma_s^* \to 2^{\Sigma_l^*}$ is defined as $P^{-1}_{\Sigma_l\Sigma_s}(t) := \{s \in \Sigma_l^* | P_{\Sigma_l\Sigma_s}(s) = t\}$. Moreover, $P_{\Sigma_l\Sigma_s}$ and $P^{-1}_{\Sigma_l\Sigma_s}$ are extended to languages by simply applying the projection map for each string in the language.

2.1.5 Observer

As we mentioned, events in the system of interest might not be observed due to limited sensing capabilities. We characterize this limited sensing by partitioning the event set into the set of observable events, denoted by Σ_{obs}, and the set of unobservable events, denoted by Σ_{uobs}. In other words, our system of interest is modeled by automaton G whose set of events $\Sigma_G = \Sigma_{obs} \cup \Sigma_{uobs}$ and $\Sigma_{obs} \cap \Sigma_{uobs} = \emptyset$.

The observer automaton of G characterizes the projected language of G with respect to Σ_G and Σ_{obs}, i.e., $\mathcal{L}(Obs(G)) = P_{\Sigma_G\Sigma_{obs}}(\mathcal{L}(G))$. To define the observer automaton, we need to define the

unobservable reach function (UR). The unobservable reach of a set of state $X \subseteq X_G$ is the set of all states that can be reached from X via unobservable strings. In words, if we start from any state in X, then we could be in any state of $UR(X)$ via unobservable strings.

$$UR(X) := \{x \in X_G \mid (\exists y \in X)(\exists s \in \Sigma_{uobs}).\delta_G(y, s) = x\}$$

Definition 2.5. *Given automaton G with observable events $\Sigma_{obs} \subseteq \Sigma_G$, the observer automaton of G is $Obs(G) := Ac((2^{X_G}, \Sigma_{obs}, \delta_{Obs(G)}, UR(\{x_{0,G}\}))$ where*

$$\delta_{Obs(G)}(X, e) := \begin{cases} UR(\delta_G(X, e)) & \text{if } e \in En_G(X) \cap \Sigma_{obs} \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{2.3}$$

Note that if we start from any state in X, then $\delta_{Obs(G)}(X, e)$ defines the set of states that G could be, after it observes event e. In words, the set of state where G could be, given what we have observed, i.e., observable event.

Example 2.3. *Let us provide an example of the Observer automaton using the automaton shown in Fig. 2.1. We assume that $\Sigma_{obs} = \{R_{int}, R_{out}\}$. The observer automaton is shown in Fig. 2.3.*

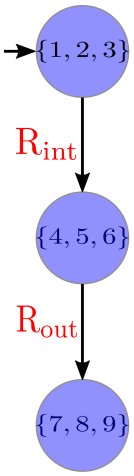


Figure 2.3: Observer automaton

2.2 Supervisory control theory

We consider the supervisory layer of a feedback control system, as depicted in Fig. 2.4, where the uncontrolled system (plant) is modeled as a DFA in the discrete-event modeling formalism. Namely, the plant is modeled by automaton $G = (X_G, \Sigma, \delta_G, x_{0,G})$[1]. Language $\mathcal{L}(G)$ is considered as the uncontrolled system behavior since it includes all possible executions of G.

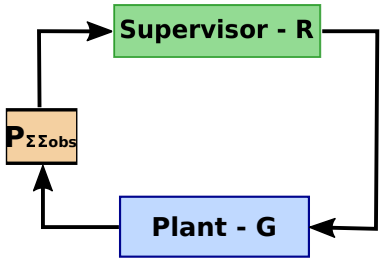


Figure 2.4: Supervisory control framework

In the context of the supervisory control theory (SCT) of DES [35], system G is considered as the plant that needs to be controlled in order to satisfy given specifications. In this framework, the plant G is controlled by a *supervisor* that dynamically enables/disables events to satisfy the given specification. Intuitively, it is assumed that the supervisor observes the events generated by the plant and decides which events are allowed to be executed next.

In the intersection example, the supervisor receives strings such as $C_{int}C_{out}$. Assume that it is unsafe to allow two cars in the intersection at the same time, i.e., string $C_{int}C_{int}$ is unsafe. Therefore, a supervisor would disable event C_{int} to occur until it knows the current car in the intersection has left (C_{out}). The controlled language for this system resembles a intersection with a stop sign where C_{int} events are immediately followed by C_{out} events, e.g., $C_{int}C_{out}C_{int}C_{out}$.

Due to lack of actuation over G, it might not be possible to disable some events in Σ. This problem is addressed by partitioning Σ into the set of controllable events and the set of uncontrollable events, Σ_{ctr} and Σ_{uctr}, respectively. Therefore, the set of admissible control decisions is

[1]We have dropped the subscript G from the set of events of G. Hereafter, Σ is the set of events of plant G unless we state differently.

defined as:

$$\Gamma = \{\gamma \subseteq \Sigma \mid \Sigma_{uctr} \subseteq \gamma\}$$

Admissibility guarantees that a control decision will never disable an uncontrollable event.

In addition, when the system is partially observed due to limited sensing capabilities of G, the event set is also partitioned into $\Sigma = \Sigma_{obs} \cup \Sigma_{uobs}$ as discussed in the previous section. In this case, the supervisor only takes its actions after receiving an observable event. For this reason, Fig. 2.4 depicts the projection map $P_{\Sigma\Sigma_{obs}}$.

Formally, a partially observation supervisor is a (partial) function

$$S : \Sigma_{obs}^* \to \Gamma$$

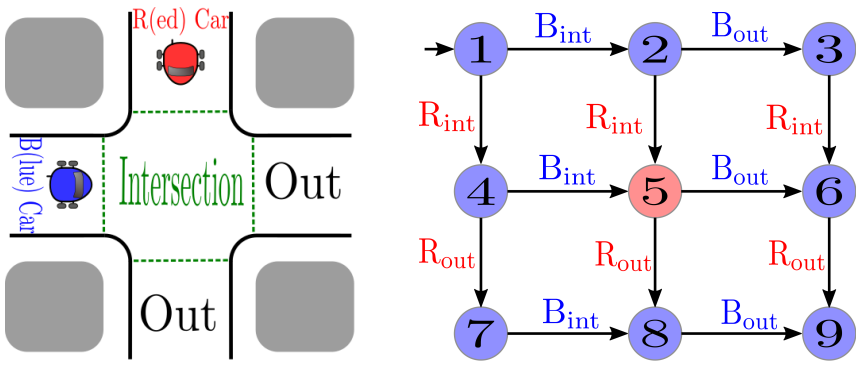
The resulting controlled behavior is a new DES denoted by S/G, resulting in the closed-loop language $\mathcal{L}(S/G)$. Basically, S/G defines feasible strings of G allowed by the supervisor S. This closed-loop language is defined recursively as follows:

$$\epsilon \in \mathcal{L}(S/G)$$

$$s \in \mathcal{L}(S/G) \text{ and } se \in \mathcal{L}(G) \text{ and } e \in S(P_{\Sigma\Sigma_{obs}}(s)) \Leftrightarrow se \in \mathcal{L}(S/G)$$

Normally, a supervisor S is encoded by an automaton R known as the supervisor realization, where every state encodes a control decision, see, e.g., [34]. Throughout our work, we use interchangeably supervisor S and its realization R. Based on the supervisor realization R, the closed-loop language can be easily obtained by parallel composing R and G, i.e., $\mathcal{L}(R/G) = \mathcal{L}(R||G)$.

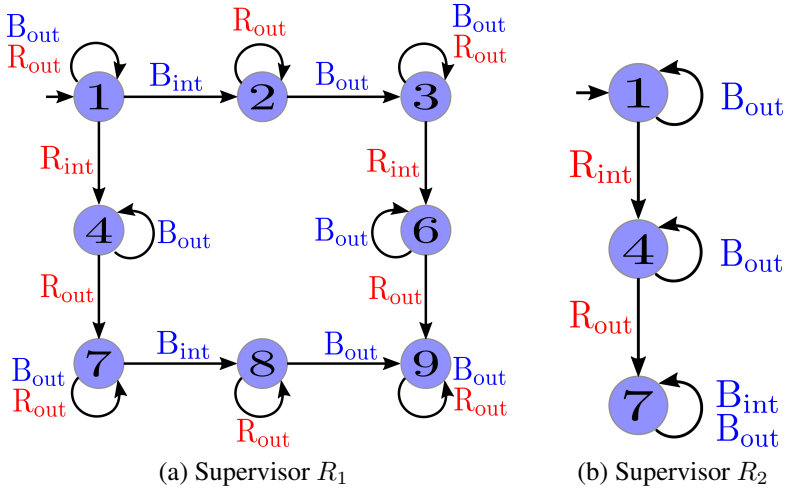
Example 2.4. *We use the collision avoidance problem of vehicles at an intersection as running example throughout this thesis. We have two roads with one car in each road approaching an intersection as in Figure 2.5(a). The cars must cross the intersection without colliding with each other, or equivalently, both cannot be at the intersection at the same time. This system is modeled by the automaton G shown in Fig. 2.5(b).*



(a) Intersection (b) Intersection model

Figure 2.5: Intersection model

First, let us assume that every event is observable but the supervisor can only control when a car enters the intersection, i.e., $\Sigma = \Sigma_{obs}$ *and* $\Sigma_{ctr} = \{B_{int}, R_{int}\}$*. The supervisor realization* R_1*, depicted in Fig. 2.6(a), guarantees that the specification is met. Next, we consider that* $\Sigma_{ctr} = \{R_{int}, B_{int}\}$ *and* $\Sigma_{obs} = \{R_{int}, R_{out}\}$*. A supervisor realization* R_2 *that guarantees the specification is depicted in Fig. 2.6(b).*



(a) Supervisor R_1 (b) Supervisor R_2

Figure 2.6: Supervisors

The closed-loop language $\mathcal{L}(R_1/G)$ *is easily defined by inspection since* R_1 *is a copy of* G *without state* 5*. Therefore,* $\mathcal{L}(R_1/G)$ *is all strings in* G *that do no reach state* 5*. On the other hand, the closed-loop language* $\mathcal{L}(R_2/G)$ *is more restrictive than* $\mathcal{L}(R_1/G)$*. This less permissiveness is the price paid by the partial observation scenario we have assumed.* $\mathcal{L}(R_2/G)$ *is described by the*

automaton in Fig. 2.7.

We also recall the notions of controllability, observability, and normality for a prefix-closed language $K \subseteq \mathcal{L}(G)$. These are properties that the closed-loop behavior must satisfy. For example, controllability ensures that the supervisor does not disable uncontrollable events while observability guarantees that the supervisor takes consistent control decision with respect to its observation. Normality is a stronger property that ensures observability. We say the language K is

- controllable w.r.t. to Σ_{ctr}, if $K\Sigma_{uctr} \cap \mathcal{L}(G) \subseteq K$;
- observable w.r.t. to Σ_{obs} and Σ_{ctr}, if $(\forall s \in K, \forall e \in \Sigma_{ctr} : se \in K)[P^{-1}_{\Sigma\Sigma_{obs}}(P_{\Sigma\Sigma_{obs}}(s))e \cap \mathcal{L}(G) \subseteq K]$;
- normal w.r.t. to Σ_{obs} and Σ_{ctr}, if $K = P^{-1}_{\Sigma\Sigma_{obs}}(P_{\Sigma\Sigma_{obs}}(K)) \cap \mathcal{L}(G)$.

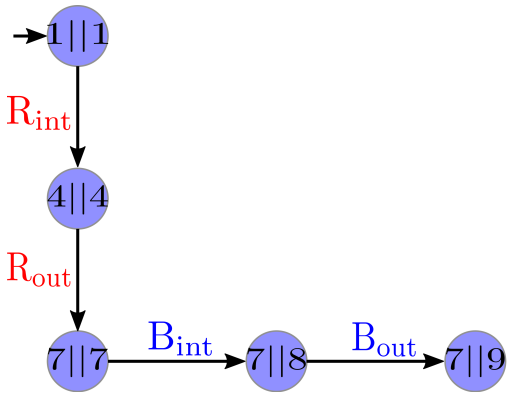


Figure 2.7: $R_2||G$

We complete this section by defining two useful functions. The unobservable reach of a set of states $X \subseteq X_G$ under the subset of events $\gamma \subseteq \Gamma$ is given by:

$$UR_\gamma(X) := \bigcup_{s \in (\Sigma_{uobs} \cap \gamma)^*} \delta_G(X, s) \tag{2.4}$$

Given $L \subseteq \mathcal{L}(G)$, the set of all possible states in G reachable from its initial state via a string with

the same projection as $s \in L$ is given by:

$$Re_G(s, L) := \bigcup_{\substack{t \in L: \\ P_{\Sigma\Sigma_{obs}}(s) = P_{\Sigma\Sigma_{obs}}(t)}} \delta_G(\{x_{0,G}\}, t) \tag{2.5}$$

CHAPTER III

Supervisory control under sensor deception attacks

3.1 Introduction

In our intersection example, the supervisor R guarantees that the two cars do not collide by functioning similarly as a stop sign. If the red car enters the intersection (R_{int}) before the blue car, then the blue car must wait until the intersection is empty. However, if for an unknown reason the supervisor does not receive the event R_{int}, then it believes that the intersection is empty and allows the blue car to enter as well. It suffices for one incorrectly transmitted event to cause a car collision when this supervisor R is deployed. The framework described in Chapter II does not allow analysis of the closed-loop system under unreliable conditions since it was not defined as such.

Although these unreliable conditions can be originated by different aspects, e.g., unreliable communication, unreliable models, our work focuses specifically on adversarial sensor communication between the plant and the supervisor as depicted in Fig. 3.1. An attacker tampers the event communication between the plant and the supervisor. This type of attacks is known as *sensor deception attacks*. In this chapter, we enhance the supervisory control framework to include sensor deception attacks.

First, we provide a general model for sensor deception attacks in the SCT. Sensor readings in DES are modeled as strings of events; and thus a sensor deception attack is modeled by string *editions*. Intuitively, an *edit* function performs these editions based on its memory and the events executed by the plant. This attack model is general and applicable to different attack goals, such as "reach an unsafe state" or "denial of service".

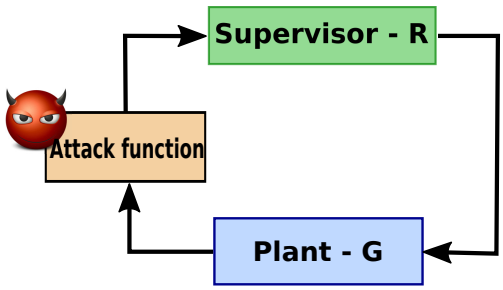


Figure 3.1: Sensor deception attacks in the supervisory control framework

The second task of this chapter is to incorporate this attack model to the supervisory control framework. Namely, we provide a way to analyze the closed-loop behavior of the controlled system under sensor deception attack. In other words, we precisely capture the language generated by the controlled system under attack.

The remainder of this chapter is organized as follows. A general sensor deception attack model is provided in Section 3.2. In Section 3.3, the closed-loop behavior of the controlled system under sensor deception attack is defined. We discuss different classes of attack functions in Section 3.4. For completeness purposes, Section 3.5 defines the closed-loop behavior using languages instead of automata. Finally, we conclude this chapter in Section 3.6.

Related work

Several recent works leveraged the concepts and techniques of SCT of DES to study cyber-security issues in CPS. The works in [20, 11] on intrusion detection and prevention of cyber-attacks using discrete event models focused on modeling the attacker as faulty behavior. Even though both works described deception attacks in the SCT framework, their attack model is intertwined with the plant and the supervisor models which makes it inflexible. Nevertheless, their attack definition is intuitive and easy to manipulate using automata operations. Our approach differs from theirs by first introducing a sensor deception attack model that is independent of the plant and the supervisor which introduces flexibility to the supervisory control framework under attack. Second, we provide a method to introduce this flexibility into their attack model so that we also leverage automata operations.

The bounded sensor deception attack model introduced in [13] is similar to the one adopted in our work. The attack model is independent of the plant and the supervisor. This model differs from our approach since it only covers bounded attacks, i.e., the attacker has a fixed bound on the number of modifications it can perform between plant executions. Our attack model covers bounded and unbounded attacks. An additional difference between our approach and the approach in [13] is the way the dynamical interaction between the attacker and the supervisor is captured.

In [14], the authors presented a study of supervisory control of DES under sensor deception attacks. Their attack model generalizes the one introduced in [13] since it does not impose a bound on the attacker. Our attack model differs from theirs in one aspect: the attacker memory. While in [14] the attacker has access to all previous events executed by the plant, it does not "remember" its own modifications. This difference allows us to precisely define the behavior generated by the interaction between the plant, the supervisor and the attack model.

Since the focus of our work is sensor deception attacks, our model differs from the attack models of actuator deception attacks described in [16]. Several prior works considered robust supervisory control under different notions of robustness [36, 37, 38, 39, 40], but they did not study robustness against attacks. In the cyber-security literature, some works have been carried out in the context of discrete event models, especially regarding opacity and privacy or secrecy properties [41, 42, 43, 24, 44]. These works are concerned with studying information release properties of the system, and they do not address the impact of an intruder over the physical parts of the system.

3.2 A general sensor deception attack model

3.2.1 Attack function

As illustrated in Fig. 3.1, the attacker intervenes in the communication channel between the plant's sensors and the supervisor. It has the ability to edit *some* of the sensor readings in this communication channel, by inserting fictitious events or deleting the legitimate events. Moreover, we denote

as the *compromised event set* the events that the attacker can manipulate. The attacker performs event editions based on its memory and the events executed by the plant. Informally, we have that the *attack function* f_A can be defined as $f_A : memory \times event\ observation \rightarrow edit\ string$. The attacker replaces the newly observed event by the edit string based on its memory and the observed event.

Let us formalize our intuitive attack function description by giving formal definitions for the compromised event set, $memory$, $event\ observation$ and $edit\ string$. First, we assume that the attacker observes all events in Σ_{obs} that are executed by G, i.e., the attacker has the same observable capabilities as the supervisor. Therefore, the $event\ observation$ domain is Σ_{obs}. For generality purposes, we assume that the compromised event set, denoted as Σ_{sen}, is a subset of the observable events, i.e., $\Sigma_{sen} \subseteq \Sigma_{obs}$. The attacker has the ability of inserting Σ_{sen} events in the communication channel or deleting Σ_{sen} events from this channel.

To formally introduce the domains for $memory$ and $edit\ string$, we first define two new sets of events. The sets $\Sigma_{ins} = \{ins(e) \mid e \in \Sigma_{sen}\}$ and $\Sigma_{del} = \{del(e) \mid e \in \Sigma_{sen}\}$ are defined as the sets of inserted and deleted events, respectively. We assume that $ins(e)$ and $del(e)$ are not defined in Σ for any $e \in \Sigma_{sen}$. These sets represent the actions of the attacker, i.e., $ins(e)$ ($del(e)$) denotes inserting fictitious (deleting legitimate) event $e \in \Sigma_{sen}$ in (from) the communication channel. For convenience, we define $\Sigma_{att} = \Sigma_{ins} \cup \Sigma_{del}$ as the set of attacker events and $\Sigma_{all} = \Sigma \cup \Sigma_{att}$ as the set of all events in this control system.

Based on these new sets, we define the two remaining missing variable domains. In the case of $memory$, we assume that we have an attacker that remembers everything it has seen, including its own edits. For this reason, $memory$ is defined to be strings in $(\Sigma_{obs} \cup \Sigma_{att})^*$. In respect to the $edit\ string$, we assume that the attacker might have more than one option in its hand, i.e., a nondeterministic edition. Moreover, the attacker outputs, in general, strings from the domain $(\Sigma_{obs} \cup \Sigma_{att})^*$. Therefore, the domain for $edit\ string$ is $2^{(\Sigma_{obs} \cup \Sigma_{att})^*}$, the set of sets of edit strings. Each element in $2^{(\Sigma_{obs} \cup \Sigma_{att})^*}$ is a set of edit strings.

Intuitively, the attacker is triggered by a new observation unless the plant has been just ini-

tialized. The initialization of the plant can trigger the attacker to insert fictitious events without observing an event. Other than this initialization case, the attacker is always triggered by a new event observation. Whenever the attacker receives a new event, there are two cases: the event is not compromised, not in Σ_{sen}; the event is compromised, in Σ_{sen}. If the event is not compromised, then the attacker must not delete the event from the channel, i.e., the event must reach the supervisor. Nonetheless, observing this event might trigger the attacker to insert fictitious events after the observation. On the other hand, if the event is compromised, then the attacker can choose to delete or not the event from the channel. Again, the observation of this event might trigger the attacker to insert fictitious events. Formally, we model an attacker as a nondeterministic string edit function.

Definition 3.1. *Given a system G and the compromised event set $\Sigma_{sen} \subseteq \Sigma_{obs}$, an attacker is defined as a (potentially partial) function $f_A : (\Sigma_{obs} \cup \Sigma_{att})^* \times (\Sigma_{obs} \cup \{\epsilon\}) \to (2^{(\Sigma_{obs} \cup \Sigma_{att})^*} \setminus \emptyset)$ such that f_A satisfies the following constraints:*

1. $f_A(\epsilon, \epsilon) \subseteq {\Sigma_{ins}}^*$ *and* $\forall s \in ((\Sigma_{obs} \cup \Sigma_{att})^* \backslash \{\epsilon\}) : \ f_A(s, \epsilon) = \{\epsilon\}$;

2. $\forall s \in (\Sigma_{obs} \cup \Sigma_{att})^*$, $e \in \Sigma_{obs} \backslash \Sigma_{sen}$ *:* $f_A(s, e) \subseteq \ \{e\}{\Sigma_{ins}}^*$;

3. $\forall s \in (\Sigma_{obs} \cup \Sigma_{att})^*$, $e \in \Sigma_{sen}$*:* $f_A(s, e) \subseteq \ \{e, \ del(e)\}{\Sigma_{ins}}^*$.

The function f_A captures a general model of sensor deception attack. Given the past *edited* string $s \in (\Sigma_{obs} \cup \Sigma_{att})^*$ and observing a new event $e \in \Sigma_{obs}$ executed by G, the attacker may choose to edit e based on Σ_{sen} and replace e by selecting an edited suffix from the set $f_A(s, e)$. The first case in the above definition gives an initial condition for an attack. The second case constrains the attacker from erasing e when e is outside of Σ_{sen}. Since the event e is not compromised, it cannot be deleted by the attacker. However, observing event e might trigger the attacker to insert event after the supervisor observes e. Lastly, the third case in Definition **3.1** is for $e \in \Sigma_{sen}$; the attacker can edit the event to any string in the set $\{e, \ del(e)\}{\Sigma_{ins}}^*$. In this case, the attacker can choose to delete or not event e since it is a compromised event. Again, after observing this event might trigger the attacker to insert fictitious events.

As mentioned before, f_A only defines the possible edited suffixes based on the last observed event and the edit history. It is interesting to define a function that defines the possible edited strings based on the executed string. Formally, the string-based edit (potentially partial) function $\hat{f}_A : \Sigma_{obs}^* \to 2^{(\Sigma_{obs} \cup \Sigma_{att})^*}$ is defined as

$$\hat{f}_A(se) := \{ut \in (\Sigma_{obs} \cup \Sigma_{att})^* \mid u \in \hat{f}_A(s) \wedge t \in f_A(u, e)\}$$

for any $s \in \Sigma_{obs}^*$ and $e \in \Sigma_{obs}$, and

$$\hat{f}_A(\epsilon) := f_A(\epsilon, \epsilon).$$

The function $\hat{f}_A(s)$ returns the set of possible edited strings for a given string $s \in \Sigma_{obs}^*$. Note that, in general, $\hat{f}_A$ is a partial function, and $\hat{f}_A(s)$ may only be defined for selected $s \in \Sigma_{obs}^*$.

Lemma 3.1. *For any $s \in \Sigma_{obs}^*$, then $\hat{f}_A(s) \neq \emptyset$.*

Proof. We prove this result by induction on the length of $s \in \Sigma_{obs}^*$.

Basis: by the definition of f_A, we have that $f_A(\epsilon, \epsilon) \neq \emptyset$. Therefore, $\hat{f}_A(\epsilon) \neq \emptyset$.

Induction hypothesis: assume that $\hat{f}_A(s) \neq \emptyset$ for $|s| = n$.

Induction step: let $s \in \Sigma_{obs}^*$ such that all s such that $|s| = n$ and $e \in \Sigma_{obs}$. The induction hypothesis provides that $\hat{f}_A(s) \neq \emptyset$, which implies that there exists $u \in \hat{f}_A(s)$. By the definition of f_A, $f_A(t, e) \neq \emptyset$ for any $t \in (\Sigma_{obs} \cup \Sigma_{att})^*$. Therefore, $\hat{f}_A(se) \neq \emptyset$. □

3.2.2 Attack function as an automaton

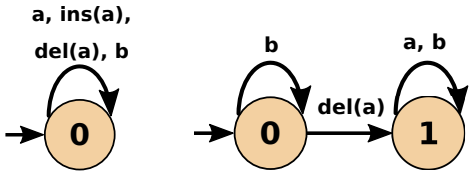
In order to have effective ways of manipulating the attack function, we assume that f_A has been encoded into a DFA $A = (X_A, \Sigma_{obs} \cup \Sigma_{att}, \delta_A, x_{0,A})$. This assumption allows us to define the closed-loop behavior using automata operations with A. However, we lose generality since the attacker's memory must be a regular language over $(\Sigma_{obs} \cup \Sigma_{att})^*$. This assumption is similar to the assumption of a supervisor realization.

Given the automaton A that encodes an f_A, then the function f_A is extracted from A as follows:

$\forall s \in \mathcal{L}(A)$ and $e \in \Sigma_{obs}$, $f_A(s,e) = \{t \in \{e, del(e)\}{\Sigma_{ins}}^* \mid \delta_A(x_{0,A}, st)!\}$, $f_A(\epsilon,\epsilon) = \{t \in {\Sigma_{ins}}^* \mid \delta_A(x_{0,A}, t)!\}$, and $f_A(s,e) = \{e\}$ for all $s \in (\Sigma_{obs} \cup \Sigma_{att})^* \setminus \mathcal{L}(A)$ and $e \in \Sigma_{obs}$.

This formulation provides a simple way to handle attack functions and it characterizes the behavior of the attacker. It also provides a way to define specific attackers that are more constrained than the usual constraints of Definition **3.1**, i.e., when some prior knowledge about the attacker is available. In other words, the automaton A can encode different attack strategies, e.g., replacement attack [14], bounded attack [13], etc. One important attack strategy for this problem is the all-out attack strategy introduced in [11, 12]. In this model, the attacker could attack whenever it is possible. We return to this discussion about attack constraints after we describe the closed-loop behavior under attack.

Example 3.1. *Let $\Sigma = \Sigma_{obs} = \{a, b\}$ be the set of events in the system G and let $\Sigma_{sen} = \{a\}$ be the compromised event set. We define f_{A^1} as: $f_{A^1}(\epsilon,\epsilon) = \Sigma_{ins}^*$, $f_{A^1}(s,a) = \{a, del(a)\}\Sigma_{ins}^*$ and $f_{A^1}(s,b) = \{b\}\Sigma_{ins}^*$ for any $s \in \Sigma_{all}^*$. Namely, the attacker can perform all possible actions that satisfy the constraints in Def. **3.1**. This attacker is the "all-out" attacker and the automaton A^1 depicted in Fig. 3.2(a) encodes f_{A^1}. Although the all-out strategy is a nondeterministic strategy since the attacker can try all possible combinations of attacks, its automaton representation is a deterministic automaton. Automaton A^2 shown in Fig. 3.2(b) encodes a one sensor deletion attack strategy.*



(a) A^1 (b) A^2

Figure 3.2: Attack functions

3.3 The controlled behavior under sensor deception attack

The output of the attack function f_A as defined in Def. **3.1** is inconsistent with the supervisor input. We have defined it in this manner on purpose so that we can define the attack function as general as possible. There are two problems that we have to fix in order to define the closed-loop behavior: the attacker's nondeterministic choice and matching both the input and the output of f_A to the output domain of G and input domain of R, respectively.

First, we deal with the attacker's nondeterministic choice in f_A. This is dealt similarly as in the plant's case. At any given state of G, there might be more than one event that the G can execute, e.g., either the red or the blue car enter the intersection from the initial state of G in Fig. 2.5. The plant selects *one* of the possible decisions in order to evolve. Similarly, we assume that the attacker selects *one* of the possible decisions from the possible choices that f_A gives. In Example **3.1**, the attacker chooses one of the strings in $f_{A^1}(\epsilon, \epsilon) = \Sigma^*_{ins}$ at its initial state, e.g., it might choose ϵ (no insertion). After this decision, the attacker updates its memory based on its selection and awaits for a new observation.

For the second issue instead of projecting the events in Σ_{att} to Σ_{obs}, we decide to augment both the plant G and the supervisor R such that they include the attacker actions. Namely, we lift the closed-loop language space from Σ to Σ_{all}. This new framework gives us leverage in the analysis of the closed-loop system since it is defined over events in Σ_{all}. The behavior added in the lifting process only reflects the attacker action and it does not add infeasible behavior with respect to G.

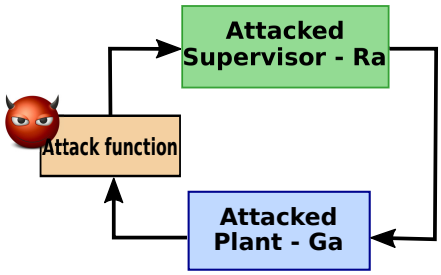


Figure 3.3: Augmented supervisory control framework under sensor deception attack

The supervisory control framework depicted in Fig. 3.1 is substituted by the one depicted in Fig. 3.3, where G_a and R_a are the augmented versions of G and R. We will define these augmented

versions of G and R shortly. Before their definition, we define three projection operator that help us analyze the lifted behavior in Σ_{all}. Intuitively, these operators project the attack actions, Σ_{att}, to how they are executed by the plant, observed by the supervisor, or projected to Σ_{sen}.

Definition 3.2. *$P^S : \Sigma_{all} \to \Sigma$, $P^G : \Sigma_{all} \to \Sigma$ and $\mathcal{M} : \Sigma_{all} \to \Sigma$ are natural projections that treat events in the following manner:*

$$P^S(ins(e)) := e,\ e \in \Sigma_{sen};\ P^S(del(e)) := \epsilon,\ e \in \Sigma_{sen};\ P^S(e) := e,\ e \in \Sigma \tag{3.1}$$

$$P^G(ins(e)) := \epsilon,\ e \in \Sigma_{sen};\ P^G(del(e)) := e,\ e \in \Sigma_{sen};\ P^S(e) := e,\ e \in \Sigma \tag{3.2}$$

$$\mathcal{M}(ins(e)) = \mathcal{M}(del(e)) := e,\ e \in \Sigma_{sen};\ \mathcal{M}(e) := e,\ e \in \Sigma \tag{3.3}$$

The superscript S determines that P^S describes how the *supervisor* observes the modified events. The supervisor observes inserted events as if they are legitimate observation, while it does not observe deleted events since they were deleted from the channel. Namely, given an event, P^S outputs the legitimate event if the event was inserted by the attacker, it outputs the empty string if the event was deleted by the attacker, and it outputs the legitimate event otherwise. Similarly, the superscript G is used because P^G describes how the *plant* executed the modified events. Insertions are fictitious events that do not occur in G, therefore they are project to the empty string by P^G. On the other hand, deletion events are events executed by G, but deleted by the attacker. For this reason, P^G projects deleted events to legitimate events. Lastly, the mask $\mathcal{M}$ removes the operators ins or del.

3.3.1 Attacked plant

The attacked plant is obtained by augmenting the plant G with the possible actions of the attacker, i.e., to augment G with events in Σ_{att}. In order words, G_a is a copy of G with additional transitions with respect to Σ_{att}. In fact, G_a contains all possible attacker actions with respect to Σ_{att} and G.

We start by introducing insertion events to the attacked plant. In the attacked plant, the events $ins(e)$ simulate the insertion ability of the attacker. Since these events are fictitious events that

might be introduced in the communication channel, they cannot alter the state of the plant G. For this reason, the insertion events $ins(e)$ are defined to be self-loops in G_a. Every state of G_a has self-loops with all insertion events.

On the other hand, the events $del(e)$ simulate the deletion ability of the attacker. An event can only be deleted by the attacker, if this event actually occurred in the plant. For this reason, these events are defined in parallel to transitions with events in Σ_{sen}. Formally, we define the attack plant as following:

Definition 3.3. *Given G and Σ_{sen}, we define the attacked plant G_a as:*

$$G_a := (X_{G_a} := X_G, \Sigma_{all}, \delta_{G_a}, x_{0,G_a} := x_{0,G}) \textit{ where}$$

$$\delta_{G_a}(x,e) := \begin{cases} \delta_G(x,e) & \textit{if } e \in \Sigma \textit{ and } e \in En_G(x) \\ x & \textit{if } e \in \Sigma_{ins} \\ \delta_G(x,e') & \textit{if } e = del(e') \textit{ and } e' \in En_G(x) \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{3.4}$$

Example 3.2. *Let us construct the attacked plant G_a for the plant of our intersection example (Example 2.4) when $\Sigma_{sen} = \{R_{int}\}$. G_a is depicted in the right figure in Fig. 3.4 where events in Σ_{att} are in red. For convenience, the model of G is shown in the left of Fig. 3.4. Note that the states in G_a are a copy of the ones in G.*

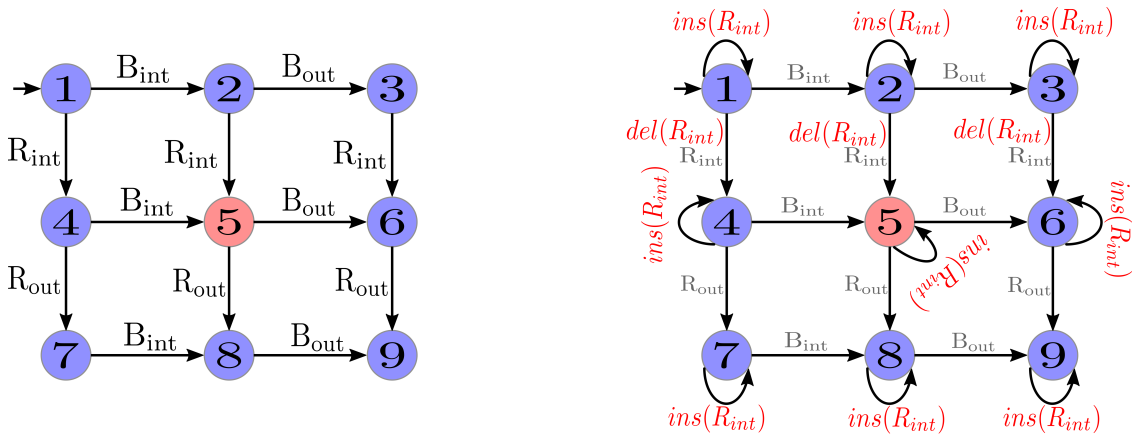


Figure 3.4: Plant of the intersection example (left) and its attacked plant (right) for $\Sigma_{sen} = \{R_{int}\}$

We can show that we can extract the language of G from G_a by simply using the projection P^G. This result follows from the definition of G_a and P^G. Although this is a straightforward result, it shows that G_a is exactly a copy of G augmented with attacker actions.

Proposition 3.1. $\mathcal{L}(G) = P^G(\mathcal{L}(G_a))$

3.3.2 Attacked supervisor

The construction of R_a is similar to the construction of G_a. However, insertion and deletion events affect the supervisor in the opposite manner as they affect the plant as we saw in the definition of P^S. Namely, insertion events are seen as legitimate events where deletion events are not observed by the supervisor.

The attacked supervisor R_a is a copy of R with additional transition with events in Σ_{att}. Deletion events $del(e)$ are defined to be self-loops in the states of R_a where the legitimate events $e \in \Sigma_{sen}$ are defined. If the event is deleted by the attacker, then it must have been enabled by the supervisor first. Moreover, they are self-loops since the supervisor does not receive any information, i.e., it remains in the same state.

On the other hand, insertion events $ins(e)$ are defined in parallel to their legitimate events $e \in \Sigma_{sen}$. In other words, insertion events are seen by the supervisor as legitimate events.

Since we have not yet constrained the attack function, it is possible that the attacker inserts an event that is not enabled by the supervisor. For simplicity, we assume that the supervisor simply "ignores" insertions of events that are not enabled by the current control decision. We revisit this assumption later in this chapter as well as in Chapter IV.

Definition 3.4. *Given R and Σ_{sen}, we define the attacked supervisor as:*

$$R_a := (X_{R_a} := X_R, \Sigma_{all}, \delta_{R_a}, x_{0,R_a} := x_{0,R}) \textit{ where}$$

$$\delta_{R_a}(x,e) := \begin{cases} \delta_R(x,e) & \text{if } e \in \Sigma \text{ and } e \in En_R(x) \\ x & \text{if } e \in \Sigma_{del} \text{ and } \mathcal{M}(e) \in En_R(x) \\ \delta_R(x, P^S(e)) & \text{if } e \in \Sigma_{ins} \text{ and } \mathcal{M}(e) \in En_R(x) \\ x & \text{if } e \in \Sigma_{ins} \text{ and } \mathcal{M}(e) \notin En_R(x) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.5}$$

Remark 3.1. *Although R_a is a modification of R, this modification does not alter the control decisions taken by R. In other words, the plant G is still supervised by the supervisor R. R_a is defined such that it takes into account the modifications made by the attacker.*

Example 3.3. *Let us construct the attacked supervisor R_a for the supervisor R_1 of our intersection example (Example **2.4**) when $\Sigma_{sen} = \{R_{int}\}$. R_a is depicted in the right figure in Fig. 3.5, where events in Σ_{att} are in red. Supervisor R_1 is shown in the left of Fig. 3.5. Note that states in R_a are a copy of the ones in G.*

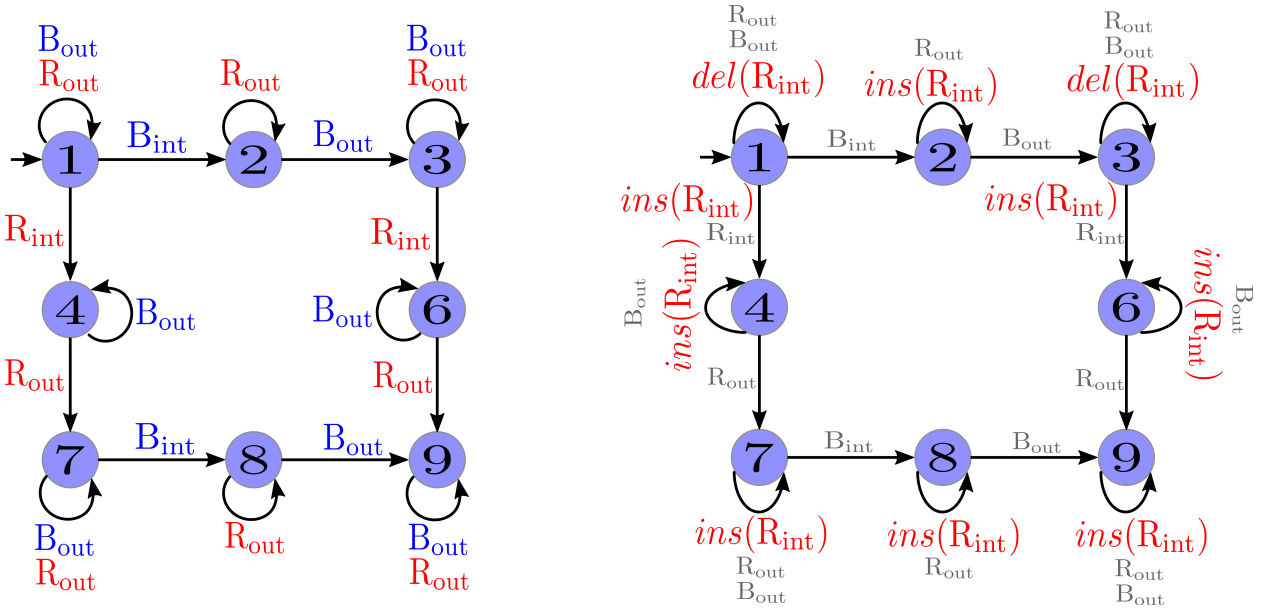


Figure 3.5: Supervisor of the intersection example (left) and its attacked supervisor (right) for $\Sigma_{sen} = \{R_{int}\}$

Since we added the assumption that the attacked supervisor R_a ignores insertion events that are disabled in R, we cannot provide a result as Prop. **3.1**.

3.3.3 Controlled language under sensor deception attack

All the ingredients to define the closed-loop language of the supervisory control framework under sensor deception attacks (Fig 3.3) are formally defined. We now need to compose them in order to formally define the closed-loop language. We use the parallel composition operator (Def. **2.3**) to compose G_a, R_a, and A.

Definition 3.5. *Given R_a, G_a and A, the closed-loop language of the attacked system under sensor deception attacks is defined by $\mathcal{L}(G_a||A||R_a)$. The closed-loop language executed by G is defined by $\mathcal{L}(S_A/G) := P^G(\mathcal{L}(G_a||A||R_a))$.*

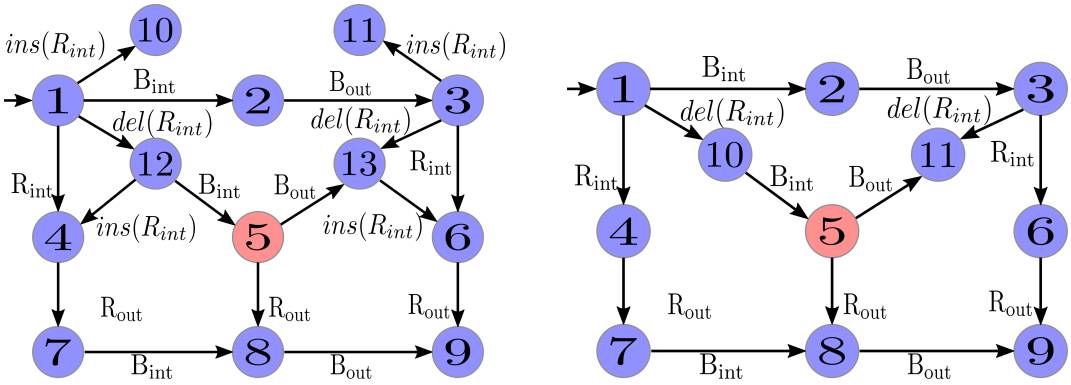
Example 3.4. *Let us obtain the closed-loop language of the attacked system our intersection example (Example **3.2**) where $\Sigma_{sen} = \{R_{int}\}$. We use the attacked supervisor constructed in Example **3.3**. First, we assume that A^1 is the all-out attacker with respect to $\Sigma_{sen} = \{R_{int}\}$. Figure 3.6(a) depicts the automaton $G_a||A^1||R_a$. In this case, we can see that the attacker can successfully reach the critical state 5 via the string $s = del(R_{int})B_{int}$. Note that, $\delta_G(x_{0,G}, P^G(s)) = 5$ which in fact reaches the critical state in G. However, this attack strategy might lead to non-critical deadlock states, states 10 and 11.*

*Figure 3.6(b) depicts the automaton $G_a||A^2||R_a$ where A^2 is the one-deletion attacker constructed as in Example **3.1**. This attack strategy can also successfully reach the critical state 5 via string $del(R_{int})B_{int}$.*

3.4 Different classes of attack functions

3.4.1 All-out attacker

As we already mentioned, the all-out attacker is an important attack strategy even though it is a simple one. The word "strategy" is misleading since the all-out attack strategy just means all possible attacker actions. In this strategy, we enumerate all possible actions that the attacker can perform at any given time. Nevertheless, it does not mean that the attacker will indeed perform them. This



(a) $G_a||A^1||R_a$ (we omitted the self-loops with $ins(R_{int})$)

(b) $G_a||A^2||R_a$

Figure 3.6: Closed-loop attacked systems

describes an attacker that is only constrained by the constraints defined in Def. **3.1**. Note that, the construction of G_a and R_a depicts the "all-out" strategy, i.e., $\mathcal{L}(G_a||R_a) = \mathcal{L}(G_a||A_{\text{all-out}}||R_a)$.

This attack strategy becomes useful when there is not much information about the attacker constraints. When we investigate defense techniques in Chapter VI, the all-out attack strategy becomes essential to show general robutness properties. This strategy was first introduced in [11] to investigate attack detection properties in attacked systems. We refrain ourselves to continue this discussion and return to it later in Chapter VI.

3.4.2 Interruptible attacker

Until this point, we have not constrained the attacker's behavior with the supervised system. We have assume that any attack modification the attacker decides to perform is successfully applied. However, the supervised system is a dynamic entity and this "unconstrained" attacker is too powerful. For this reason, we introduce the subclass of sensor deception attacks called *interruptible attacks*. We assume that the plant can interrupt the attacker's modification at any point. This constraint is intuitively depicted in Fig. 3.7 where the attacker must have a contingency strategy when interrupted by the plant.

Formally, we define interruptible attack strategies as:

Definition 3.6. *An attack function f_A is an* ***interruptible*** *attack function if* $(\forall s \in (\Sigma_{obs} \cup$

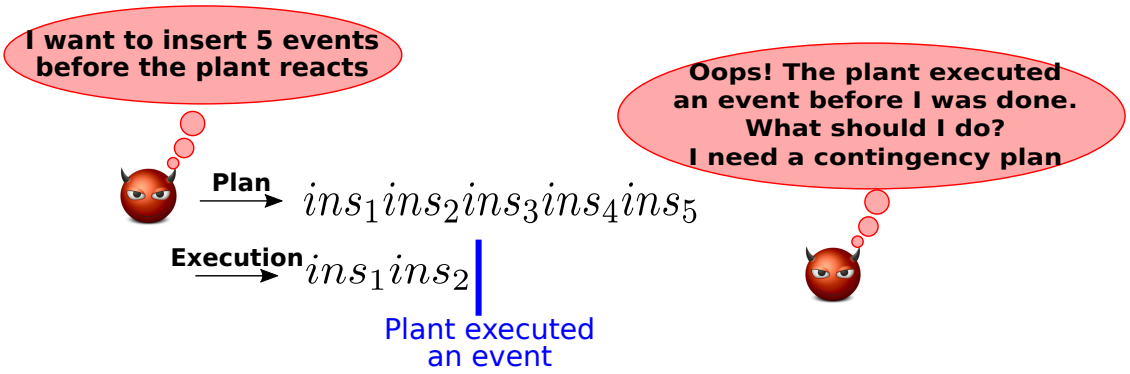


Figure 3.7: Interruptible attacker strategy

$\Sigma_{att})^*)(\forall e \in \Sigma_{obs})\ [t \in f_A(s,e) \Rightarrow pre(t)\backslash\{\epsilon\} \subseteq f_A(s,e)]$.

3.4.3 Unbounded deterministic attacker

Contrary to the interruptible attack strategies, we now assume that the attacker has "time" to perform any *unbounded* modification. In other words, the system does not execute any event until the attacker finishes its modification. Such an attack function is defined as an *unbounded deterministic* attack function.

Definition 3.7. *An **unbounded** deterministic attack function is an attack function f_A s.t.* $\big(\forall s \in (\Sigma_{obs} \cup \Sigma_{att})^*\big)(\forall e \in \Sigma_{obs})[f_A(s,e)! \Rightarrow |f_A(s,e)| = 1]$.

3.4.4 Bounded deterministic attacker

The previous scenario considers a powerful attacker, and it might be unrealistic in many real applications. In this case, the second scenario limits the first one by considering only *bounded deterministic* attack functions. In other words, we assume that the system does not react up to a bounded number of modifications made by the attacker. Bounded deterministic attack functions are defined next.

Definition 3.8. *Given $N_A \in \mathbb{N}^+$, a **bounded** deterministic attack function is an attack function f_A s.t.* $\big(\forall s \in (\Sigma_{obs} \cup \Sigma_{att})^*\big)(\forall e \in \Sigma_{obs})\ [(f_A(s,e)! \Rightarrow |f_A(s,e)| = 1) \wedge (t \in f_A(s,e) \Rightarrow |s_A| \leq N_A)]$.

3.4.5 Other attackers

Although there are other attacker constraints that we can introduce, we limit ourselves to these four subclasses of sensor deception attack strategies. Definition **3.1** models in a general manner sensor deception attacks in the framework of supervisory control theory.

3.5 Controlled behavior under sensor deception attack - language definition

In Section 3.3, we defined the closed-loop behavior of the controlled system under sensor deception attack with the assumption that the attacker is encoded via automaton A. For completeness of this chapter, we provide this closed-loop behavior without this assumption. This section is mainly for completeness purposes and it will not be used in the following chapters. The reader can skip this section without any consequences since we only use the more intuitive definition described in Section 3.3.

The presence of the attacker induces a new controlled language that needs to be investigated. More specifically, S, $\hat{f}_A$, and P^S together effectively generate a new supervisor S_A for system G. Formally, we define $S_A : \Sigma^*_{obs} \rightarrow 2^\Gamma$ as $S_A(s) = [S_P \circ P^S \circ \hat{f}_A(s)]$. Note that $\hat{f}_A(s)$ returns the set of modified strings and S assigns a control decision to each projected modified string; S returns the set containing all these control decisions. An equivalent definition is $S_A(s) = \{\gamma \mid \exists s_A \in \hat{f}_A(s) \text{ s.t. } \gamma = S(P^S(s_A))\}$. Defining the supervised language based on nondeterministic control decisions is cumbersome and complicated [37]. Nonetheless, we can avoid this difficulty by analyzing the language generated by the events in $\Sigma \cup \Sigma_{att}$. For this reason, we define the function $S^d_A = S_P \circ P^S$ to be the deterministic part of S_A. The function S^d_A is used when the attacker has decided which modified string to send to the supervisor. Based on f_A and S^d_A the

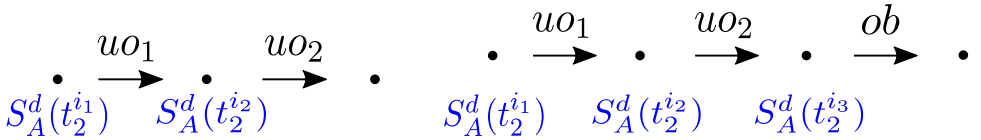
language generated by S_A/G is defined recursively as follows:

1. $\epsilon \in \mathcal{L}(S_A/G)$

2. $\big(t_1 \in \mathcal{L}(G) \cap \Sigma^*_{uobs}(\Sigma_{obs} \cup \{\epsilon\})\big) \wedge \big(\exists t_2 \in f_A(\epsilon,\epsilon) \text{ and } i_1 \leq i_2 \leq \ldots \leq i_{|t_1|} \in \mathbb{N}^{|t_2|} \text{ s.t. } \forall j \in \mathbb{N}^{|t_1|} : t_1[j] \in S^d_A(t_2^{i_j})\big) \wedge \big(P_{\Sigma\Sigma_{obs}}(t_1) \neq \epsilon \Rightarrow i_{|t_1|} = |t_2|\big) \Leftrightarrow t_1 \in \mathcal{L}(S_A/G)$

3. $\big(s \in \mathcal{L}(S_A/G)\big) \wedge \big(s[|s|] \in \Sigma_{obs}\big) \wedge \big(st_1 \in \mathcal{L}(G) \text{ where } t_1 \in \Sigma^*_{uobs}(\Sigma_{obs} \cup \{\epsilon\})\big) \wedge \big(\exists t_3 \in \hat{f}_A(P_{\Sigma\Sigma_{obs}}(s^{|s|-1})),\ \exists t_2 \in f_A(t_3, s[|s|]) \text{ and } i_1 \leq i_2 \leq \ldots \leq i_{|t_1|} \in \mathbb{N}^{|t_2|} \text{ s.t. } \forall j \in \mathbb{N}^{|t_1|} : t_1[j] \in S^d_A(t_3 t_2^{i_j})\big) \wedge \big(P_{\Sigma\Sigma_{obs}}(t_1) \neq \epsilon \Rightarrow i_{|t_1|} = |t_2|\big) \Leftrightarrow st_1 \in \mathcal{L}(S_A/G)$

The above definition captures the intricate interaction between plant, supervisor and attacker. Two important concepts are applied in this definition. First, the supervisor issues a control decision whenever it receives an observable event. An observable event can be either a legitimate event or a fictitious event inserted by an attacker. Second, the plant can execute any unobservable event enabled by the current control decision. To demonstrate how to compute $\mathcal{L}(S_A/G)$, we illustrate condition (2) in Figure 3.8.



(a) Only unobservable events (b) Last event is observable

Figure 3.8: Demonstration of Condition (2)

Assume that $uo_1, uo_2 \in \Sigma_{uobs}$, $ob \in \Sigma_{obs}$ and uo_1uo_2, $uo_1uo_2ob \in \mathcal{L}(G)$. Figure 3.8(a) describes how $t_1 = uo_1uo_2 \in \mathcal{L}(S_A/G)$. The string uo_1uo_2 belongs to $\mathcal{L}(S_A/G)$ whenever there exists a string $t_2 \in f_A(\epsilon,\epsilon)$ and indices $i_1 \leq i_2 \in \mathbb{N}$ such that $uo_1 \in S^d_A(t_2^{i_1})$ and $uo_2 \in S^d_A(t_2^{i_2})$. Similarly, Fig. 3.8(b) describes how $uo_1uo_2ob \in \mathcal{L}(S_A/G)$. If we use the same indices i_1, i_2 as before, then we just need to test if $ob \in S^d_A(t_2^{i_3}) = S^d_A(t_2)$. In the case of the last event being observable, we assume that the attacker has finished its entire modification t_2. On the other hand, we assume that unobservable events can be executed based on control decisions of string prefixes

of t_2. Condition (3) applies the same mechanism of Condition (2), however it has nuances related to previous modifications made by the attacker.

3.6 Conclusion

In this chapter, we defined a general model for sensor deception attacks. This attacker intervenes in the communication channels between the system's sensors and the supervisor and it has the ability to edit some of the sensor readings in these communication channels, by inserting fictitious events or deleting the legitimate events.

Following the attacker definition, we integrate this attacker model into the supervisory control framework. In other words, we define the supervisory control framework under sensor deception attacks. Based on this definition, we are able to analyze the closed-loop behavior of the controlled system under sensor deception attacks.

Four specific sensor deception attack models are formally introduced. These attack models are defined by constraining our general sensor deception attack model based on different assumptions on the attacker's capability.

CHAPTER IV

Synthesis of stealthy sensor deception attacks for supervisory control

4.1 Introduction

Many complex systems already have existing supervisors in place; however, these supervisors were designed without taking into account sensor deception attacks. In Chapter III, we provide a supervisory control framework for analyzing closed-loop systems under sensor deception attacks. Nevertheless, we did not explore when and how a sensor deception attack could cause damage to the plant. Our main goal in this chapter is to answer these two questions: When does a sensor deception attack cause damage? How does this sensor deception attack cause damage?

Answering these questions discloses possible unknown vulnerabilities of the closed-loop system. To answer them, we put ourselves into the attacker's shoes. We search for ways, if possible, to cause damage to the plant. Namely, the behavior generated by the controlled system under attack can reach a critical region of the uncontrolled system, as shown in Figure 4.1

Back to the definition of the closed-loop attacked system, the described attacker did not care if its identity was revealed, i.e., when inserting unfeasible behavior. At this point, we (as the attacker) think that it is important to conceal our attacks. In other words, the damage must be caused without raising any alarm in the control system, i.e., in a *stealthy* manner. By raising alarms, we assume that the closed-loop system has an embedded intrusion detection module, which determines whether the system is under attack. A key issue is determining whether an attacker can exploit the supervisor and cause damage to the system without setting off the intrusion module.

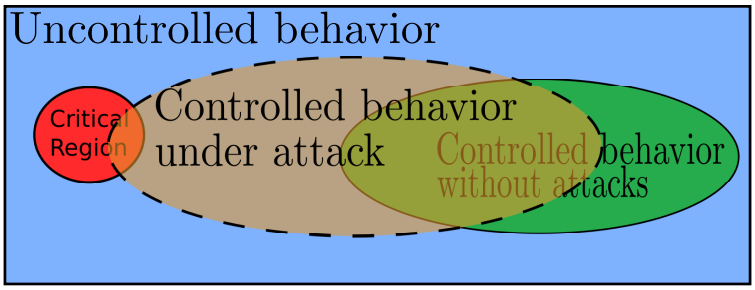


Figure 4.1: Controlled system under attack

In our research, we first note that the supervisor is passive since its actions have already been designed; and second, the attacker does not want to set off the intrusion detection module. Because the attacker wants to remain stealthy during the entire attack, each sensor modification it makes must be deliberately planned. Therefore, a game between attacker and supervisor is designed.

Leveraging graph-games theory techniques [45], an algorithm that synthesizes stealthy sensor deception attacks that successfully causes damage to the system. In words, this attack can reach a critical region of the uncontrolled system and it does not trigger the intrusion detector. At this point, it becomes possible to know the vulnerabilities of the controlled system under attack.

We study three specific types of attacks that are based on the interaction between the attacker and the controlled system. The methodology developed to synthesize these attacks is inspired by the work in [24, 29, 30]. As in these works, we employ a discrete structure to model the game-like interaction between the supervisor and the environment. We call this structure an *Insertion-Deletion Attack* structure (or IDA). By construction, an IDA embeds all desired scenarios where the attacker modifies some subset of the sensor events without being noticed by the supervisor. Once constructed according to the classes of attacks under consideration, an IDA serves as the basis for solving the synthesis problem. In fact, this game-theoretical approach provides a structure for each attack class that incorporates *all successful stealthy attacks*. Different stealthy attack strategies can be extracted from this structure. This is a distinguishing feature of our work as compared to previous works mentioned above. By providing a general synthesis framework, our goal is to allow CPS engineers to detect and address potential vulnerabilities in their control systems.

The remainder of this chapter is organized as follow. The problem statement is formalized in

Section 4.2. Section 4.3 describes the IDA structure and its properties. Section 4.4 introduces the AIDA (All Insertion-Deletion Attack structure) and provides a construction algorithm for it. Sections 4.5 introduce for each attack type their stealthy IDA structure, together with a simple synthesis algorithm to extract an attack function. Lastly, Section 4.6 presents concluding remarks.

Related work

Previous works such as [11, 21, 20] on intrusion detection and prevention of cyber-attacks using discrete event models were focused on modeling the attacker as faulty behavior. Their corresponding methodologies relied on fault diagnosis techniques.

Recently, [13] proposed a framework similar to the one adopted in our work, where they formulated a model of bounded sensor deception attacks. Our approach is more general than the one in [13], since we do not impose a *normality* condition to create an attack strategy; this condition is imposed to obtain the so-called *supremal controllable and normal language* under the attack model. In addition to bounded sensor deception attacks, we consider two other attack models.

In [14], the authors presented a study of supervisory control of DES under attacks. They introduced a new notion of *observability* that captures the presence of an attacker. However, their study is focused on the supervisor's viewpoint and they do not develop a methodology to design attack strategies. They assume that the attack model is given and they develop their results based on that assumption. In that sense, the work in [14] is closer to robust supervisory control and it is complementary to our work.

Several prior works considered robust supervisory control under different notions of robustness [36, 37, 38, 39, 40], but they did not study robustness against attacks. In the cyber-security literature, some works have been carried out in the context of discrete event models, especially regarding opacity and privacy or secrecy properties [41, 42, 43, 24]. These works are concerned with studying information release properties of the system, and they do not address the impact of an intruder over the physical parts of the system.

4.2 Problem formulation

In Chapter III, we provide means to analyze the supervisory control framework under sensor deception attacks. To analyze this new framework, we assumed that the attacker function is given, i.e., f_A is well-defined. As we discussed in the introduction of this chapter, we now focus on *searching* specific attack functions, i.e., we investigate a synthesis problem. Nonetheless, Chapter III is crucial for the formalization of the problem investigated in this chapter. In other words, we assume the general framework described in Chapter III as our starting point in the formalization of this chapter. Namely, the standard supervisory control framework with plant G and supervisor R, and an attacker that hijacks the sensor readings $\Sigma_{sen} \subseteq \Sigma_{obs}$ as shown in Fig. 4.2.

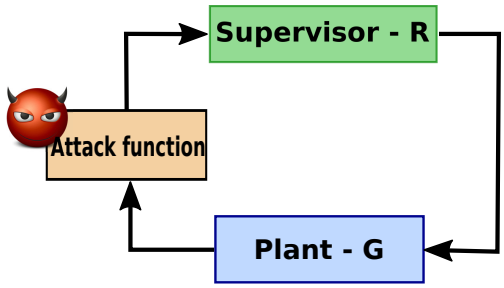


Figure 4.2: Sensor deception attacks in the supervisory control framework

Since we are searching for specific attack functions, we need to think as the attacker. We need to answer questions such as "What is the attacker's goals?", "Which information about the system does the attacker have?", "Are there any constraints to follow?", etc. In this way, we can portray the attacker so that we can formalize the "searching" problem.

First, we informally portray the attacker we would like to formalize. Intuitively, the attacker desires *to damage the plant*. For example, the attacker wants to cause the car collision in Example **3.4** where as the deadlock scenario is not appealing for this attacker.

This attacker also yearns *stealthiness*, i.e., it does not want its actions to be "discovered". In the definition of the attacked supervisor, we assumed that the supervisor does not react when it observes an unexpected event. This assumption was reasonable when we were defining the attacked controlled language. However, this new attacker considers risky to have the same assumption. In

fact, it is likely that the controlled system has an intrusion detection mechanism which the attacker does not want to trigger.

Finally, this attacker is *knowledgeable*, i.e., it has information about the controlled system. We start our formalization by making an assumption to describe the attacker's knowledge of the controlled system.

Assumption 4.1. *The attacker* knows *the models of G and R.*

Therefore, the attacker has full knowledge of the controlled system. This assumption lets the attacker understand its actions in the controlled system, i.e., it can effectively construct the framework depicted in Fig 3.3 once it has decided its strategy.

With respect to the attacker's desire of causing damage to the plant, this can be specified based on the states of G.

Assumption 4.2. *The plant G has a set of critical unsafe states defined as $X_{crit} \subset X_G$ such that $\forall x \in X_{crit}$, x is never reached when R controls G and no attacker is present.*

In general, not all states reached by strings of G that are disabled by R (when no attacker is present) are critically unsafe. In practice, there will be certain states among those that correspond to physical damage to the system, such as "overflow" states or "collision" states, for instance. Similar notions of critical unsafe states have been used in other works, e.g., [46, 21]. Therefore, the objective of the attacker is to force the controlled behavior under attack $\mathcal{L}(S_A/G)$ to reach any state in X_{crit}.

Lastly, we must address the attacker's desire to remain stealthy. The attacker does not want its actions to be "discovered"[1]. Therefore, all the actions taken by the attacker and their consequences must "look" as if the controlled system is not under attack. Every string that the supervisor observes must be defined in the controlled system without attacks (Fig. 4.1). In other words, the supervisor cannot receive an unexpected event otherwise it will seem questionable.

[1]This implies that the attacker assumes an intrusion detection trying to "discover" its actions.

Definition 4.1. *An attacker f_A is* stealthy *if* $\forall s \in \mathcal{L}(S_A/G)$

$$P^S\big(\hat{f}_A(P_{\Sigma\Sigma_{obs}}(s))\big) \subseteq P_{\Sigma\Sigma_{obs}}(\mathcal{L}(R/G))$$

If f_A is given as an automaton A, then it is stealthy if

$$P_{\Sigma\Sigma_{obs}}(P^S(\mathcal{L}(G_a||A||R_a))) \subseteq P_{\Sigma\Sigma_{obs}}(\mathcal{L}(R/G))$$

Finally, we are able to formally state the synthesis of stealthy sensor deception attack problem.

Problem 4.1 (Synt. of Stealthy Sensor Deception Attacks)**.** *Given an attacker that has full knowledge of the models G and R, and is capable of compromising events $\Sigma_{sen} \subseteq \Sigma_{obs}$, synthesize an attack function f_A such that it generates a controlled language $\mathcal{L}(S_A/G)$ that satisfies:*

1. $\forall s \in P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_A/G)).\hat{f}_A(s)$ *is defined* ***(Admissibility)****;*

2. $\forall s \in \mathcal{L}(S_A/G).P^S\big(\hat{f}_A(P_{\Sigma\Sigma_{obs}}(s))\big) \subseteq P_{\Sigma\Sigma_{obs}}(\mathcal{L}(R/G))$ ***(Stealthiness)****;*

3(a). $\exists s \in \mathcal{L}(S_A/G)$, $\forall t \in [P^{-1}_{\Sigma\Sigma_{obs}}(P_{\Sigma\Sigma_{obs}}(s)) \cap \mathcal{L}(S_A/G)].\delta_G(x_0, t) \in X_{crit}$.

In this case, we say that f_A is a ***strong attack****. We additionally define the notion of a* ***weak attack*** *as follows:*

3(b). $\exists s \in \mathcal{L}(S_A/G).\delta_G(x_0, s) \in X_{crit}$.

The *Admissibility* condition guarantees that f_A is well defined for all projected strings in the modified controlled language $P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_A/G))$. The *Stealthiness* condition guarantees that the attacker stays undetected by the supervisor, meaning that any string in $P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_A/G))$ should be modified to a string within the original controlled behavior. Lastly, the reachability of critical states is stated in condition 3, where condition 3(a) is a strong version of the problem. In the strong case, the attacker is sure that the system has reached a critical state if string s occurs in the system. Condition 3(b) is a relaxed version, where the attacker might not be sure if a critical

state was reached, although it could have been reached. Both variations of condition 3 guarantee the existence of at least one successful attack, namely, when string s occurs in the new controlled behavior.

Remark 4.1. *Problem **4.1** assumes that the attacker has full knowledge of the plant and the supervisor models. Although it might be difficult to achieve this assumption in practice, our work studies the worst attack scenario case. Moreover, this assumption is common practice within the cyber-security domain.*

Problem **4.1** is posed based on the languages $\mathcal{L}(S_A/G)$ and $\mathcal{L}(R/G)$. Conditions (1), (3a) and (3b) are relatively easy to test using automata operations. On the other hand, condition (2) is more intricate. We could replace condition (2) by Eq. (**4.1**), if we restrict ourselves to automata versions of f_A. However, we explore another solution for this problem so that we can easily test all conditions of Problem **4.1** in a similar manner.

To capture the behavior that detects the attacker, we define an automaton that captures both the control decisions of the supervisor and the normal/abnormal behavior. We start by defining the automaton $H = obs(R||G)$ and $H = (X_H, \Sigma_{obs}, \delta_H, x_{0,H})$, where $X_H \subseteq 2^{X_R \times X_G}$. The automaton H captures only the normal projected behavior of the controlled system. However, H cannot be used as an supervisor since it might contain inadmissible control decisions and it does not have all decisions of R.

Definition 4.2. *Based on H, we define $\tilde{R} := (X_{\tilde{R}} := X_H \cup \{dead\}, \Sigma, \delta_{\tilde{R}}, x_{0,\tilde{R}} := x_{0,R})$, and $\delta_{\tilde{R}}$:*

For any $x \in X_H$

$$\delta_{\tilde{R}}(x,e) := \begin{cases} \delta_H(x,e) & \textit{if } e \in En_H(x) \\ dead & \textit{if } e \in \Sigma_{uctr} \setminus En_H(x) \\ x & \textit{if } e \in \Sigma_{uctr} \cap \Sigma_{uobs} \textit{ or } e \in \Sigma_{ctr} \cap \Sigma_{uobs} \cap En_{R||H}(x) \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{4.1}$$

And $\delta_{\tilde{R}}(dead, e) := dead$ for any $e \in \Sigma_{uctr}$, and undefined otherwise.

Automaton $\tilde{R}$ embeds the same admissible control decisions as automaton R and it differentiates normal/abnormal behavior. The state $dead$ in $\tilde{R}$ captures the abnormal behavior of the controlled system. The attacker remains stealthy as long as $\tilde{R}$ does not reach the state $dead$. Figure 4.3 illustrates the supervisor $\tilde{R}$ computed for supervisor R_1 in Example **2.4**.

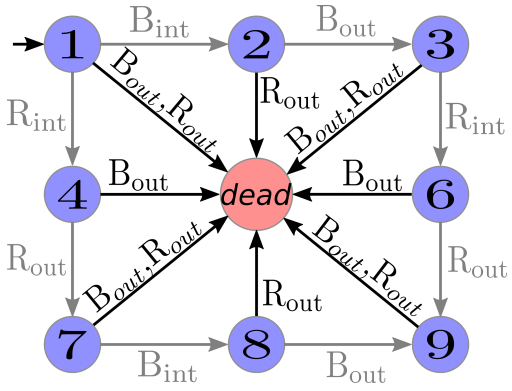


Figure 4.3: Supervisor $\tilde{R}$ based on Supervisor R from Example **2.4**. New transitions are in black and we omit self-loops in state $dead$.

Remark 4.2. *No new transition to the dead state with controllable events are included in the definition of $\tilde{R}$. For simplicity, we assume that the supervisor does not enable controllable events unnecessarily. In this manner, only uncontrollable events can reach the dead state.*

4.3 Insertion deletion attack structure

4.3.1 Definition

An Insertion-Deletion Attack structure (IDA) is an extension of the notion of *bipartite transition structure* presented in [30]. An IDA captures the game between the environment and the supervisor considering the possibility that a subset of the sensor network channels may be compromised by a malicious attacker, whose moves will be constrained according to various rules. In this game we fix the supervisor's decisions as those defined in the given $\tilde{R}$. The environment's decisions are those of the attacker and the system. Therefore, in this game, environment states have both attacker's and system's decisions. In this section, we define the generic notion of an IDA. In the

next sections, we will construct specific instances of it, according to the permitted moves of the attacker.

Intuitively, the IDA represents the framework of supervisory control under sensor deception attack presented in Chapter III. However, the supervisor and the environment decisions are explicitly separated in the IDA. This allows a better understanding of the attacker actions in the controlled system.

In order to build the game, we define an information state as a *pair* $IS \in 2^{X_G} \times X_{\tilde{R}}$, and the set of all information states as $I = 2^{X_G} \times X_{\tilde{R}}$. The first element in an IS represents the *correct state estimate* of the system, as seen by the attacker for the actual system outputs. The second element represents the supervisor's state, which is the current state of its realization based on the edited string of events that it receives. As defined, an IS embeds the necessary information for either player to make a decision.

Definition 4.3. *An Insertion-Deletion Attack structure (IDA) T w.r.t. G, Σ_{sen}, and $\tilde{R}$, is a 7-tuple*

$$T := (Q_S, Q_E, h_{SE}, h_{ES}, \Sigma_{obs}, \Sigma_{att}, y_0) \tag{4.2}$$

where:

- $Q_S \subseteq I$ *is the set of S-states, where S stands for Supervisor and where each S-state is of the form* $y = (I_G(y), I_S(y))$*, where* $I_G(y)$ *and* $I_S(y)$ *denote the correct system state estimate and the supervisor's state, respectively;*

- $Q_E \subseteq I$ *is the set of E-states, where E stands for Environment; each E-state is of the form* $z = (I_G(z), I_S(z))$ *defined in the same way as in the S-states case;*

- $h_{SE} : Q_S \times \Gamma \to Q_E$ *is the partial transition function from S-states to E-states that satisfies:*

$$h_{SE}(y, \gamma)! \Rightarrow h_{SE}(y, \gamma) := (UR_\gamma(I_G(y)),\ I_S(y)) \tag{4.3}$$

- $h_{ES} : Q_E \times (\Sigma_{obs} \cup \Sigma_{att}) \to Q_S$ *is the partial transition function from E-states to S-states, satisfying:*

  For any $e \in \Sigma_{obs}$

$$h_{ES}(z, e)! \Rightarrow \begin{array}{c} e \in En_G(I_G(z)) \cap En_{\tilde{R}}(I_S(z)) \wedge \\ h_{ES}(z, e) := \big(\delta_G(I_G(z), e),\ \delta_{\tilde{R}}(I_S(z), e)\big) \end{array} \tag{4.4}$$

  For any $e \in \Sigma_{sen}$

$$h_{ES}(z, ins(e))! \Rightarrow \begin{array}{c} e \in En_{\tilde{R}}(I_S(z)) \wedge \\ h_{ES}(z, ins(e)) := \big(I_G(z),\ \delta_{\tilde{R}}(I_S(z), e)\big) \end{array} \tag{4.5}$$

$$h_{ES}(z, del(e))! \Rightarrow \begin{array}{c} e \in En_G(I_G(z)) \cap En_{\tilde{R}}(I_S(z)) \wedge \\ h_{ES}(z, e) := \big(\delta_G(I_G(z), e),\ I_S(z)\big) \end{array} \tag{4.6}$$

- Σ_{obs} *is the set of observable events of* G*;*
- Σ_{att} *is the set of attacked events;*
- $y_0 \in Q_S$ *is the initial S-state:* $y_0 := (\{x_{0,G}\}, x_{0,\tilde{R}})$.

Since the purpose of an IDA is to capture the game between the supervisor and the environment, we use a bipartite structure to represent each entity. An S-state is an IS containing the state estimate of the system G and the supervisor's state; it is where the supervisor issues its control decision. An E-state is an IS at which the environment (system or attacker) selects one among the observable events to occur.

A transition from an S-state to an E-state represents the updated unobservable reach in G's state estimate together with the current supervisor state. Note that h_{SE} is only defined for y and γ such that $\gamma = En_{\tilde{R}}(I_S(y))$. On the other hand, a transition from an E-state to an S-state represents the "observable reach" immediately following the execution of the observable event by the environment. In this case, both the system's state estimate and the supervisor's state are updated.

However, these updates depend on the type of event generated by the environment: (i) true system event unaltered by the attacker; (ii) (fictitious) event insertion by the attacker; or (iii) deletion by the attacker of an event just executed by the system. Thus, the transition rules are split into three cases, described below.

The partial transition function h_{ES} is characterized by three cases: Eqs. (4.4-4.6). Each case is coupled to the set of events $\Sigma_{obs}, \Sigma_{ins}$ or Σ_{del}. The case of events in Σ_{obs} is related to *plant's* actions, while the cases of events in Σ_{ins} or Σ_{del} are related to *attacker's* actions. Equation (4.4) simulates the plant generating a feasible (enabled) event and the attacker letting the event reach the supervisor intact, either because it cannot compromise that event, or because it chooses not to make a move. In the case of insertion events, Eq. (4.5), the attacker only inserts events consistent with the control decision of the current supervisor state. On the other hand, the attacker only deletes actual observable events generated by the plant (Eq. (4.6)).

Remark 4.3. *A given IDA contains some attack moves (since it is a generic structure), all of which have to satisfy the constraints in the definition of h_{ES} and h_{SE}. A generic IDA represents a controlled system under sensor deception attack given a specific attack function.*

We assume that all states included in an IDA are reachable from its initial state.

Example 4.1. *Let us consider the intersection example (Example **2.4**) with supervisor $\tilde{R}$ as in Fig. 4.3. Considering the compromised event set $\Sigma_{sen} = \{R_{int}\}$ and $X_{crit} = \{5\}$, Fig. 4.4 gives one IDA example. As we mentioned, this IDA represents an attacked controlled system given a specific attack function. We can observe from Fig. 4.4 that the attack function in this example has $f_A(\epsilon, \epsilon) = \{\epsilon, ins(R_{int})\}$ as its initial condition. This attack function is not stealthy since the $dead$ state is reachable. We will address this problem in the next section.*

4.3.2 Properties

Given two IDAs T_1 and T_2, we say that T_1 is a subsystem of T_2, denoted by $T_1 \sqsubseteq T_2$, if $Q_E^{T_1} \subseteq Q_E^{T_2}$, $Q_S^{T_1} \subseteq Q_S^{T_2}$, and for any $y \in Q_S^{T_1}$, $z \in Q_E^{T_1}$, $\gamma \in \Gamma$, and $e \in \Sigma_{obs} \cup \Sigma_{att}$, we have that:

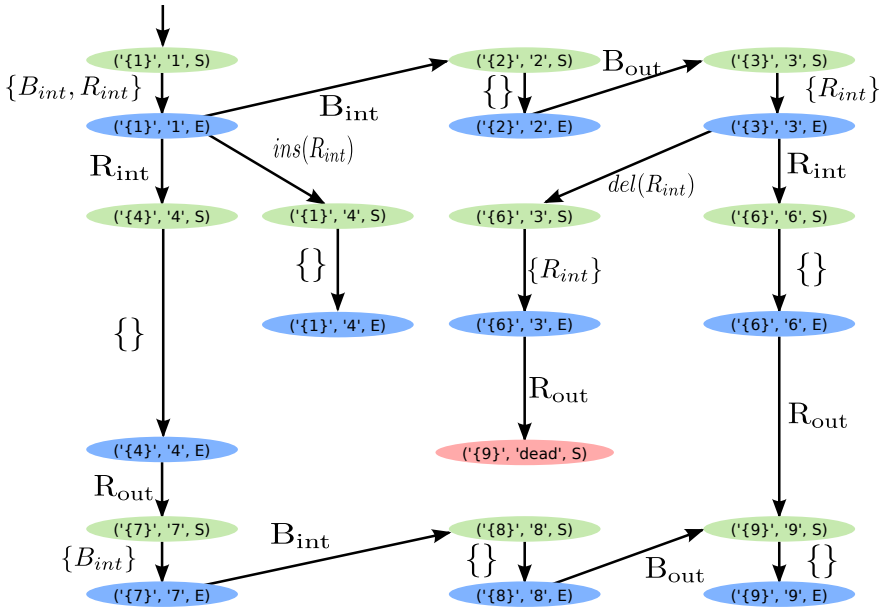


Figure 4.4: IDA, intersection example with $\Sigma_{sen} = \{R_{int}\}$. States with *S* are supervisor states (most in light green) while states with *E* are environment (most in light blue). States in red are states where the attacker is discovered. Uncontrollable events are omitted from the control decisions, i.e., $\{\} := \{R_{out}, B_{out}\}$.

1. $h_{SE}^{T_1}(y, \gamma) = z \Rightarrow h_{SE}^{T_2}(y, \gamma) = z$ and

2. $h_{ES}^{T_1}(z, e) = y \Rightarrow h_{ES}^{T_2}(z, e) = y$.

Before, we discuss relevant properties of the IDA structure, we define a property about E-states.

(P1) An E-state $z \in Q_E$ is called a *race-free state*, if the following condition holds:

$$(e \in En_{\tilde{R}}(I_S(z)) \cap En_G(I_G(z)) \cap \Sigma_{obs}) \Rightarrow [h_{ES}(z, e)! \vee h_{ES}(z, del(e))!]$$

Property (P1) ensures the non-existence of a race condition between the attacker and the system in a given E-state. Specifically, when (P1) holds in any E-state, the attacker has the option of either letting the event reach the supervisor intact or preventing the event from reaching the supervisor (or allowing both). It means that the attacker can wait for the system's response to the most recent control action and react accordingly. Note that if the event in question is a compromised event, then the attacker has the option of allowing both actions since we defined nondeterministic attack functions. In the case of deterministic attack functions, we force the attacker to select one of the

actions. If (P1) does not hold at a particular E-state, then the attacker *must insert an event*, and this insertion must take place before the system reacts to the most recent control action sent by the supervisor. (In some sense, the attacker is "racing" with the system.) An IDA that satisfies (P1) for all E-states is called a *race-free* IDA.

Definition 4.4 (Induced E-state)**.** *Given an IDA A, $IE(z,s)$ is defined to be the E-state induced by string $s \in (\Sigma_{obs} \cup \Sigma_{att})^*$, when starting in the E-state z. $IE(z,s)$ is computed recursively as:*

$$IE(z,\epsilon) := z$$
$$IE(z,se) := \begin{cases} h_{SE}\big(y, En_{\tilde{R}}\big(I_S(y)\big)\big) & \text{if } h_{ES}(IE(z,s),e)! \\ \text{undefined} & \text{otherwise} \end{cases} \tag{4.7}$$

where $y = h_{ES}(IE(z,s),e)$

We also define $IE(s) := IE(z_0,s)$, where $z_0 = h_{SE}(y_0, En_{\tilde{R}}(I_S(y_0)))$.

We conclude this section by defining the notion of *embedded* attack function in an IDA and presenting a lemma about reachability in IDAs. First, we show that after observing an edited string t, the attacker correctly keeps track of the supervisor state $\tilde{R}$. Then we show that an IDA correctly estimates the set of possible states of the system after the occurrence of edited string t.

Definition 4.5. *An attack function f_A is said to be* ***embedded in an IDA*** *T if $\big(\forall s \in P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_A/G))\big)\big(\forall t \in \hat{f}_A(s)\big)\big(\forall i \in \mathbb{N}^{|t|-1}\big)$, then $IE(t^i)$ is defined.*

Intuitively, an attack function is embedded in an IDA if for every modified string that is consistent with the behavior of G, we can find a path in the IDA for that string. Note that we limit this constraint on f_A to strings in $P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_A/G))$, since these are the ones consistent with G under a given f_A. We are not interested in the definition of any f_A for strings outside of the controlled behavior f_A defines.

Next, we provide a lemma that ties the definition of IDA with the attacked controlled system defined in Chapter III. Previously, we informally stated this connection between the IDA and a

specific attack function. Actually, we can generalize this statement based on the definition of embedded attack functions. Intuitively, the IDA captures the attacked controlled system for a family of attack functions, i.e., the attack functions embedded in this IDA. This statement is formalized in the following lemma. This lemma states that the information states in the IDA carry the same information as the attacked controlled system.

Lemma 4.1. *Given a system G, supervisor $\tilde{R}$, and an IDA structure T with an embedded attack function f_A with automaton representation A, for any string $s \in \mathcal{L}(S_A/G)$ and any string $t \in \hat{f}_A(P_{\Sigma\Sigma_{obs}}(s))$, we have*

$$I_S\big(IE(t)\big) = \delta_{\tilde{R}}\big(x_{0,\tilde{R}}, P^S(t)\big) \tag{4.8}$$

$$I_G\big(IE(t)\big) = RE_{G_a}\big(t, \mathcal{L}(G_a||A||R_a)\big) \tag{4.9}$$

Proof. We prove the result by induction on the length of t. First, we show Eq. (4.8). Let $|t| = n$. Let y_0 be defined as usual, $q_i = \delta_{\tilde{R}}(x_{0,\tilde{R}}, P^S(t^i))$, $z_i = h_{SE}(y_i, En_{\tilde{R}}(q_i))$ for $i \in \mathbb{N}^n$, and $y_{i+1} = h_{ES}(z_i, t[i+1])$ for $i \in \mathbb{N}^{n-1}$.

Induction Basis: $t = \epsilon$

$$IE(\epsilon) = h_{SE}(y_0, En_{\tilde{R}}(I_S(y_0)))$$

We recall that the supervisor state does not change in the IDA from S-states to E-states since it did not receive any observable event, thus $y_k \to z_k \Rightarrow I_S(y_k) = I_S(z_k)$, then

$$I_S(IE(\epsilon)) = x_{0,\tilde{R}} = \delta_{\tilde{R}}(x_{0,\tilde{R}}, \epsilon)$$

Induction hypothesis: Assume that $I_S(IE(t^i)) = q_i$ holds for $i \in \mathbb{N}^k$, $k < n$.

Induction step: At $k+1$ we have

$$y_{k+1} = h_{ES}(z_k, t[k+1])$$

And we know that

$$I_S(z_k) = q_k$$

$$I_S(y_{k+1}) = \delta_{\tilde{R}}(I_S(z_k), P^S(t[k+1])) = \delta_{\tilde{R}}(x_{0,\tilde{R}}, P^S(t^{k+1}))$$

Given that $I_S(y_{k+1}) = I_S(z_{k+1})$, then $I_S(z_{k+1}) = \delta_{\tilde{R}}(x_{0,\tilde{R}}, P^S(t^{k+1}))$. Consequently, $I_S(IE(t^{k+1})) = \delta_{\tilde{R}}(x_{0,\tilde{R}}, P^S(t^{k+1}))$.

Next, we also show Eq. (4.9) by induction on t. We use the same notation as in the previous induction proof. Moreover, we use $L := \mathcal{L}(G_a||A||R_a)$ and $\Sigma_{m,o} := \Sigma_{obs} \cup \Sigma_{att}$.

Induction Basis: $t = \epsilon$ By the definition of the IDA, we have

$$I_G(IE(\epsilon)) = UR_{En_{\tilde{R}}(x_{0,\tilde{R}})}(\{x_{0,G}\})$$

and by the definition of RE_{G_a}, we have

$$\begin{aligned}
RE_{G_a}(\epsilon, L) &= \bigcup_{\substack{s \in L: \\ P_{\Sigma_{all}\Sigma_{m,o}}(s)=\epsilon}} \delta_{G_a}(\{x_{0,G_a}\}, s) \\
&= \bigcup_{s \in (En_{\tilde{R}}(x_{0,\tilde{R}}) \cap \Sigma_{uobs})^*} \delta_{G_a}(\{x_{0,G_a}\}, s) \text{ by def. of } R_a, A \text{ and } G_a||A||R_a \\
&= \bigcup_{s \in (En_{\tilde{R}}(x_{0,\tilde{R}}) \cap \Sigma_{uobs})^*} \delta_{G}(\{x_{0,G}\}, s) \text{ by def. of } G_a \\
&= UR_{En_{\tilde{R}}(x_{0,\tilde{R}})}(\{x_{0,\tilde{R}}\})
\end{aligned}$$

Induction hypothesis: Assume that $I_G(IE(t^i)) = RE_{G_a}(t^i, L)$ holds for $i \in \mathbb{N}^k$, $k < n$.

Induction step: At $k+1$ we have

$$y_{k+1} = h_{ES}(z_k, t[k+1]) = (\delta_G(I_G(z_k), P^G(t[k+1])), \delta_{\tilde{R}}(I_S(z_k), P^S(t[k+1])))$$

Therefore, $I_G(z_{k+1}) = UR_{En_{\tilde{R}}(q_{k+1})}(\delta_G(I_G(z_k), t[k+1]))$. It can be shown that

$$RE_{G_a}(t^{k+1}, L) = UR_{En_{\tilde{R}}(q_{k+1})}(\delta_G(RE_{G_a}(t^k, L), t[k+1]))$$

This concludes our proof. □

4.4 All insertion deletion attack structure

In this section, we define the *All* Insertion-Deletion Attack Structure, a specific type of IDA abbreviated as AIDA hereafter. The AIDA embeds *all* insertion-deletion actions the attacker is able to execute. In order words, the AIDA embeds the "all-out" strategy. We discuss its construction and properties.

4.4.1 Definition

As consequence of Lemma **4.1**, if we construct an IDA structure based on $\tilde{R}$ and Σ_{sen} that is "as large as possible", then it will include all valid insertion and deletion actions for the attacker. We formally define such a structure as the All Insertion-Deletion Attack structure.

Definition 4.6 ($\mathsf{AIDA}(G, \tilde{R}, \Sigma_{sen})$)**.** *Given a system G, a supervisor $\tilde{R}$, and a set of compromised events Σ_{sen}, the **All Insertion-Deletion Attack structure (AIDA)**, denoted by $\mathsf{AIDA}(G, \tilde{R}, \Sigma_{sen}) = (Q_S, Q_E, h_{SE}, h_{ES}, \Sigma, \Sigma_{att}, y_0)$, is defined as the **largest** IDA w.r.t to G, $\tilde{R}$ and Σ_{sen} s.t.*

1. *For any $y \in Q_S$, we have $|\Gamma_{AIDA}(y)| = 0 \Leftrightarrow I_S(y) =$ dead*
2. *For any $z \in Q_E$, we have*
   - *(a) $\forall e \in En_{\tilde{R}}(I_S(z)) \cap En_G(I_G(z)) \cap \Sigma_{obs} : (h_{ES}(z, e)! \vee h_{ES}(z, del(e))!)$ or*
   - *(b) $I_G(z) \subseteq X_{crit} \Rightarrow |\Gamma_{AIDA}(z)| = 0$*

Condition 2(a) alone satisfies the non-existence of a race condition in the AIDA. Conditions 1 guarantees that the AIDA stops its search once the attack is detected. Similarly, Condition 2(b) stops the search given that the attacker knows it has reached its goal.

By "largest" structure, we mean that for any IDA T satisfying the above conditions: $T \sqsubseteq \mathsf{AIDA}(G, \tilde{R}, \Sigma_{sen})$. This notion of "largest" IDA is well defined. If T_1 and T_2 are two IDA structures satisfying the above conditions, then their union still satisfies the conditions, where the union $T_1 \cup T_2$ is defined as: $Q_E^{T_1 \cup T_2} = Q_E^{T_1} \cup Q_E^{T_2}$, $Q_S^{T_1 \cup T_2} = Q_S^{T_1} \cup Q_S^{T_2}$, and for any $y \in Q_E^{T_1 \cup T_2}$, $z \in Q_S^{T_1 \cup T_2}$, $\gamma \in \Gamma$ and $e \in \Sigma_{obs} \cup \Sigma_{att}$, we have that $h_{SE}^{T_1 \cup T_2}(y, \gamma) = z \Leftrightarrow \exists i \in \{1, 2\} : h_{SE}^{T_i}(y, \gamma) = z$ and $h_{ES}^{T_1 \cup T_2}(z, e) = y \Leftrightarrow \exists i \in \{1, 2\} : h_{ES}^{T_i}(z, e) = y$.

4.4.2 Construction

The construction of the AIDA follows directly from its definition. It enumerates all possibles transitions for each state by a breadth-first search. Each S-state has at most one control decision, which is related to the supervisor state $\tilde{R}$. On the other hand, each E-state enumerates both system's and attacker's actions, according to its system and supervisor estimate, respectively. In practice, we do not need to search the entire state space; we stop the branch search when an S-state reaches a state with the supervisor at the "dead" state or when the system estimate of an E-state is a subset of X_{crit}.

The procedure mentioned above is described in Algorithm 1 (Construct-AIDA). It has the following parameters: $AIDA$ is the graph structure of the AIDA we want to construct; $AIDA.E$ and $AIDA.S$ are the E- and the S-state sets of the structure, respectively; $AIDA.h$ is its transition function; Q is a queue. We begin the procedure by initializing $AIDA.S$ with a single element $y_0 = (\{x_{0,G}\}, x_{0,\tilde{R}})$. The breath-first search is then performed by the procedure DoBFS. The transitions between S-states and E-states are dealt within lines 7 to 11. The transitions between E-states and S-states are defined in lines 12 to 25, where each attack possibility is analyzed. These transitions are defined exactly as in Definition **4.3**, Eqs. (4.4-4.6). (For the sake of readability, we employ the usual triple notation (origin, event, destination) for the transition function.) The

procedure converges when all uncovered states (states in the queue) are covered, meaning we have traversed the whole reachable space of E- and S-states. Note that lines 29 and 33 impose the stop conditions of Definition **4.6**.

Theorem 4.1. *Algorithm Construct-AIDA correctly constructs the AIDA.*

Proof. Conditions 1 and 2 of Definition **4.6** follow directly by the construction of the Algorithm 1 (lines 14, 29, 30). Thus, we only need to prove that the IDA returned by the algorithm is the largest one. We show it by contradiction.

Assume that A is the IDA returned by the algorithm; however assume that $\exists A^*$ that satisfies conditions 1 and 2, where $A \sqsubset A^*$. It means that $Q_S^A \subseteq Q_S^{A^*}$ or $Q_E^A \subseteq Q_E^{A^*}$. If A and A^* have the same states, then either $A = A^*$ or A^* has more transitions than A. The first case is a contradiction of our arguments. The second case implies that some transitions were not included in A, which is also a contradiction. Algorithm 1 applies an exhaustive BFS therefore it cannot leave it out any transition if all states where covered. Thus, it is the case that A^* has more states than A. Let us start with A^* having one additional E-state namely z, then $Q_S^A = Q_S^{A^*}$. Therefore, $\exists y \in Q_S^A$ such that $y \rightarrow z$, which means that $I_S(y) \neq dead$. Therefore, Algorithm 1 will not converge to A, contradicting our assumption. The same reasoning can be used for the case of one more S-state or when both sets are larger. □

Example 4.2. *We return our intersection example using the same parameters as in Example **4.1**. The IDA shown in Fig. 4.4 **is not** the AIDA for this example since it clearly misses some attacker transitions. The AIDA for this example is depicted in Fig. 4.5. Note that there exists an S-state where $\tilde{R}$ reaches the state* dead*. Moreover, there exists an E-state where G reaches a critical state.*

Remark 4.4. *The AIDA has at most $2^{|X_G|+1}|X_{\tilde{R}}|$ states since $Q_1, Q_2 \subseteq I$. If $\Sigma_{uobs} = \emptyset$, then it has at most $2|X_G||X_{\tilde{R}}|$ states.*

Algorithm 1 Construct-AIDA

Require: G, $\tilde{R}$ and Σ_{sen}
Ensure: $AIDA$

$AIDA \leftarrow$ DoBFS$\big(G, \tilde{R}, (\{x_{0,G}\}, x_{0,\tilde{R}})\big)$
procedure DoBFS(G, R, y)
    $AIDA.S \leftarrow \{y\}$, $AIDA.E \leftarrow \emptyset$, $AIDA.h \leftarrow \emptyset$
    Queue $Q \leftarrow \{y\}$
    while Q is not empty **do**
        $c \leftarrow Q.dequeue(\,)$
        if $c \in AIDA.S$ **then**
            $\gamma \leftarrow En_{\tilde{R}}(I_S(c))$
            $z \leftarrow \big(UR_\gamma\big(I_G(c)\big), I_S(c)\big)$
            $AIDA.h \leftarrow AIDA.h \cup \{(c, \gamma, z)\}$
            Add-State-to-AIDA($z, AIDA, Q$)
        else if $c \in AIDA.E$ **then**
            for all $e \in \Sigma_{obs} \cap En_{\tilde{R}}\big(I_S(c)\big)$ **do**
                if $e \in En_G\big(I_G(c)\big)$ **then**
                    $y \leftarrow \Big(\delta_G(I_G(c), e),\ \delta_{\tilde{R}}(I_S(c), e)\Big)$
                    $AIDA.h \leftarrow AIDA.h \cup \{(c, e, y)\}$
                    Add-State-to-AIDA($y, AIDA, Q$)
                if $e \in En_G\big(I_G(c)\big) \wedge e \in \Sigma_{sen}$ **then**
                    $y \leftarrow \big(\delta_G\big(I_G(c), e\big),\ I_S(c)\big)$
                    $AIDA.h \leftarrow AIDA.h \cup \{(c, del(e), y)\}$
                    Add-State-to-AIDA($y, AIDA, Q$)
                if $e \in \Sigma_{sen}$ **then**
                    $y \leftarrow \big(I_G(c),\ \delta_{\tilde{R}}\big(I_S(c), e\big)\big)$
                    $AIDA.h \leftarrow AIDA.h \cup \{(c, ins(e), y)\}$
                    Add-State-to-AIDA($y, AIDA, Q$)
procedure ADD-STATE-TO-AIDA($c, AIDA, Q$)
    if $c \notin AIDA.E \wedge c$ is an E-state **then**
        $AIDA.E \leftarrow AIDA.E \cup \{c\}$
        if $I_G(c) \nsubseteq X_{crit}$ **then**
            $Q.enqueue(c)$
    else if $c \notin AIDA.S \wedge\ c$ is an S-state **then**
        $AIDA.S \leftarrow AIDA.S \cup \{c\}$
        if $I_S(c) \neq$ dead **then**
            $Q.enqueue(c)$

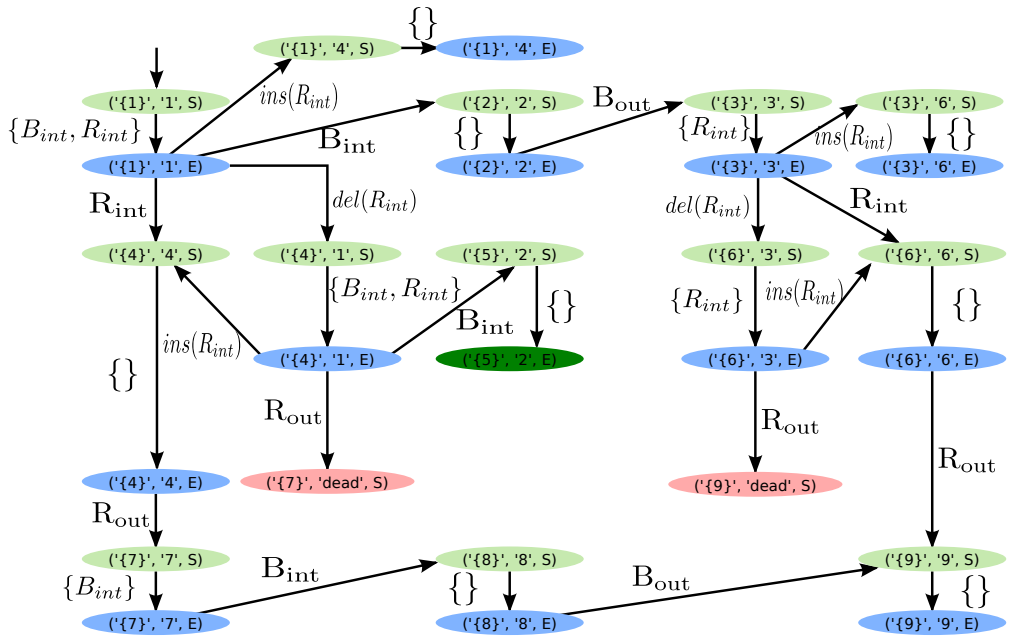


Figure 4.5: AIDA, intersection example with $\Sigma_{sen} = \{R_{int}\}$. States with *S* are supervisor states (most in light green) while states with *E* are environment (most in light blue). States in red are states where the attacker is discovered while states where the attacker reaches the dead state are in dark green.

4.5 Pruning the all insertion deletion attack structure

The AIDA embeds all attack functions, including non-stealthy strategies, i.e., those that lead to state *dead* of the supervisor. Since our goal is to obtain stealthy attack functions, we "eliminate" non-stealthy strategies from the AIDA. Here, eliminating means pruning the AIDA such that only stealthy strategies are embedded (Def. **4.5**). Lastly, we are left with the task of extracting an attack function from the pruned structured, if they exists. Namely, we are able to answer Problem **4.1** based on the pruned structure.

Although our goal of removing non-stealthy strategies and our pruning the AIDA idea method are clear, this pruning procedure must be defined carefully. The pruning procedure depends on the attacker capabilities and interaction with the controlled system. For example, if the attacker is unsure about the plant responsiveness, then synthesizing an interruptiple attack function (Def. **3.6**) seems ideal for this scenario. For this reason, the pruning procedure can be tailored for different attacker assumptions.

In Chapter III, we have provided three different attack functions based on different attacker constraints: interruptible attack functions (Def. **3.6**), deterministic unbounded attack function (Def. **3.7**), and deterministic bounded attack function (Def. **3.8**). In this section, we provide pruning procedures for these three attack scenarios as well as how to extract attack functions, if they exist, that are a solution of Problem **4.1**.

4.5.1 Synthesis of interruptible attack strategies

The AIDA could reach state *dead* of supervisor $\tilde{R}$. Each time this occurs, it means that the last step of the attack is no longer stealthy. Hence, we must prune the AIDA in order to embed only stealthy attacks. We pose this pruning process as a *meta-supervisory-control* problem, where the "plant" is the entire AIDA, the specification for that plant is to prevent reaching state *dead* of the supervisor, and the *controllable* events are the actions of the attacker. We assume that the reader is familiar with the standard "Basic Supervisory Control Problem" of [35]; we adopt the presentation of that problem as BSCP in [34]. First, we show necessary modifications to the standard BSCP algorithm in order to construct the Stealthy-AIDA.

Algorithm 2 Modified BSCP for Interruptible Attacker

Require: $B := (X_B, E := \Sigma_{obs} \cup \Sigma_{att} \cup \Gamma, \delta_B, x_{0,B})$, E_{uctr} and $B_t = (X_{B_t}, E, \delta_{B_t}, x_{0,B_t})$, where $X_{B_t} \subseteq X_B$

1: **Step 1** Set $H_0 := (X_{H_0}, E, \delta_{H_0}, x_{0,H_0}) = B_t$, and $i = 0$
2: **Step 2** Calculate
3: **Step 2.1** $X'_{H_i} = \{x \in X_{H_i} | En_B(x) \cap E_{uctr} \subseteq En_{H_i}(x)\}$
4: **Step 2.2** $X^*_{H_i} = \{x \in X'_{H_i} | e \in En_B(x) \Rightarrow (e \in En_{H_i}(x) \vee del(e) \in En_{H_i}(x)\}$
$\delta'_{H_i} = \delta_{H_i} | X^*_{H_i}$ [transition function update]
5: **Step 2.3** $H_{i+1} = Ac(X^*_{H_i}, E, \delta'_{H_i}, x_{0,H_i})$
6: **Step 3** If $H_{i+1} = H_i$, Stop; otherwise $i \leftarrow i + 1$, back to Step 2

The difference between the original BSCP algorithm (see, e.g., [34]) and its modified version in Algorithm 2 is the addition of Step 2.2. Since the BSCP algorithm has quadratic worst-time complexity in the number of states of the automaton B_t, it follows that Algorithm 2 also has quadratic worst-time complexity. In order to ensure the desired interruptibility condition of an attack function, each visited state in the AIDA must be race free. Therefore, at Step 2.2 we enforce

the race-free condition at every E-state of the IDA. In order to enforce such condition, the algorithm deletes E-states where both the "let through" transition *and* the "erasure" transition are absent for a feasible system event. Therefore, the resulting IDA from Algorithm 2 is a race-free IDA.

To compute the stealthy AIDA structure, we define as system the AIDA constructed according to Algorithm 1. Moreover, any event $e \in \Sigma_{sen} \cup \Sigma_{all}$ is treated as *controllable* while any control decision $\gamma \in \Gamma$ and any event $e \in \Sigma_{obs} \backslash \Sigma_{sen}$ is treated as *uncontrollable*. The specification language, realized by B_t, is obtained by deleting the states where the supervisor reaches the dead state, i.e., by deleting in the AIDA all states of the form $y = (S, \text{dead})$ for any $S \subseteq X_G$.

We formalize the pruning process for obtaining all stealthy insertion-deletion attacks as follows.

Definition 4.7. *Given the AIDA constructed according to Algorithm 1, define the automaton* $B := (Q_E^{AIDA} \cup Q_S^{AIDA}, E := \Sigma_{obs} \cup \Sigma_{att} \cup \Gamma, h_{ES}^{AIDA} \cup h_{SE}^{AIDA}, y_0^{AIDA})$ *with* $E_{ctr} = \Sigma_{sen} \cup \Sigma_{att}$ *as the set of controllable events and* $E_{uctr} = (\Sigma_{obs} \backslash \Sigma_{sen}) \cup \Gamma$ *as the set of uncontrollable events. The specification automaton is defined by* B_t*, which is obtained by trimming from* B *all its states of the form* $(S, dead)$*, for any* $S \subseteq X_G$*. The* Stealthy AIDA structure*, called the ISDA (Interruptible Stealthy Deceptive Attack), is defined to be the automaton obtained after running Algorithm 2 on* B_t *with respect to* B *and* E_{uctr}*.*

Example 4.3. *We return to the* $AIDA$ *in Fig 4.5, where we would like to obtain the ISDA as in Def. 4.7. Let us go over the Def. 4.7 and the Algorithm 2 in a step by step manner. The automaton* B *is just a copy of the* $AIDA$ *while* B_t *is defined by deleting states* $(\{7\}, dead, S), (\{9\}, dead, S)$ *from* B*. Next, we feed* B *and* B_t *to Algorithm 2. In the first iteration, Step 2.1 deletes states* $((\{4\}, 1, E), (\{6\}, 3, E))$ *since they are reached by event* $R_{out} \in E_{uc}$*. Next, Step 2.3 deletes states* $((\{5\}, 2, S), (\{5\}, 2, E))$ *since they have just become unreachable.*

The second iteration starts and Step 2.1 deletes states $((\{4\}, 1, S), (\{6\}, 3, S))$ *since they are reached by events* E_{uc}*. No more states are deleted in the rest of the second iteration and the third iteration since states* $((\{4\}, 1, S), (\{6\}, 3, S))$ *were reached by event* $del(R_{int})$*. Therefore, the algorithm converges to a fixed point that is depicted in Fig. 4.6.*

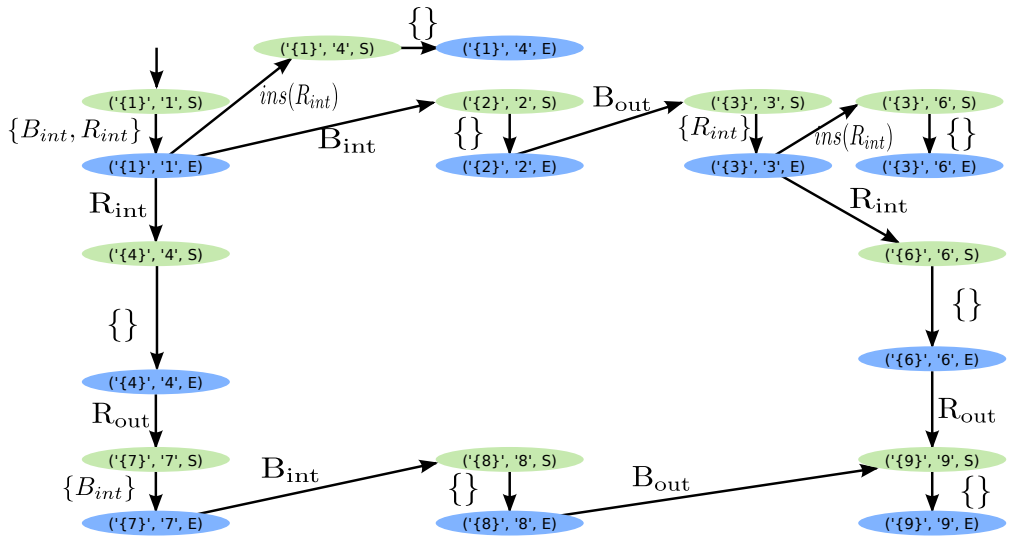


Figure 4.6: ISDA, intersection example with $\Sigma_{sen} = \{R_{int}\}$.

Unfortunately, state $(\{5\}, 2, E)$ that reaches the critical state 5 got deleted in the pruning process. Next, we show that the lack of these states in ISDA guarantees the non-existence of interruptible attack functions that are a solution of Problem **4.1**. On the other hand, if at least one of these state is spared in the pruning process, then there exists an interruptible attack function that solves Problem **4.1**. We also show conditions for when the solution gives a strong attack function or a weak attack function.

Lemma 4.2. *If an interruptible attack function f_A satisfies the admissibility and the stealthy conditions from Problem **4.1**, then it is embedded in the ISDA.*

Proof. By contradiction, assume that we have an interruptible f_A that is admissible and stealthy, but is not embedded in the ISDA. There exists a string $s \in \mathcal{L}(S_A/G)$, $s_a \in \hat{f}_A(P_{\Sigma\Sigma_{obs}}(s))$ where $IE(s_a)$ is not defined in the ISDA. For simplicity and without loss of generality, assume that $s_a = t_a e$ where $e \in \Sigma_{obs} \cup \Sigma_{att}$, and $IE(t_a)$ is defined but $IE(s_a)$ is not defined. There are two reasons why $IE(s_a)$ is not defined.

1. $IE(s_a)$ is not defined in the AIDA. We have that $z = IE(t_a)$ in the AIDA, however $IE(z, e)$ is not defined. Based on the construction of the AIDA, at state z the transition function is exhaustively constructed given $En_{\tilde{R}}(I_S(z))$. Therefore, if $IE(z, e)$ is not defined, then

$\mathcal{M}(e) \notin En_{\tilde{R}}(I_S(z))$. Since $\tilde{R}$ is admissible, $\mathcal{M}(e) \in \Sigma_{ctr} \cap \Sigma_{obs}$ and $e \in \Sigma_{ins}$. As consequence of Lemma **4.1**, $P^S(s_a) \notin P_{\Sigma\Sigma_{obs}}(\mathcal{L}(S_P/G))$, which trivially violates stealthiness. This contradicts our assumption of stealthy f_A. Note that, this case is different than reaching the dead state. In this case, the attacker inserts an event that is not allowed by the supervisor.

2. $IE(s_a)$ is defined in the AIDA but it was pruned by Algorithm 2. Algorithm 2 returns the "supremal controllable sublanguage" of the AIDA, i.e., it is maximally permissive, under the race-free and controllability conditions. It removes all sequences that are non-stealthy or that uncontrollably lead to non-stealthiness. Similarly for the race-free condition. Moreover, one cannot define an interruptible attack decision at a state that is not race free. Finally, the definition of the set of controllable events guarantees admissibility. Thus, overall, s_a must lead to a non-stealthy, non-interruptible, or inadmissible strategy, which makes the function f_A also either non-stealthy, non-interruptible, or inadmissible, a contradiction.

This completes the proof. □

Lemma 4.3. *If an interruptible attack function f_A is constructed from the ISDA, then the stealthy condition of Problem **4.1** is satisfied.*

Proof. The proof follows directly by the construction of the ISDA. □

Constructing an f_A from the ISDA means selecting decisions at E-states and properly defining f_A from the selected decisions.

Remark 4.5. *Lemma **4.3** does not guarantee the admissibility condition. An inadmissible interruptible f_A could be synthesized from the ISDA. However an admissible interruptible f_A can always be synthesized from the ISDA. Since the ISDA is race-free, an inadmissible f_A synthesized from the ISDA can always be extended to make it admissible.*

Theorem 4.2. *The ISDA embeds all possible interruptible stealthy insertion-deletion attack strategies with respect to Σ_{sen}, $\tilde{R}$ and G.*

Proof. The proof follows from Lemmas **4.2** and **4.3**. □

Based on the ISDA, we can synthesize interruptible attack functions that satisfy the admissibility and the stealthy conditions. In order to fully satisfy Problem **4.1**, we need to address its last condition about the existence of strong or weak attacks.

Theorem 4.3. *Given the ISDA, there exists an interruptible f_A that* strongly *satisfies Problem **4.1** if and only if there exists an E-state z in the ISDA s.t. $I_G(z) \subseteq X_{crit}$. On the other hand, it* weakly *satisfies Problem **4.1** if and only if there exists an E-state z in the ISDA s.t. $\exists x \in I_G(z)$, $x \in X_{crit}$.*

Proof. The proof follows from Theorem **4.3**. □

Theorem **4.3** gives necessary and sufficient conditions for synthesis of interruptible attack functions. The following algorithm (Algorithm 3) synthesizes a simple interruptible attack function that satisfies Problem **4.1**. Specifically, Algorithm 3 encodes an interruptible attack function in automaton A. The encoded function simply includes *one* attempt to reach a given critical state. First, it computes the shortest path from the initial state to a critical state via the function *Shortest-Path*$(ISDA, z \in Q_E^{ISDA})$ such that $I_G(z) \subseteq X_{crit}$ (strong attack) or $I_G(z) \cap X_{crit} \neq \emptyset$ (weak attack). The second step is to expand the function, based on the shortest path, in order to satisfy the admissibility condition.

Remark 4.6. *We presented a simple synthesis algorithm. Clearly, one could choose another strategy to extract an interruptible function from the ISDA. The important point here is that the ISDA provides a representation of the desired "solution space" for the synthesis problem. We are not focused on the synthesis of minimally invasive attack strategies as studied in [13], where minimally invasive means an attack strategy with the least number of edits to reach a critical state.*

Example 4.4. *Unfortunately*[2]*, we saw that the ISDA in Example **4.3** prunes the critical state since the attacker could be discovered. Algorithm 3 cannot be used since there is not a single stealthy attack in this example.*

To demonstrate the attacker extraction, we slightly modify our Example by assuming that $\Sigma_{ctr} = \Sigma$. The supervisor for this scenario is obtained by simply deleting state 5 from G, i.e.,

[2]Remember, we are placing ourselves in the attacker's shoes.

Algorithm 3 Synthesis-f_A

Require: $ISDA$, $z \in Q_E^{ISDA}$ s.t. $I_G(z) \subseteq X_{crit}$ or $I_G(z) \cap X_{crit} \neq \emptyset$
Ensure: Encoded f_A in automaton A

$path \leftarrow$ Shortest-Path$(ISDA, z)$
$A \leftarrow$Expand-Path$(path, ISDA)$
procedure EXPAND-PATH$(path, ISDA)$
    Queue $Q \leftarrow \emptyset$; $A \leftarrow \emptyset$
    $A.X \leftarrow \{y_0^{ISDA}\}$; $A.x_0 = y_0^{ISDA}$
    $A.\delta \leftarrow \emptyset$; $A.\Sigma = \Sigma_{all}$
    for all $(q, e, f) \in path \wedge q$ is an E-State **do**
        $t := IE(q, e)$
        Add(q, e, f, Q, A)
    while Q is not empty **do**
        $q \leftarrow Q.dequeue()$
        for all $(q, e, f) \in ISDA.h \wedge (q, e, f) \notin A.h$ **do**
            if $e \in \Sigma_{obs} \wedge \nexists f^*$ s.t. $(q, del(e), f^*) \in A.h$ **then**
                $t := IE(q, e)$
                Add(q, e, t, Q, A)
            else if $e \in \Sigma_{del} \wedge (q, \mathcal{M}(e), f^*) \notin ISDA.h$ **then**
                $t := IE(q, e)$
                Add(q, e, t, Q, A)
procedure ADD(q, e, t, Q, A)
    $A.\delta \leftarrow A.\delta \cup \{(q, e, t)\}$
    if $f \notin Q$ **then**
        $Q.enqueue(t)$
        $A.X \leftarrow A.X \cup \{t\}$

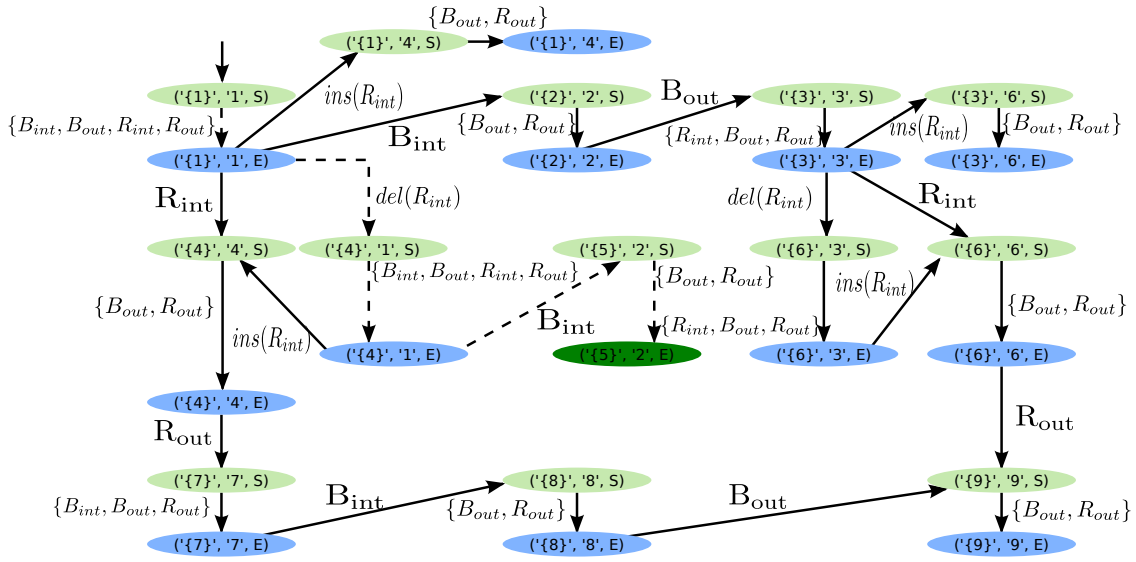
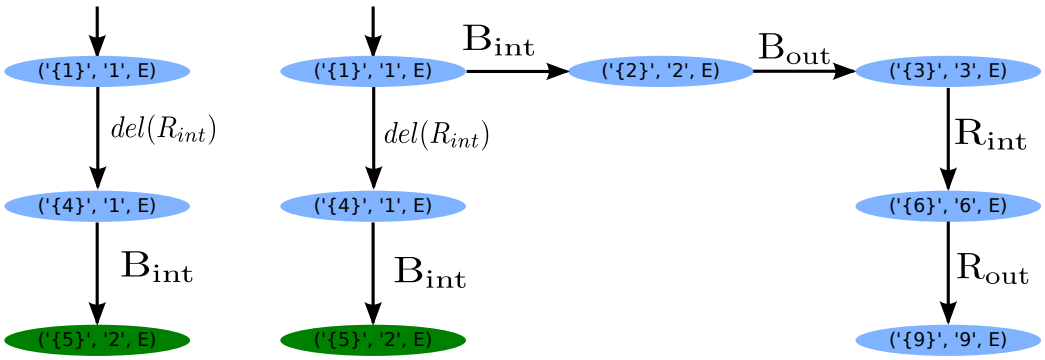


Figure 4.7: ISDA, intersection example with $\Sigma_{ctr} = \Sigma$ and $\Sigma_{sen} = \{R_{int}\}$.

$R = Ac(G, \{5\})$.

The ISDA for this scenario is depicted in Fig. 4.7. Note that the critical state is part of this ISDA, which implies that we can find a stealthy interruptible attack function that reaches this state. In other words, the attack can covertly cause a car collision. Next, we show how this attack function is extracted via Algorithm 3.

First, Algorithm 3 finds a path to the critical state. This path is depicted by the dashed transitions. Next, each environment state in this path is added to automaton A. This partially defined A is shown in Fig. 4.8(a)



(a) Unadmissible attack function

(b) Admissible attack function

Figure 4.8: Attack extraction

At this point, the attack strategy A in Fig. 4.8(a) is not admissible since $f_A(\epsilon, B_{int})$. Admissibility is taken care by the while loop starting in line 9 in Algorithm 3. The output of Algorithm 3 is depicted in Fig. 4.8(b)

4.5.2 Synthesis of unbounded deterministic attack strategies

Different from the interruptible attack, in the deterministic unbounded (det-unb) attack case, we do not need to consider that the system may interrupt during an attack insertion. The attacker can insert events, possibly an arbitrarily long string in fact, before the system reacts. As consequence, the pruned IDA for the det-unb attack is not necessarily race free.

Algorithm 2 prunes the AIDA enforcing it to be race free (Step 2.2); however this condition

needs to be relaxed for the det-unb case, resulting in Algorithm 4. Step 2.1 is also modified since we also need to relax the controllability condition. Specifically, Step 2.1 relaxes the controllability condition because the attacker "races" with the system at states that violate this condition.[3] Algorithm 4 flags states that violate the controllability condition to later analyze if they need to be pruned or not. Note that once a state is flagged, it remains flagged throughout the algorithm. Step 2.2 is divided into three steps. First, deadlocks created by the pruning process are deleted. Second, we flag all states violating the race-free condition. Then, the transition function is updated based on the flagged states. This update is such that only insertions transitions are possible from flagged states, since the attack will not wait for a reaction of the system. We can adapt Definition

Algorithm 4 Det-Unb Modification

1: **Step 2.1** Flag all $x \in X_{H_i}$ s.t. $En_B(a) \cap E_{uctr} \not\subseteq En_{H_i}(x)$
2: **Step 2.2**
3: **Step 2.2.1** $X^*_{H_i} = \{x \in X_{H_i} |\ En_{H_i}(x) = \emptyset \Rightarrow En_B(x) = \emptyset\}$
4: **Step 2.2.2** Flag all $x \in X^*_{H_i}$ s.t. $\big(e \in \Sigma_{obs} \wedge e \in En_B(x)\big) \Rightarrow \big(\{e, del(e)\} \cap En_{H_i}(x) = \emptyset\big)$
5: **Step 2.2.3** For $x \in X^*_{H_i}$ and $e \in E$

$$\delta'_{H_i}(x,e) = \begin{cases} \delta_{H_i}(x,e) & \text{if } e \in (\Sigma_{ins} \cup \Gamma) \wedge \delta_{H_i}(x,e)! \\ \delta_{H_i}(x,e) & \text{if } e \in (\Sigma_{del} \cup \Sigma_{obs}) \wedge \delta_{H_i}(x,e)! \wedge x \text{ not flagged} \\ \text{undefined} & \text{otherwise} \end{cases}$$

4.7 to prune the AIDA for the case of det-unb attacks by considering the modification of Step 2.1 and Step 2.2, as presented in Algorithm 4. We name the resulting stealthy IDA as the USDA, for *Unbounded Stealthy Deceptive Attack* structure. Versions of Lemmas (**4.2**-**4.3**), and Theorem **4.3** are created for this specific attacker.

Lemma 4.4. *A deterministic unbounded attack function f_A is embedded in the USDA if it satisfies conditions (1) and (2) from Problem **4.1**.*

Lemma 4.5. *If a det-unb attack function f_A is synthesized from USDA, then the stealthy condition of Problem **4.1** is satisfied.*

Theorem 4.4. *The USDA embeds all possible det-unb stealthy insertion-deletion attack strategies with respect to Σ_{sen}, $\tilde{R}$ and G.*

[3] It is interesting to mention the similarity of "racing" in our work with the work on supervisory control with forced events [47].

4.5.3 Synthesis of bounded deterministic attack strategies

The AIDA structure is general enough for the interruptible and the unbounded attack scenarios; however, as constructed, it does not capture the bound in the case of deterministic bounded attacks (or det-bounded case). We now present a simple mechanism in order to compute a bounded version of the AIDA, that we term BAIDA. (The $||$ operation is the standard parallel composition of automata.)

Definition 4.8. *Given the AIDA constructed by Algorithm 1 and the automaton shown in Figure 4.9, the BAIDA is defined as* $BAIDA = AIDA||G_{bound}$. *(For the purpose of* $||$*, the AIDA is treated as an automaton.)*

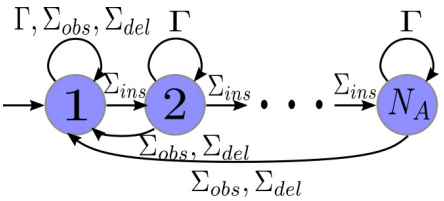


Figure 4.9: G_{bound}

The det-bounded attack case has similar conditions as the previously-discussed det-unb case, however, it can only perform bounded modifications. We need to take into account how many modifications the attacker has already performed at a given E-state. Therefore, synthesis algorithms for det-bounded attacks must use the BAIDA, as in Definition **4.8**.

The BAIDA, as the AIDA previously, has to be pruned. We only show the modification of Step 2 for Algorithm 2, resulting in Algorithm 5. Bounded attacks include features from both unbounded and interruptible attacks. E-states that have reached the maximum allowed modification behave as E-states in the interruptible case, in other words, they must satisfy the race-free condition. On the contrary, E-states that have not reached the maximum allowed editions are similar to E-states in the unbounded scenario. Consequently, Step 2.1 and Step 2.2 are a combination of the corresponding steps in the previous cases. A similar structure as the ISDA and the USDA can be introduced, however we omit such definition given that it would follow the same steps as in

Algorithm 5 Det-Bounded Modification

1: **Step 2.1**
2: **Step 2.1.1** Flag all $(x,n) \in X_{H_i}$ s.t. $En_B(x) \cap E_{uctr} \not\subseteq En_{H_i}((x,n)) \wedge n < N_A$
3: **Step 2.1.2** $X'_{H_i} = \{(x,n) \in X_{H_i} |\, n = N_A \Rightarrow En_B(x) \cap E_{uctr} \subseteq En_{H_i}((x,n))\}$
4: **Step 2.2**
5: **Step 2.2.1** $X''_{H_i} = \{(x,n) \in X'_{H_i} |\, En_{H_i}\big((x,n)\big) = \emptyset \Rightarrow En_B(x) = \emptyset\}$
6: **Step 2.2.2** $X^*_{H_i} = \{(x,n) \in X''_{H_i} |\, n = N_a \wedge e \in \Sigma_{sen} \wedge e \in En_B(x) \Rightarrow \big(e \in En_{H_i}((x,n)) \vee del(e) \in En_{H_i}((x,n))\big)\}$
7: **Step 2.2.3** Flag all $(x,n) \in X^*_{H_i}$ s.t. $n < N_a$, $(e \in \Sigma_{obs} \wedge e \in En_B(x)) \Rightarrow \big(\{e, del(e)\} \cap En_{H_i}((x,n)) = \emptyset\big)$
8: **Step 2.2.4** For $(x,n) \in X^*_{H_i}$ and $e \in E$

$$\delta'_{H_i}((x,n),e) = \begin{cases} \delta_{H_i}((x,n),e) & \text{if } e \in (\Sigma_{ins} \cup \Gamma) \wedge \delta_{H_i}((x,n),e)! \\ \delta_{H_i}((x,n),e) & \text{if } e \in (\Sigma_{del} \cup \Sigma_{obs}) \\ & \wedge \delta_{H_i}((x,n),e)! \wedge (x,n) \text{ is not flagged} \\ \text{undefined} & \text{otherwise} \end{cases}$$

the previous cases. The same comment holds for the lemmas and the theorems introduced in the previous cases. The details are left for the readers to work out.

4.6 Conclusion

We have considered the supervisory layer of feedback control systems, where sensor readings may be compromised by an attacker in the form of insertions and deletions. In this context, we have formulated the problem of synthesizing stealthy sensor deception attacks, that can cause damage to the system without detection by an existing supervisor.

Our solution procedure is game-based and relies on the construction of a discrete structure called the AIDA, which is used to solve the synthesis problem for three different attack scenarios. The AIDA captures the game between the environment (i.e., system and attacker) and the given supervisor. It embeds all valid actions of the attacker. Based on the AIDA, we specified a pruning procedure for each attack type, thereby constructing stealthy structures denoted as the ISDA and the USDA. Based on each type of stealthy structure, we can synthesize, if it exists, an attack function that leads the system to unsafe critical states without detection, for the corresponding attack scenario.

CHAPTER V

Synthesis of optimal sensor deception attacks for stochastic supervisory control

5.1 Introduction

In the previous chapter, we have considered synthesis of sensor deception attacks for systems modeled by logical models. Therefore, the attack strategies are analyzed qualitatively, meaning an attack strategy either succeeds or fails.

In this chapter, we study the same synthesis problem but for *stochastic* systems. Therefore, we augment the system with a quantitative measure that allows a quantitative analysis of the attack strategies. This gives rise to a broader class of attack strategies, as compared with our previous results.

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 an unsafe state of the uncontrolled plant. In this manner, a quantitative measure is introduced in the synthesis problem of attack strategies. First, we investigate the synthesis of an *attack function* that generates the maximum likelihood of reaching an unsafe state. This problem is denoted as the *probabilistic reachability attack function problem*. In the probabilistic reachability attack function problem, only a set of compromised sensor readings constrains the attacker on how to alter the communication channel between the system sensor and the supervisor. For this reason, we investigate a second problem where the attacker is penalized for each sensor modification. The second problem investigated is the *synthesis of attack functions that satisfy multiple objectives* (multi-objective). Namely, the attack function must reach an unsafe

state with maximum probability while minimizing a cost function based on the attacker sensor modifications.

Our solution methodology employs results from the area of stochastic controlled systems, more specifically Markov Decision Processes (MDPs). First, we show how to build the "right" MDP that captures the interaction of the attacker and the control system. Next, we show that the solution of the probabilistic reachability attack function problem is reducible to the probabilistic reachability problem in MDPs [48, 49]. Based on the solution of the first problem, we trim the previously constructed MDP to obtain a solution space for the multi-objective attack function problem. Lastly, we show that the solution of the multi-objective attack function problem is reducible to the stochastic shortest path problem in MDPs [48, 49].

Related work

Several works have addressed in recent years problems of cyber-security in the field of DES. We give a very brief overview of the existing work and compare it to our work in this chapter. We use herein a similar framework for how attacks take place on the communications from the sensors to the supervisor as in Chapters III and IV. In Chapter III, an attacker has compromised a subset of the sensors and is able to delete sensor readings or insert fictitious ones in the communication channel. Then, in Chapter IV, we investigate the problem of synthesizing stealthy *sensor deception attacks* for a known *logical* control system [25, 26].

In [20, 11, 12, 19, 22], the authors develop diagnostic tools to detect when controlled systems are being attacked. Their work is closely related to the work on fault diagnosis in DES, and it applies to both sensor and/or actuator attacks. Moreover, only [20, 19, 22] consider stochastic models, while the others consider logical DES models. Our problem differs from the problem considered in these works since we aim to compute an attack function that successfully reaches the critical state. Nonetheless, these diagnostic tools could be incorporated into our framework in order to add additional constraints to the attack function synthesis problems, i.e., synthesis of

stealthy attack functions.

5.2 Stochastic supervisory control theory

5.2.1 Probabilistic finite-state automaton

We consider a stochastic DES modeled as a Probabilistic Finite-State Automaton (PFA). A PFA is denoted by $H := (X_H, \Sigma, Pr_H, x_{0,H})$, where X_H, Σ, $x_{0,H}$ are defined as in a DFA, $Pr_H : X_H \times \Sigma \times X_H \to [0,1]$ is the probabilistic transition function (PTF). The PTF $Pr_H(x,e,y)$ specifies the probability of moving from state $x \in X_H$ to state $y \in X_H$ with event $e \in \Sigma$. We only consider PFAs that satisfy for any $x \in X_H$: $\sum_{e\in\Sigma}\sum_{y\in X_H} Pr_H(x,e,y) \in \{0,1\}$ but our methodology can be easily extended for the general case [50].

The function δ_H is defined to bridge the gap between a PFA and a DFA, where $\delta_H(x,e) = y$ if $Pr_H(x,e,y) > 0$. In this work, we assume that δ_H is deterministic, i.e., there does not exist $y, y^* \in X_H$, $y^* \neq y$, such that $Pr_H(x,\sigma,y) > 0$ and $Pr_H(x,\sigma,y^*) > 0$. Using this definition, every PFA H is associated with a corresponding DFA G where $\delta_G(x,e) = \delta_H(x,e)$ and $\mathcal{L}(H) := \mathcal{L}(G)$.

Finally, the notion of probabilistic languages (p-languages) of a PFA was introduced in [51].

Definition 5.1. *Formally, $L_p(H) : \Sigma^* \to [0,1]$ is defined recursively for $s \in \Sigma^*$ and $e \in \Sigma$:*

$$L_p(H)(\epsilon) := 1 \tag{5.1}$$

$$L_p(H)(se) := \begin{cases} L_p(H)(s)Pr_H(x,e,y) & \text{if } x = \delta_H(x_{0,H}, s) \text{ and } y = \delta_H(x_{0,H}, se) \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

In fact, $L_p(H)$ defines a probability measure on the σ-algebra $\mathcal{F}$ as defined in [51]. Given $s \in \mathcal{L}(H)$, its measurable set in $\mathcal{F}$ is defined as the set of all strings having s as a prefix. Two strings $s, s' \in \mathcal{L}(H)$ define disjoint measurable sets in $\mathcal{F}$ if and only if one is not a prefix of the other.

Following similar steps as in [51], we also define the expectation of a function $f : \Sigma^* \to$

$[0, \infty)$. Let $L \subseteq \mathcal{L}(H)$ be such that all strings in L define disjoint measurable sets in $\mathcal{F}$ and $\sum_{s \in L} L_p(H)(s) = 1$[1]. In this manner, we can define a probability mass function on L based on $L_p(H)$. The expectation of f is defined with respect to this probability mass function as:

$$\mathbb{E}^{L,H}[f(s)] := \sum_{s \in L} f(s) L_p(H)(s) \tag{5.3}$$

To simplify the notation in this section, we denote by e_s^i the i^{th} event of $s \in \Sigma^*$ such that $s := e_s^1 e_s^2 \dots e_s^{|s|}$. Moreover, we denote by $\bar{s}$ the set of all prefixes of s.

5.2.2 Stochastic supervisory control

In stochastic supervisory control theory, the system H is considered as the plant but there are different ways of studying its closed-loop behavior [52, 50, 53]. In our work, we use the results of supervisory control of stochastic DES introduced by [50], where only the plant behaves stochastically. Namely, both the specification and the supervisor are deterministic and defined as in the previously-described supervisory control framework. However, the supervisor *alters* the probabilistic behavior of the plant via the control actions it takes (disabling events). Conditions for the existence of a supervisor for the above control problem are provided in [50].

Formalizing the previous discussion, the disablement of events by R increases the probability of the enabled ones. In other words, R/H generates another p-language, in general, different than the p-language of H. Given a state $x \in X_H$, a state $y \in X_R$, and an event $e \in En_H(x) \cap En_R(y)$, the probability of e being executed is given by the standard normalization:

$$Pr_e^{x,y} := \frac{Pr_H(x, e, \delta_H(x, e))}{\sum_{e' \in En_H(x) \cap En_R(y)} Pr_H(x, e', \delta_H(x, e'))} \tag{5.4}$$

Similarly to the logical case, the closed-loop behavior can be described as the parallel composition $R||H$. However, the probabilistic information is lost in $R||H$. For this reason, we define

[1] This language matches the definition of terminating p-languages in [51].

a probabilistic parallel composition $||_p$ based on Eq. 5.4 and the standard parallel composition $||$ (Def. **2.3**). This composition performs the standard $||$ and updates the PTF for the composed system. Formally, $R||_pH := Ac((X_{R||_pH}, \Sigma, Pr_{R||_pH}, x_{0,R||_pH}))$ is defined by $X_{R||_pH} \subseteq X_H \times X_R$, $x_{0,R||_pH} := (x_{0,H}, x_{0,R})$, and for $x = (x_1, x_2),\ y = (y_1, y_2) \in X_H \times X_R$ and $e \in \Sigma$ the transition probability is:

$$Pr_{R||_pH}(x, e, y) := \begin{cases} Pr^e_{x_1,x_2} & \text{if } \delta_H(x_1, e) = y_1 \wedge \\ & \delta_R(x_2, e) = y_2 \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

In this manner, the closed-loop system R/H is described by $R||_pH$.

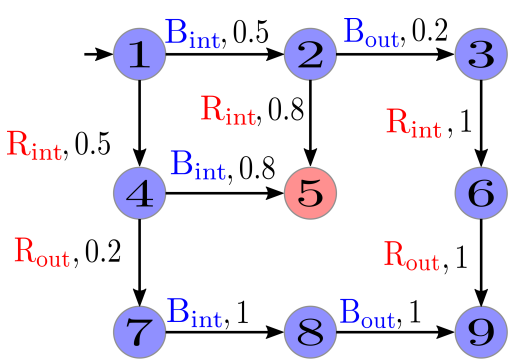
For simplicity and without loss of generality, we assume that the plant H has one deadlock critical state, denoted by $x_{crit} \in X_H$ and supervisor R ensures that this state is not reachable in R/H. Namely, we assume that $\forall s \in \mathcal{L}(H) \setminus \mathcal{L}(R/H)$, then $\delta_H(x_{0,H}, s) = x_{crit}$. We define the set of unsafe strings as $U_{uns} := \{s \in \Sigma^* \mid \delta_H(x_{0,H}, s) = x_{crit}\}$ and the set of unsafe state pairs for the controlled system R/H by

$$X_{uns} := \{(x_1, x_2) \in X_H \times X_R \mid \nexists s \in \mathcal{L}(R/H).\ x_1 = \delta_H(x_{0,H}, s) \wedge x_2 = \delta_R(x_{0,R}, s)\} \tag{5.6}$$

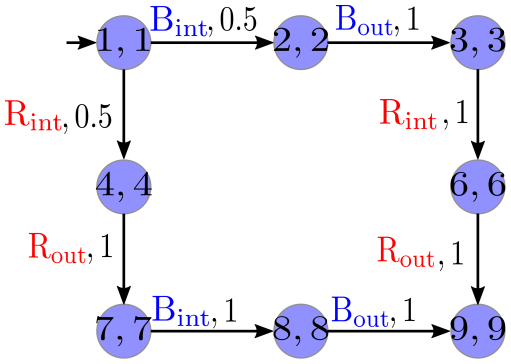
Example 5.1. *We return to the intersection example where we assume to have probabilistic knowledge about the model. Namely, we assume that the intersection is modeled by the PFA depicted in Fig. 5.1(a) with $\Sigma_{ctr} = \{B_{int}, R_{int}\}$. As in Example **2.4**, state 5 is the critical state*[2]*.*

The same supervisor shown in Fig. 2.6(a) guarantees that the controlled system does not reach state 5. Figure 5.1(b) depicts the closed-loop system R_1/H. Based on R_1 and H, we can compute the set X_{uns} as the pair of states that are not in $X_{R_1||H}$.

[2]Here, state 5 is transformed to be a deadlock state such that H matches our assumption of a single deadlock state.



(a) PFA H for the intersection example. Transitions are labelled as *event, probability of transition*.



(b) $H||_p R_1$

Figure 5.1: Stochastic intersection example

5.3 Problem formulation

In this section, we pose the two problems that are solved in this chapter: the probabilistic reachability attack function problem and the multi-objective optimal attack function.

5.3.1 Stochastic supervisory control under sensor deception attacks

We start by recalling some of the definitions in Chapter III as well as specializing them for this specific chapter.

In this chapter, we assume that the attacker is modeled by a deterministic unbounded attack function (Def. **3.7**). We restate the attack function definition specifically for this scenario. Recall that we also assume that every event is observable, i.e., $\Sigma_{obs} = \Sigma$.

Definition 5.2. *An attacker that hijacks the events in $\Sigma_{sen} \subseteq \Sigma$ in the communication channel between the plant and the supervisor is defined as a map $A : \Sigma^*_{all} \times \Sigma \cup \{\epsilon\} \to \Sigma^*_{all}$, that satisfies for any $t \in \Sigma^*_{all}$ and $e \in \Sigma \cup \{\epsilon\}$:*

1. *$A(\epsilon, \epsilon) \in {\Sigma_{ins}}^*$ and $A(t, \epsilon)$ is undefined for $t \neq \epsilon$*
2. *If $e \in \Sigma_{sen}$, then $A(t, e) \in \{e, del(e)\}{\Sigma_{ins}}^*$*
3. *If $e \in \Sigma \backslash \Sigma_{sen}$, then $A(t, e) \in \{e\}{\Sigma_{ins}}^*$*

The attack function A defines a deterministic strategy for event e based on the previous of modification t. Namely, $A(t, e)$ is a substitution rule where e is substituted by $A(t, e)$. For convenience, we define the function $\hat{A}$ that recursively concatenates these modifications for any string $s \in \Sigma^*$. Let $s \in \Sigma^*$, then we define $\hat{A}(s) = \hat{A}(s^{|s|-1})A(\hat{A}(s^{|s|-1}), e_s^{|s|})$ and $\hat{A}(\epsilon) = A(\epsilon, \epsilon)$. Note that the inverse function of $\hat{A}(s)$ is the projection P^H, i.e., $s = P^H(\hat{A}(s))$, where $P^H := P^G$.

A new controlled behavior is generated when the attack function A is placed in the communication channel between the plant and the supervisor. Namely, a new supervisor denoted by S_A is defined, where $S_A(s) = S \circ P^S \circ \hat{A}(s)$ is the resulting control action, under attack, after string s has been executed by the system. The language $\mathcal{L}(S_A/H) \subseteq \mathcal{L}(H)$ is defined as usual (see Chapter II) and S_A/H is denoted as the *attacked system*. This language is defined over Σ, and not Σ_{all}, due to the projection P^S of the attacker editions. Note that, S_A/H generates a p-language in the same manner as S/H.

Remark 5.1. *In the definition of the language of S_A/H, the attacker completes its string modification without any interruption of the plant H (Def. **3.7**). In other words, the plant H does not execute any event in the middle of the attacker editions following each event executed by the plant.*

We introduce the notion of *complete and consistent* attack functions. 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. Moreover, an attack function is consistent if its insertions are *consistent* with the current supervisor's control decision. This means that the attacker does not insert an event that is not allowed by the current supervisor's control decision.

Definition 5.3. *An attack function A is* complete *with respect to H and S if for any s in $\mathcal{L}(S_A/H)$, we have that $\hat{A}(s)$ is defined. Moreover, 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 $i \in |t| - 1$. We define Π_A as the set of all complete and consistent attack functions w.r.t. H and S.*

Remark 5.2. *Although we did not consider a detection module in this framework, a detection module can be incorporated into the supervisor if we slightly modify supervisor R. Namely, we*

*can introduce a supervisor that embeds an intrusion detection module (Def. **4.2**).*

5.3.2 The maximal reachability problem

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 $U_{uns} \cap \mathcal{L}(S_A/H) \neq \emptyset$. Moreover, an attack function A is quantified by the probability of generating 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,

$$win_A := \sum_{s \in U_{uns}} L_p(S_A/H)(s) \tag{5.7}$$

It follows from U_{uns} that $win_A \leq 1$ for any $A \in \Pi_A$. Namely, win_A captures the *probability* that S_A/H generates unsafe strings.

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 equally or 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 5.1. *[The probabilistic reachability attack function problem] Given a plant modeled as PFA H, a supervisor modeled as DFA R, and the set of compromised events $\Sigma_{sen} \subseteq \Sigma$, synthesize $A^{max} \in \Pi_A$ such that:*

$$win_{A^{max}} := \sup_{A \in \Pi_A} win_A \tag{5.8}$$

5.3.3 The multi-objective problem

In Problem **5.1**, the attacker is "eager" to reach the critical state and it is not constrained, except by Σ_{sen}, in how to do so. Finding a cost-optimal attack function would be a natural extension

for the investigated problem [54, 55]. Nonetheless, the reachable-optimal and the cost-optimal problems are conflicting. In this manner, a multiple-objective problem is investigated, i.e., maximal reachability and minimal expected cumulative cost.

We follow a similar approach as in [56, 57] where objectives have priorities. Namely, the attacker has multiple objectives to satisfy: (i) maximize the probability of reaching the critical state; (ii) minimize the expected cost while performing (i). First, the attacker prioritizes the probability of reaching the critical state and finds all attack functions that maximize this probability. Second, within these maximal attack functions, it searches for an attack function with the minimum expected cost.

In order to formally pose this problem, we need to introduce a cost function over the strings in $\mathcal{L}(S_A/H)$ for a given $A \in \Pi_A$. Recall that in Eq. (5.7), the winning level only considers unsafe strings. In addition to the set of unsafe strings, we consider the set of strings in $\mathcal{L}(S_A/H)$ from where the critical state is unreachable. Intuitively, cost is incurred for strings that reach the critical state and for strings from where the critical state is unreachable. These two sets are "stopping points" for the attacker, i.e., we assume that the attacker stops its attack once it reaches the critical state or once the critical state is unreachable. Note that, the union of these two sets is related to language L in the definition of Eq. (5.3). Formally, let $L_r = \overline{\mathcal{L}(S_A/H) \cap U_{uns}}$ be all prefixes from where the system can still reach the critical state. Then, the set of strings from where the critical state is unreachable is defined as:

$$U_{unr,A} := \{s \in \mathcal{L}(S_A/H) \mid s \in \mathcal{L}(S_A/H) \setminus L_r \wedge s^{|s|-1} \in L_r\} \tag{5.9}$$

It follows that $\sum_{s \in (U_{uns} \cup U_{unr,A})} L_p(S_A/H)(s) = 1$ for any $A \in \Pi_A$. Lastly, we assume a weight function $w : \Sigma_{all} \to [0, \infty)$ that captures the weight of each event in this set. Since we are interested in penalizing the attacker for its editions, w is defined to be zero for events in Σ and non-negative for events in Σ_{att}. Based on U_{uns} and $U_{unr,A}$, and w let the cumulative cost function $cost : \Sigma^* \to [0, \infty)$ be defined as follows:

$$cost(s) := \begin{cases} \sum_{i=1}^{|\hat{A}(s)|} w(e^i_{\hat{A}(s)}) & \text{if } s \in U_{uns} \cup U_{unr,A} \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

Remark 5.3. *The function $cost$ is defined for each string $s \in \mathcal{L}(S_A/H) \subseteq \Sigma^*$, but its value is calculated based on $\hat{A}(s) \in \Sigma^*_{all}$. The attacker modifications are only identified in strings in Σ^*_{all}. Since we penalize the attacker modifications of s, the value of the function $cost$ must be calculated based on $\hat{A}(s) \in \Sigma^*_{all}$. The function $cost$ is well define since the attack function is an* injective *function.*

Posing a minimization problem only based on this cumulative cost function is an ill-posed problem since an attacker that does not attack is a solution. Thus, this cumulative cost function only works since the minimization problem is preceded by a maximization problem.

Problem 5.2. *[Multi-Objective Optimal Attack Function Synthesis Problem] Given a plant H, a supervisor R, the set of compromised events $\Sigma_{sen} \subseteq \Sigma$, a solution of Problem **5.1** A^{max}, and the cumulative cost function $cost$, find*

$$A^{multi} \in \arg \inf_{A \in \Pi_A \text{ s.t. } win_A = win_{A^{max}}} \mathbb{E}^A[cost(s)] \tag{5.11}$$

where $\mathbb{E}^A$ is an abbreviation for $\mathbb{E}^{(U_{uns} \cup U_{unr,A}), S_A/H}$, the expected value defined with respect to the p-language generated by S_A/H and the set $U_{uns} \cup U_{unr,A}$ as defined in Eq. (5.3).

We search for an attack function in Π_A that satisfies Problem **5.1** and produces the minimal expected cumulative cost. This problem is well-posed since we search for an attack function that generates the minimum expected cost within the policies that reach the critical state with the maximum winning level. Therefore, the two problems we pose (Problems **5.1** and **5.2**) can be solved in sequence.

5.4 Markov decision processes

We briefly review concepts and notations of Markov Decision Processes (MDPs). An MDP is a tuple $M = (Q, q_{0,M}, Act, \delta_M)$, where Q is a finite set of states, $q_{0,M} \in Q$ is an initial state, Act is a finite set of actions, and $\delta_M : Q \times Act \times Q \to [0, 1]$ is a PTF (Probability Transition Function), where $\sum_{q' \in Q} \delta_M(q, a, q') \in \{0, 1\}$ for all $q \in Q$ and $a \in Act$.

An infinite run is a sequence $\rho = q_\rho^0 a_\rho^0 q_\rho^1 a_\rho^1 \ldots$, where $\delta_M(q_\rho^i, a_\rho^i, q_\rho^{i+1}) > 0$ for all $i \in \mathbb{N}$. Given ρ, we denote by q_ρ^i and a_ρ^i to be the i^{th} state and action in the run, respectively[3]. The set of all runs in M initialized at state q is defined as $Runs_q(M)$, and $Runs(M) = Runs_{q_{0,M}}(M)$. Similarly, $Pref_q(M)$ ($Pref(M)$) is the set of all finite-length prefixes of runs in $Runs_q(M)$ ($Runs(M)$). For a finite run $\rho \in Pref_q(M)$, the length of ρ is denoted by $|\rho|$, i.e., $\rho = q_\rho^0 a_\rho^0 \ldots a_\rho^{|\rho|-1} q_\rho^{|\rho|}$.

A deterministic strategy $\pi : \cup_{q \in Q} Pref_q(M) \to Act$ is a function that maps finite prefixes of runs into feasible actions. The set of all (deterministic) policies of M is defined as Π_M. Under a fixed strategy $\pi \in \Pi_M$, the behavior of M is fully probabilistic and it can be represented by an induced discrete-time Markov chain [55, 48]. This fixed strategy leads us to the standard definition of a probability measure $Pr_{M,q}^\pi$ over $Runs_q(M)$ [55]. We use the notation $Runs_q^\pi(M), Pref_q^\pi(M)$ to denote the set of runs generated by the MDP instantiated by $\pi \in \Pi_M$. In this probability measure $Pr_{M,q}^\pi$, the probability of generating a finite run $\rho \in Pref_q^\pi(M)$ is defined as

$$Pr_{M,q}^\pi(< \rho >) := \prod_{i \in [|\rho|-1]} \delta_M(q^i, a^i, q^{i+1}) \tag{5.12}$$

where $< \rho >:= \{\rho^* \in Runs_q^\pi(M) \mid q_\rho^i = q_{\rho^*}^i, i \in [|\rho|]\}$ [55].

We are interested in two MDP problems, the *probabilistic reachability problem* [54, 49] and the *stochastic shortest path problem* [48, 49]. The probability of reaching an absorbing set $Q_{abs} \subseteq Q$ from state $q \in Q$ is defined as:

$$p_{M,q}^\pi(Q_{abs}) := Pr_{M,q}^\pi(\{\rho \in Runs_q^\pi(M) | \exists j \in \mathbb{N},\ q_\rho^j \in Q_{abs}\}) \tag{5.13}$$

[3] If ρ is clear, the subscripts are omitted.

Similarly to the definition of runs, we define $p^{\pi}_{M}(Q_{abs}) = p^{\pi}_{M,q_{0,M}}(Q_{abs})$. Policies $\pi \in \Pi_M$ that reach Q_{abs} with probability 1 are denoted as proper policies [48]. The *probabilistic reachability problem* is defined as follows:

Problem 5.3. *Given an MDP M and a target absorbing set $Q_{abs} \subseteq Q$, find*

$$p^{*}_{M}(Q_{abs}) := \sup_{\pi \in \Pi_M} p^{\pi}_{M}(Q_{abs}) \tag{5.14}$$

Let $\mathbb{E}^{\pi}_{M,q}(f)$ denote the expected value of a measurable function $f : Runs_q(M) \to [0, \infty]$ with respect to $Pr^{\pi}_{M,q}$, see, e.g., [54, 55] for details. Moreover, the state q is omitted when $q = q_{0,M}$. Then, the *stochastic shortest path problem* is defined as follows:

Problem 5.4. *[48] Given an MDP M, and a cost function $c : Q \times Act \to [0, \infty]$, find*

$$c^{*}_{M}(Q_{abs}) := \inf_{\pi \in \Pi_M} \mathbb{E}^{\pi}_{M}[cumul(Q_{abs}, \rho)] \tag{5.15}$$

where $\rho \in Runs(M)$ and

$$cumul(Q_{abs}, \rho) := \begin{cases} \sum_{i=0}^{j_\rho} c(q^i_\rho, a^i_\rho) & \text{if } \exists j \in \mathbb{N} \text{ s.t.} \\ & q^j_\rho \in Q_{abs} \\ \infty & \text{otherwise} \end{cases}$$

and $j_\rho = \min\{j : q^j_\rho \in Q_{abs}\}$.

Both problems can be solved using standard MDP algorithms, e.g., value iteration, policy iteration, or linear programming, see [55, 48, 49]. Problem **5.4** has a solution, $c^{*}_{M} < \infty$, if and only if $\sup_{\pi \in \Pi_M} p^{\pi}_{M}(Q_{abs}) = 1$. Moreover, both problems accept memoryless and deterministic strategies as solutions [55, 48].

5.5 Solution of the probabilistic reachability attack function problem

5.5.1 Construction of the MDP

In Problem **5.1**, we assume that H and R are given as a PFA and a DFA, respectively. For this reason, an MDP is constructed based on these two models and its state space is defined based on the states of H and R. Moreover, actions in the MDP represent the attacker actions on the attacked controlled system. Formally, the MDP M is defined as follows.

Definition 5.4. *Given plant H, supervisor R, and compromised event set Σ_{sen}, the MDP $M := (Q, Act, \delta_M, q_{0,M})$ is constructed in the following manner.*

- $Q \subseteq X_H \times X_R \times ((\Sigma \setminus \Sigma_{sen}) \cup \Sigma_{del} \cup \{\epsilon\})$. *For a state q, we denote by q_H, q_R and q_e the plant state, the supervisor state, and the executed event, i.e., $q = (q_H, q_R, q_e)$.*
- *The set $Act := \{\tau, nd\} \cup \Sigma_{att}$.*
- *The initial state $q_{0,M} := (x_{H,0}, x_{R,0}, \epsilon)$.*
- *The PTF δ_M is defined for $x, y \in Q$ and action τ:*

$$\delta_M(x, \tau, y) := \begin{cases} Pr_{P^H(y_e)}^{x_H, x_R} & \text{if } y_e \in (\Sigma \setminus \Sigma_{sen}) \cup \Sigma_{del}, y_H = \delta_H(x_H, P^H(y_e)), \\ & y_R = \delta_R(x_R, P^S(y_e)) \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

For $x, y \in Q$ and action nd:

$$\delta_M(x, nd, y) := \begin{cases} 1 & \text{if } x_e \in \Sigma_{del},\ y_H = x_H, y_R = \delta_R(x_R, P^H(x_e)),\ y_e = \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

While for any $x, y \in Q$ and action $a \in \Sigma_{att}$:

$$\delta_M(x,a,y) := \begin{cases} 1 & \textit{if } a \in \Sigma_{ins},\ (x_H, x_R) \notin X_{uns}, \\ & y = (x_H, \delta_R(x_R, P^S(a)), \epsilon) \\ 1 & \textit{if } a = x_e \in \Sigma_{del},\ (x_H, x_R) \notin X_{uns}, \\ & y = (x_H, x_R, \epsilon) \\ 0 & \textit{otherwise} \end{cases} \tag{5.18}$$

- *Post-Processing: for any $x \in Q$ such that $\sum_{y \in Q} \delta_M(x,a,y) = 0$ for all $a \in Act$, δ_M is augmented for x by defining:*

$$\delta_M(x, \tau, (x_H, x_R, \epsilon)) := 1$$

and

$$\delta_M((x_H, x_R, \epsilon), \tau, (x_H, x_R, \epsilon)) := 1$$

The encoding of the attacked system to an MDP is not unique. Namely, there are different ways that an MDP can be constructed such that it encodes the dynamics of all possible attacked systems. We fixed a specific encoding where the action set of M is mapped to attacker actions in the attacked system.

Action τ represents the attacker waiting for the controlled system to execute an event, i.e., H executes an event. If in state $x \in Q$ there exists an event $e \in En(x_H) \cap En(x_R)$, then $\delta_M(x, \tau, y) = Pr_e^{x_H, x_R}$. Finally, the last function for action τ is to maintain the MDP live; this function is defined in the post-processing.

The nd (not deleting) action represents the attacker action of not deleting a compromised event from the channel, Eq. (5.17). If $x_e \in \Sigma_{del}$ at state $x \in Q$, it means that event $P^H(x_e)$ was just executed. At this point the attacker has two choices: report $P^H(x_e)$ to the supervisor, or decide to not report this event (deletion). Equation (5.17) represents the former choice, i.e., the supervisor receives the execution of event $P^H(x_e)$ and updates accordingly $\delta_R(x_R, P^H(x_e))$.

Lastly, actions in Σ_{att} represent the attacker inserting events in the channel or deleting events from the channel. The attacker can insert event $e \in \Sigma_{sen}$ at state $x \in Q$ if $e \in En(x_R)$; and it can delete event $e \in \Sigma_{sen}$ at state $x \in Q$ if $x_e = del(e)$, i.e., the event e was just executed.

Example 5.2. *In order to illustrate the construction of M, recall the plant H and the supervisor R_1 shown in Example **5.1**. The compromised event set for this example is $\Sigma_{sen} = \{R_{int}\}$. Figure 5.2 represents M for the controlled system R_1/H. In Figure 5.2, the transition function is represented by the directed edges with a respective action and probability value.*

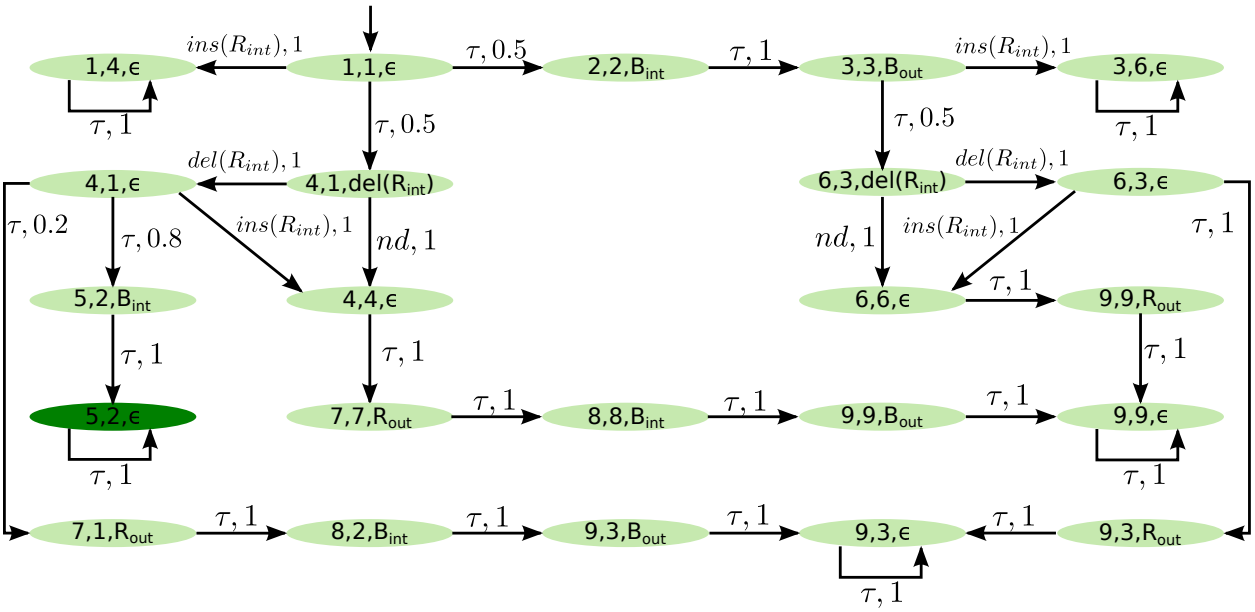


Figure 5.2: MDP for controlled system R_1/H and $\Sigma_{sen} = \{R_{int}\}$

To connect the MDP M to our original problem, we extract two strings from a given finite run in M. First, let $s_\rho = P^H(q^0_{\rho,e})P^H(q^1_{\rho,e}) \dots P^H(q^{|\rho|}_{\rho,e})$ for $\rho \in Pref(M)$. Second, for $\rho \in Pref(M)$ such that $q^{|\rho|}_{\rho,e} \notin \Sigma_{del}$, we define $t_\rho = e_0 e_1 \dots e_{|\rho|-1}$ where

$$e_i := \begin{cases} a^i_\rho & \text{if } a^i_\rho \in \Sigma_{att} \\ \mathcal{M}(q^{i+1}_{\rho,e}) & \text{if } (q^{i+1}_{\rho,e} \in \Sigma \setminus \Sigma_{sen}) \text{ or} \\ & (q^{i+1}_{\rho,e} \in \Sigma_{del} \text{ and } a^{i+1}_\rho = nd) \\ \epsilon & \text{otherwise} \end{cases} \tag{5.19}$$

Note that in the construction of t_ρ, to confirm that an event in H is executed between states i and $i+1$ we look at $q_{\rho,e}^{i+1}$. In Example **5.2**, given

$$\rho = (1,1,\epsilon)\tau(4,1,del(R_{int}))del(R_{int})(4,1,\epsilon)ins(R_{int})(4,4,\epsilon)\tau(7,7,R_{out})$$

then

$$s_\rho = P^H(\epsilon)P^H(del(R_{int}))P^H(\epsilon)P^H(\epsilon)P^H(R_{out}) = R_{int}R_{out}$$

$$t_\rho = \epsilon del(R_{int})ins(R_{int})R_{out} = del(R_{int})ins(R_{int})R_{out}$$

The string $t_\rho \in \Sigma_{all}^*$ represents the modified behavior while $s_\rho \in \mathcal{L}(H)$ represents the executed behavior. We also define the objective set Obj as the set of absorbing states where the attacker has reached its goal.

$$Obj := \{(q_H, q_R, \epsilon) \in Q \mid (q_H, q_R) \in X_{uns}\} \tag{5.20}$$

5.5.2 Maximal reachability attack function

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 MDP M; once a strategy π is fixed, we can calculate the probability of reaching the states in Obj. The construction of MDP M and the set Obj ties these two problems together. The following theorem connects the solution of Problem **5.3** for MDP M and set Obj with the solution of Problem **5.1**.

Theorem 5.1. *Consider the MDP M and the set Obj, then*

$$win_{A^{max}} = p_M^*(Obj) \tag{5.21}$$

Although Problem **5.3** and Problem **5.1** have similarities, to formally prove Theorem **5.1** a sequence of intermediate results is needed. These intermediate results are intuitively stated in the

following subsection with their formal proofs. Moreover, the solution procedure of Problem **5.3** not only outputs the maximum probability $p^*_M(Q_{abs})$ but also outputs $p^*_{M,q}(Q_{abs})$ for all $q \in Q$. In this manner, one can use standard methods to extract an optimal strategy π^* that achieves $p^{\pi^*}_M(Obj) = p^*_M(Obj)$. In Section 5.5.3, we show how to construct an attack function A^{max} based on π^* such that A^{max} is a solution of Problem **5.1**.

Example 5.3. *We return to our intersection example, where we solve Problem **5.3** for the MDP depicted in Fig. 5.2. The solution procedure of Problem **5.3** outputs $p^*_q(Obj)$ for each $q \in Q$; these values are shown in blue in Fig. 5.2. The probability of reaching the set $Obj = \{5, 2, \epsilon\}$ (in dark green) is $p^*_M(Obj) = 0.4$. Therefore, $win_{A^{max}} = 0.4$ and A^{max} is defined as: $A^{max}(\epsilon, R_{int}) = del(R_{int})$ and $A^{max}(s, e) = e$ for all $s \in \Sigma^*_{all}$ and $e \in \Sigma \cup \{\epsilon\}$. This is of course the expected solution for Problem **5.1** since deleting event R_{int} after its execution leads the supervisor to allow a move to the illegal state* 5.

5.5.3 Proof of theorem 5.1

In order to prove Theorem **5.1**, we present a sequence of equivalences between an attacked system, and the MDP M instantiated by a strategy $\pi \in \Pi_M$. As the MDP M is constructed, it allows strategies that are mapped to attack functions with infinite insertions. Since we only allow attack functions with finite insertion (arbitrarily long), we must address this problem first. We show that the set of strategies that are mapped to attack functions with finite insertions is sufficient to solve Problem **5.3**.

Next, we show that there is a one-to-one map between these strategies in M and complete and consistent attack functions. We also show that the probability space generated by the probabilistic language of an attacked system is equivalent to the probability space generated by M and a strategy π. Finally, the winning level of an attack function is equivalent to the probability of reaching the set Obj using a strategy π in M.

We start by stating a lemma about any strategy in the MDP M and runs that represent infinite insertion. This lemma states that the probability of reaching the objective set is zero for states with

infinite insertion future.

Lemma 5.1. *Let* $\pi \in \Pi_M$. *For all* $\rho \in Runs^{\pi}(M)$ *such that*

$$(\exists j \in \mathbb{N})(\forall k \geq j)[\pi(q_\rho^0 \dots q_\rho^k) \in \Sigma_{ins}]$$

then $p^{\pi}_{M,q_\rho^j}(Obj) = 0$.

Proof. Let $\rho \in Runs^{\pi}(M)$ such that $(\exists j \in \mathbb{N})(\forall k \geq j)[\pi(q_\rho^0 \dots q_\rho^k) \in \Sigma_{ins}]$. Then $p^{\pi}_{M,q_\rho^j}(Obj) < 1$ otherwise $\exists k \geq j$ such that $\pi(q_\rho^0 \dots q_\rho^k) = \tau$. It implies that $q_\rho^j \notin Obj$ and $q^j_{\rho,H} \neq x_{crit}$. Moreover, $\pi(q_\rho^0 \dots q_\rho^k) \in \Sigma_{ins}$ for all $k \geq j$, then by construction of M we have $\delta_M(q_\rho^k, \pi(q_\rho^0 \dots q_\rho^k), q_\rho^{k+1}) = 1$. Again by construction of M, $q^k_{\rho,H} \neq x_{crit}$ for all $k \geq j$. It follows that $p^{\pi}_{M,q_\rho^k}(Obj) = 0$. □

Let Π^f_M be the set of all strategies of M that *do not* generate infinite insertions, i.e., $\Pi^f_M = \{\pi \in \Pi_M \mid \rho \in Runs^{\pi}(M) \Rightarrow (\forall j \in \mathbb{N})(\exists k \geq j)[\pi(q_\rho^0 \dots q_\rho^k) \notin \Sigma_{ins}]\}$. Based on Lemma **5.1**, the following proposition states that Π^f_M is sufficient to solve Problem **5.3** for MDP M.

Proposition 5.1. $p^*_M(Obj) = \max_{\pi \in \Pi^f_M} p^{\pi}_M(Obj)$.

Proof. We show that if $\pi \in \Pi_M$ is optimal and it has infinite insertions, then we can construct a $\pi^f \in \Pi^f_M$ based on π that is also optimal. Assume $\pi \in \Pi_M \setminus \Pi^f_M$ and $p^{\pi}_M(Obj) = p^*_M(Obj)$. Let I be the set of runs $\rho \in Runs^{\pi}(M)$ that satisfies $(\exists j \in \mathbb{N})(\forall k \geq j)[\pi(q_\rho^0 \dots q_\rho^k) \in \Sigma_{ins}]$. By Lemma **5.1**, for any $\rho \in I$ then there exists $j \in \mathbb{N}$ such that $p^{\pi}_{M,q_\rho^j}(Obj) = 0$. This implies that we can modify π for prefixes of runs in I without altering the probabibility of reaching Obj. Without loss of generality, we can assume that π is a memoryless strategy [55, 48]. Let π^f be defined in the following manner for any $q \in Q$: $\pi^f(q) = \tau$, if $\pi(q) \in \Sigma_{ins}$ and $(\forall \rho \in Pref^{\pi}_q(M))[a^j_\rho \in \Sigma_{ins}]$; $\pi^f(q) = \pi(q)$, otherwise. It follows that $\pi^f \in \Pi^f_M$ and $p^{\pi^f}_M(Obj) = p^{\pi}_M(Obj) = p^*_M(Obj)$. □

Proposition **5.1** allows us to focus only on strategies in Π^f_M. The proof of Proposition **5.1** also provides the result that a memoryless strategy in Π^f_M achieves $p^*_M(Obj)$ since a memoryless strategy in Π_M achieves $p^*_M(Obj)$ [55, 48]. In the proof of Proposition **5.1**, a memoryless strategy

in Π_M remains a memoryless strategy in Π_M^f. As consequence, an optimal strategy can be found in the space of memoryless policies in Π_M^f. First, we state a proposition that ties the relation between the language $\mathcal{L}(S_A/H)$ for any complete and consistent attack function A with finite runs in M.

Proposition 5.2. *Given $A \in \Pi_A$, for any $s \in \mathcal{L}(S_A/H)$ and $t = \hat{A}(s)$, there exists a unique $\rho \in Pref(M)$ such that $s = s_\rho$, $t = t_\rho$ and $(a^{|\rho|-1} = \tau \wedge q_e^{|\rho|} = \epsilon) \Rightarrow q^{|\rho|} \neq q^{|\rho|-1}$.*

Proof. We use induction on the length of s to prove the result the existence of $\rho \in Pref(M)$.

Induction basis: $s = \epsilon \in \mathcal{L}(S_A/H)$ and $t = \hat{A}(\epsilon) = A(\epsilon, \epsilon)$.

By the definition of A, the string t is finite. Let $a^{i-1} = e_t^i$ for $i \in [|t|]^+$, by Definition **5.2** each $e_t^i \in \Sigma_{ins}$. Also, let $q^i = (x_{0,H}, \delta_R(x_{0,R}, t^i), \epsilon)$ for $i \in [|t|]$. Each $q^i \in Q$ since A is consistent, $\delta_R(x_{0,R}, t^i)!$, and by construction of M. Thus, $\delta_M(q^i, a^i, q^{i+1}) = 1$ by construction of M. Therefore, $\rho = q^0 a^0 \dots q^{|t|} \in Pref(M)$, $q_e^{|t|} \notin \Sigma_{del}$, $s_\rho = s$ and $t_\rho = t$.

Induction hypothesis: The proposition holds for all $s \in \mathcal{L}(S_A/H)$ with $|s| \leq n$.

Induction step: Let $|s| = n + 1$.

From the induction hypothesis, $\exists \rho \in Pref(M)$ such that the proposition holds for s^n. Recall that $\hat{A}(s) = \hat{A}(s^{|s|-1})A(s^{|s|-1}, e_s^{n+1})$. Let $t' = A(s^n, e_s^{n+1})$, then it is enough to show that $\exists \rho' \in Pref_{q_\rho^{|\rho|}}(M)$ such that $e_s^{n+1} = P^H(q_{\rho',e}^1) = P^H(q_{\rho',e}^1) \dots P^H(q_{\rho',e}^{|\rho'|})$ and $t' = t_{\rho'}$. Based on Definition **5.2**, $e_{t'}^1 \in \Sigma \cup \Sigma_{del}$ and $\mathcal{M}(e_{t'}^1) = e_s^{n+1}$.

Assume that $e_{t'}^1 \in \Sigma \setminus \Sigma_{sen}$ and let $a^0 = \tau$ and $q^0 = q_\rho^{|\rho|}$. Since $s \in \mathcal{L}(S_A/H)$ and by the induction hypothesis, $\delta_H(x_{0,H}, s) = \delta_H(\delta_H(x_{0,H}, s^n), e_s^{n+1}) = \delta_H(q_H^0, e_{t'}^1) = q_H$ and $\delta_R(x_{0,R}, \hat{A}(s^n)e_{t'}^1) = \delta_R(\delta_R(x_{0,R}, \hat{A}(s^n)), e_{t'}^1) = \delta_R(q_R^0, e_{t'}^1) = q_R$ are well defined. Thus, $\delta_M(q^0, \tau, q^1) > 0$ by construction of M, where $q^1 = (q_H, q_R, e_s^{n+1})$. Since $e_{t'}^{i+1} \in \Sigma_{ins}$ for $i \in [|t'| - 1]^+$ by Definition **5.2**, we can apply the same strategy as in the induction basis. Namely, $a^{i+1} = e_{t'}^{i+2}$ and $q^{i+2} = (q_H, \delta_R(q_R, t'^{i+2}), \epsilon) \in Q$ for $i \in [|t'| - 2]$. Again since A is consistent, it follows that $\delta_M(q^{i+1}, a^{i+1}, q^{i+2}) = 1$ for $i \in [|t'| - 2]$. In this manner $\rho' = q^0 a^0 \dots q^{|t|} \in Pref_{q_\rho^{|\rho|}}(M)$ and $\rho a^0 \dots q^{|t'|} \in Pref(M)$ such that $s_{\rho a^0 \dots q^{|t|}} = s$ and $t_{\rho a^0 \dots q^{|t|}} = \hat{A}(s^n)t' = \hat{A}(s)$. Similar arguments can be made when $e_{t'}^1 \in \Sigma_{del} \cup \Sigma_{sen}$. In this

case, $e^1_{t'}$ defines q^0, q^1, q^2, a^0 and a^1 while q^{i+3} and a^{i+2} are defined based on $e^{i+2}_{t'}$ for $i \in [|t'| - 2]$. This concludes the first assertion of the proof.

The condition $(a^{|\rho|-1} = \tau \wedge q_e^{|\rho|} = \epsilon) \Rightarrow q^{|\rho|} \neq q^{|\rho|-1}$ is satisfied by the construction of ρ in the induction proof. Finally, uniqueness follows from the fact that $\hat{A}$, H and R are deterministic. $\square$

In the post-processing of δ_M in Definition **5.4**, we have to extend finite strings in the attacked system to infinite runs in the MDP. If we disregard this "superfluous" behavior introduced in the post-processing, then Proposition **5.2** relates any pair of strings $(s, \hat{A}(s))$, given an attack function A, to a unique finite run in M.

Thus, the condition $(a^{|\rho|-1} = \tau \wedge q_e^{|\rho|} = \epsilon) \Rightarrow q^{|\rho|} \neq q^{|\rho|-1}$ eliminates the "superfluous" behavior introduced by the post-processing. Thus, there exists a one-to-one map between pairs of strings and finite runs. Consequently, attack functions can be related to strategies for the MDP M (Π^f_M) by a one-to-one map. Let us provide this one-to-one map by first constructing a relation between attack functions and MDP strategies.

Definition 5.5. *Given $A \in \Pi_A$, we define $\pi_A \in \Pi^f_M$ using the result of Prop. 5.2. Let $s \in \mathcal{L}(S_A/H)$ and $\hat{A}(s) \in \Sigma^*_{all}$, then define π_A for $\rho \in Pref(M)$ such that $s_\rho = s$, and $t_\rho = \hat{A}(s)$ as:*

$$\pi_A(q^0_\rho a^0_\rho q^1_\rho \dots q^j_\rho) := a^j_\rho,\ j \in [|\rho| - 1] \tag{5.22}$$

The strategy π_A for a given $A \in \Pi_A$ is well-defined even though it re-assigns strategies for a run $\rho \in Pref(M)$. Namely if two strings $s, s' \in \mathcal{L}(S_A/H)$ share a prefix, then the respective finite runs $\rho, \rho' \in Pref(M)$ also share a prefix and π_A is defined for each prefix of ρ, ρ'. Since A is deterministic these re-assignments of π_A are identical.

For example, we partially define an attack function for Example **5.1** as $A(\epsilon, \epsilon) = \epsilon$ and $A(\epsilon, R_{int}) = del(R_{int})$. For this case, $s = \epsilon$ has

$$(1, 1, \epsilon)$$

as its respective finite run in the MDP M shown in Figure 5.2. It follows from $s = \epsilon$ and $A(\epsilon, \epsilon) = \epsilon$ that $\pi_A((1, 1, \epsilon)) = \tau$. Similarly, $s = R_{int}$ has

$$(1, 1, \epsilon)\tau(4, 1, del(R_{int}))del(R_{int})(4, 1, \epsilon)$$

as its respective finite run. In this case, it follows from $s = R_{int}$ and $A(\epsilon, R_{int}) = del(R_{int})$ that

$$\pi_A((1, 1, \epsilon)) = \tau$$

and

$$\pi_A((1, 1, \epsilon)\tau(4, 1, del(R_{int}))) = del(R_{int})$$

Although $\pi_A((1, 1, \epsilon))$ is defined by both of these finite runs, each of them defines

$$\pi_A((1, 1, \epsilon)) = \tau$$

Definition 5.6. *Let $\pi \in \Pi_M^f$, then we define $A_\pi \in \Pi_A$ by defining $\hat{A}_\pi(s)$ for $s \in \Sigma^*$. Let $\rho \in Pref^\pi(M)$ such that $\pi(\rho) = \tau$, then we define $\hat{A}_\pi$ for $\rho' = q_\rho^0 a_\rho^0 \dots q_\rho^{|\rho|-1}$ as:*

$$\hat{A}_\pi(s_{\rho'}) := t_{\rho'} \tag{5.23}$$

Note that in Definition **5.6**, $\hat{A}_\pi$ is defined for strings constructed based on finite traces $\rho \in Pref^\pi(M)$ such that $\pi(\rho) = \tau$. This definition matches the definition of attack function where the end of an attacker modification is marked by an observable event executed by H. We give an example of Definition **5.6** based on Figure 5.2. Let $\pi((1, A, \epsilon)) = ins(R_{int})$ and $\pi(q) = \tau$ for $q \in Q \setminus \{(1, 1, \epsilon)\}$, then

$$Pref^\pi(M) = \{(1, 1, \epsilon), (1, 1, \epsilon)ins(R_{int})(1, 4, \epsilon)\tau, \dots\}$$

In this manner, selecting $\rho = (1,1,\epsilon)ins(R_{int})(1,4,\epsilon)\tau$ defines $\hat{A}_\pi(\epsilon) = ins(R_{int})$, which is a complete and consistent attack function. The attacked language is $\mathcal{L}(S_{A_\pi}/H) = \{\epsilon\}$ since the controlled system deadlocks after the insertion of R_{int} at the initial state of H and R_1.

Based on Definitions **5.5** and **5.6**, the following proposition relates the probability spaces of an attacked system and the MDP instantiated by a strategy $\pi \in \Pi_M^f$.

Proposition 5.3. *Given $A \in \Pi_A$, then for any $s \in \mathcal{L}(S_A/H)$:*

$$L_p(S_A/H)(s) = Pr_M^{\pi_A}(< \rho_s >) \tag{5.24}$$

where $\rho_s \in Pref^{\pi_A}(M)$ such that $s = s_{\rho_s}$, and $\hat{A}(s) = t_{\rho_s}$.

Conversely, given $\pi \in \Pi_M^f$, then for any $\rho \in Pref^{\pi}(M)$:

$$Pr_M^{\pi}(< \rho >) = L_p(S_{A_\pi}/H)(s_\rho) \tag{5.25}$$

Proof. We start with Eq. (5.24). The equality trivially holds when $s = \epsilon$. Let $s \in \mathcal{L}(S_A/H)$ such that $|s| > 0$, then

$$L_p(S_A/H)(s) = \prod_{i \in [|s|]^+} Pr_{e_s^i}^{\delta_H(x_{0,H}, s^{i-1}), \delta_R(x_{0,R}, \hat{A}(s^{i-1}))} \tag{5.26}$$

Using Def. **5.5**, let $\rho \in Pref^{\pi_A}(M)$ such that $s = s_\rho$ and $t_\rho = \hat{A}(s)$. Such ρ always exists since we can construct as in Def. **5.5**. Then for $i \in [|s|]$ there exists $j \geq i$ such that $\delta_H(x_{0,H}, s^i) = q_H^j$, $\delta_R(x_{0,R}, \hat{A}(s^i)) = q_R^j$, $s^i = s_{q^0 a^0 \ldots q^j}$, and $\hat{A}(s^i) = t_{q^0 a^0 \ldots q^j}$. It follows that:

$$L_p(S_A/H)(s) \overset{\text{Def. 5.4}}{=} \prod_{i \in [|\rho|-1] \text{ s.t. } q_e^{i+1} \neq \epsilon} Pr_{P^H(q_e^{i+1})}^{q_H^i, q_R^i} \tag{5.27}$$

$$\overset{\text{Def. 5.4}}{=} \prod_{i \in [|\rho|-1]} \delta_M(q^i, a^i, q^{i+1}) \tag{5.28}$$

$$\overset{\text{Eq. (5.12)}}{=} Pr_M^{\pi_A}(< \rho >) \tag{5.29}$$

This concludes the first part of the proof.

We now show Eq. (5.25). The equality trivially holds for $\rho \in Pref^{\pi}(M)$ such that $\pi(\rho) = \tau$ and $(\forall j \in [|\rho|])[q^j_{\rho,e} = \epsilon]$. Let $\rho \in Pref^{\pi}(M)$ such that $\pi(\rho) = \tau$ and $(\exists j \in [|\rho|])[q^j_{\rho,e} \neq \epsilon]$, then

$$Pr^{\pi}_{M}(< \rho >) \overset{\text{Eq. (5.12)}}{=} \prod_{i \in [|\rho|-1]} \delta_M(q^i, a^i, q^{i+1}) \tag{5.30}$$

$$\overset{\text{Def. 5.4}}{=} \prod_{i \in [|\rho|-1] \text{ s.t. } q_e^{i+1} \neq \epsilon} Pr^{q_H^i, q_R^i}_{P^H(q_e^{i+1})} \tag{5.31}$$

Using Def. **5.4** and Def. **5.6**, we have that for $i \leq |\rho|$ such that $q_e^i \neq \epsilon$, $q_H^{i-1} = \delta_H(x_{0,H}, s_{q^0 a^0 \ldots q^{i-1}})$ and $q_R^{i-1} = \delta_R(x_{0,R}, t_{q^0 a^0 \ldots q^{i-1}}) = \delta_R(x_{0,R}, \hat{A}_{\pi}(s_{q^0 a^0 \ldots q^{i-1}}))$ and $s_\rho \in \mathcal{L}(S_{A_\pi}/H)$. Hence

$$= \prod_{i \in [|\rho|-1] \text{ s.t. } q_e^{i+1} \neq \epsilon} Pr^{\delta_H(x_{0,H}, s_{q^0 a^0 \ldots q^i}), \delta_R(x_{0,R}, \hat{A}_{\pi}(s_{q^0 a^0 \ldots q^i}))}_{P^H(q_e^{i+1})} \tag{5.32}$$

$$= \prod_{k \in [|s_\rho|-1]} Pr^{\delta_H(x_{0,H}, s^k_\rho), \delta_R(x_{0,R}, \hat{A}(s^k_\rho))}_{e^{k+1}_{s_\rho}} = L_p(S_{A_\pi}/H)(s_\rho) \tag{5.33}$$

This concludes our proof. □

Finally, we state a proposition which is a direct consequence of Proposition **5.3**. This proposition relates the winning level with the probability of reaching the set Obj in the MDP.

Proposition 5.4. *Given a strategy $\pi \in \Pi^f_M$, then $p^{\pi}_M(Obj) = win_{A_\pi}$. Conversely, let $A \in \Pi_A$, then $win_A = p^{\pi_A}_M(Obj)$.*

Proof. First, we show that $p^{\pi}_M(Obj) = win_{A_\pi}$ for any strategy $\pi \in \Pi^f_M$. Define the set $Win^{\pi} = \{\rho \in Pref^{\pi}(M) \mid q^{|\rho|} \in Obj \wedge q^{|\rho|-1} \notin Obj\}$. Let $\mathbb{1}_A(x)$ for any given set A denote the indicator function, i.e., $\mathbb{1}_A(x) = 1$ if $x \in A$ and $\mathbb{1}_A(x) = 0$ otherwise. Then, the value $p^{\pi}_M(Obj)$ can be obtained by:

$$p^{\pi}_M(Obj) = \mathbb{E}^{\pi}_M[\mathbb{1}_{Win^{\pi}}(\rho)] \tag{5.34}$$

Recall that ρ is related to a unique pair of strings s_ρ and $\hat{A}_\pi(s_\rho)$, as shown by Proposition **5.2** and used in Definition **5.6**. The set Win^{π} excludes cases of infinite runs and Win^{π} relates to the set

$U_{uns,A_\pi} = U_{uns} \cap \mathcal{L}(S_{A_\pi}/H)$ by a one-to-one map. Since Win^π is countable, it follows:

$$p_M^\pi(Obj) \overset{\text{Eq.}(5.34)}{=} \sum_{\rho \in Win^\pi} Pr_M^\pi(< \rho >) \tag{5.35}$$

$$\overset{\text{Def.}\mathbf{5.6}}{=} \sum_{s \in U_{uns,A_\pi}} L_p(S_{A_\pi}/H)(s) \tag{5.36}$$

$$= \sum_{s \in U_{uns}} L_p(S_{A_\pi}/H)(s) \overset{\text{Eq.}(5.7)}{=} win_{A_\pi} \tag{5.37}$$

In the same manner, we can show that for a given attack function A and its derived strategy π_A, the equality $win_A = p_M^{\pi_A}(Obj)$ holds. The steps are the same as in the first part of the proof. □

Finally, the proof of Theorem **5.1** follows directly from Proposition **5.4**. Let $\pi^* \in \Pi_M^f$ be a strategy such that $p_M^*(Obj) = p_M^{\pi^*}(Obj)$, then $win_{A_{\pi^*}} = p_M^{\pi^*}(Obj)$. For any $A \in \Pi_A$, we have that $win_A = p_M^{\pi_A}(Obj) \leq p_M^*(Obj) = p_M^{\pi^*}(Obj) = win_{A_{\pi^*}}$. Consequently, $win_{A^{max}} = win_{A_{\pi^*}}$ and A_{π^*} is a solution of Problem **5.1**.

5.6 Solution of the multi-objective problem

5.6.1 Construction of the MDP

As was mentioned before, Problems **5.1** and **5.2** can be solved in sequence. In this manner, MDP M is trimmed such that only policies that are a solution of Problem **5.3** remain. Based on this trimmed MDP, we are going to relate Problem **5.4** to Problem **5.2** similarly as in Section 5.5.

In Problem **5.3**, we obtain the maximum probability of reaching an absorbing state set Q_{abs} from the initial state of an MDP. In fact, the solution procedure not only outputs the maximum probability $p_M^*(Q_{abs})$ but also outputs $p_{M,q}^*(Q_{abs})$ for all $q \in Q$.

Back to our specific MDP M as in Definition **5.4**, we obtain $p_{M,q}^*(Obj)$ for all $q \in Q$ by solving Problem **5.3** for the absorbing state set Obj. The trimmed MDP is defined based on actions that guarantee the optimal value $p_{M,q}^*(Obj)$ in state $q \in Q$. Namely, we use the principle of optimality to identify optimal actions at any given state $q \in Q$. We also create a fictitious state called q_{abs} that

aggregates states of Q such that $p^*_{M,q}(Obj) = 0$, i.e., states that cannot reach the set Obj. Formally, the trimmed MDP is defined as follows:

Definition 5.7. *Given the MDP M as in Definition **5.4**, the set Obj, and the values $p^*_{M,q}(Obj)$ obtained by solving Problem **5.3**, we construct the trimmed MDP $M_{tr} = (Q_{tr}, Act, \delta_{tr}, q_{0,tr})$ as follows:*

- *$Q_{tr} \subseteq Q$ is defined as $Q_{tr} = \cup_{i \geq 1} Q_i \cup \{q_{abs}\}$ for Q_i defined recursively as:*

$$\begin{aligned} Q_1 =& \{q_{0,M}\} \\ Q_{i+1} =& \{q \in Q \mid \delta_M(q', a^*, q) > 0 \text{ for } q' \in Q_i, \\ & a^* \in \arg\max_{a \in Act} N(q', a)\} \end{aligned}$$

  where $N(q', a) = \sum_{q \in Q} p^*_{M,q}(Obj)\delta_M(q', a, q)$.
  By the principle of optimality $\max_{a \in Act} N(q', a) = p^*_{M,q'}(Obj)$.

- *The set $Act = \{\tau, nd\} \cup \Sigma_{att}$.*

- *The initial state $q_{0,tr} = q_{0,M}$.*

- *The PTF δ_{tr} is defined for $x, y \in Q_{tr} \setminus \{q_{abs}\}$ and action $a \in Act$:*

$$\delta_{tr}(x, a, y) = \delta_M(x, a, y) \tag{5.38}$$

$$\delta_{tr}(x, \tau, q_{abs}) = 1 - \sum_{q \in Q_{tr} \setminus \{q_{abs}\}} \delta_{tr}(x, \tau, q) \tag{5.39}$$

$$\delta_{tr}(q_{abs}, \tau, q_{abs}) = 1 \tag{5.40}$$

Remark 5.4. *Equation (5.39) is only defined for τ actions since all other actions occur with probability 1 as defined in Equations (5.17-5.18).*

Example 5.4. *Let M be the MDP depicted in Figure 5.2, where $p^*_{M,q}$ for $q \in Q$ is given as in Example **5.3**. In this manner, we construct M_{tr} as specified in Def. **5.7**. The trimmed MDP is depicted*

in Figure 5.3. States $(1,4,\epsilon),(3,6,\epsilon)$, $(4,4,\epsilon)$, $(6,3,\epsilon)$, $(7,7,R_{out})$, $(8,8,B_{int})$,$(9,3,R_{out})$, *and* $(9,9,B_{out})$ *are not part of* Q_{tr} *since they are reached by nonoptimal actions. On the other hand, states* $(2,2,B_{int})$, $(3,3,B_{out})$, $(6,3,del(R_{int}))$, $(6,6,\epsilon)$, $(7,1,R_{out})$, $(8,2,B_{int})$, $(9,3,B_{out})$, $(9,3,\epsilon)$, $(9,9,R_{out})$, *and* $(9,9,\epsilon)$ *are replaced by state* q_{abs} *since they can be reached by an optimal action but both cannot reach the set* Obj.

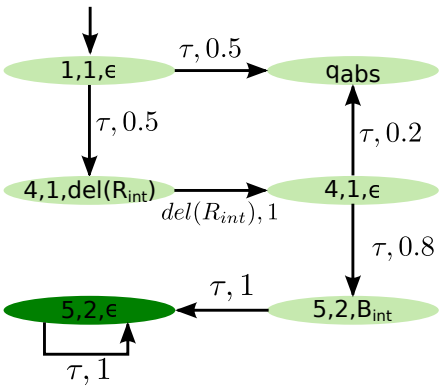


Figure 5.3: Trimmed MDP M_{tr}

In order to have a complete instance of Problem **5.4** for M_{tr}, a cost function must be defined. Let c be defined as follows:

$$c(q,a) = \begin{cases} w(a) & \text{if } a \in \Sigma_{att} \\ 0 & \text{otherwise} \end{cases} \tag{5.41}$$

We finalize this section by providing an important property of MDP M_{tr}.

Proposition 5.5. *Given* M_{tr}*, then:*

(1) $\max_{\pi\in\Pi_{M_{tr}}} p^{\pi}_{M_{tr}}(Obj \cup \{q_{abs}\}) = 1$*;*

(2) $p^{\pi}_{M_{tr}}(Obj) = p^{*}_{M}(Obj)$ *for any* $\pi \in \Pi_{M_{tr}}$ *such that* $p^{\pi}_{M_{tr}}(Obj \cup \{q_{abs}\}) = 1$.

Proof. We first prove (1). Let $\pi \in \Pi^{f}_{M}$ such that $p^{\pi}_{M,q}(Obj) = p^{*}_{q}(Obj)$ for all $q \in Q$. Similar to the proof of Lemma **5.1**, we can show that for any $\rho \in Runs^{\pi}(M)$ either $\exists j \in \mathbb{N}$ such that $q^{j}_{\rho} \in Obj$ or $p^{\pi}_{M,q^{j}_{\rho}}(Obj) = 0$, or the set of runs that do not satisfy the first condition has probability zero of being generated. By construction of M_{tr} and since π is optimal, we can convert π to be in $\Pi_{M_{tr}}$ and $p^{\pi}_{M_{tr}}(Obj \cup \{q_{abs}\}) = 1$. It follows that $\max_{\pi\in\Pi_{M_{tr}}} p^{\pi}_{M_{tr}}(Obj \cup \{q_{abs}\}) = 1$. The second property follows from (1) and the construction of M_{tr}. □

Recall that in Problem **5.4** has a solution if and only if it has at least one proper strategy with respect to Q_{abs}, i.e., $\exists \pi \in \Pi_M$ such that $p^{\pi}_{M}(Q_{abs}) = 1$. Thus, there exists at least one proper strategy in M_{tr} with respect to the set $Obj \cup \{q_{abs}\}$. This implies that Problem **5.4** for M_{tr}, $Obj \cup \{q_{abs}\}$, and c as Eq. (5.41) has a solution. Second, the probability of reaching the set Obj in M_{tr} is equal to $p^{*}_{M}(Obj)$ if a proper strategy is selected.

5.6.2 Solution procedure

We follow a similar methodology as in Section 5.5 but we connect Problem **5.2** with Problem **5.4**. Once an attack function A that is a solution of Problem **5.1** is fixed, we obtain the language $\mathcal{L}(S_A/H)$ and consequently the value of $\mathbb{E}^{A}[cost(s)]$. The same idea applies to the MDP M_{tr}; once a proper strategy is fixed, we can calculate the expected cumulative cost value $\mathbb{E}^{\pi}_{M_{tr}}[cumul(Obj \cup \{q_{abs}\}, \rho)]$. The construction of MDP M_{tr} and the set Obj ties these two problems together. The following theorem connects the solution of Problem **5.4** for M_{tr}, the set $Q_{abs} = Obj \cup \{q_{abs}\}$, and the cost function c with the solution of Problem **5.2**.

Theorem 5.2. *Consider the MDP M_{tr}, the set $Q_{abs} = Obj \cup \{q_{abs}\}$, and the cost function c, then*

$$\inf_{A \in \Pi_A \text{ s.t. } win_A = win_{A^{max}}} \mathbb{E}^{A}[cost(s)] = c^{*}_{M_{tr}}(Q_{abs}) \tag{5.42}$$

Again, the proof of the above theorem needs to be written with care even though it appears plausible that the solution of Problem **5.4** provides a solution for Problem **5.2**. We provide intermediate results intuitively and with their formal proofs in the following subsection. Similar to Problem **5.3**, the solution procedure of Problem **5.4** not only outputs the maximum probability $c^{*}_{M_{tr}}(Q_{abs})$ but also outputs $c^{*}_{M_{tr},q}(Q_{abs})$ for all $q \in Q$. In this manner, one can use standard methods to extract an optimal strategy π^{*} that achieves $c^{\pi^{*}}_{M_{tr}}(Q_{abs}) = c^{*}_{M_{tr}}(Q_{abs})$. Therefore, an attack function A^{multi} based on π^{*} such that A^{multi} is a solution of Problem **5.2** can be constructed based on Definition **5.6**.

Example 5.5. *We return to our running example, where we solve Problem **5.4** for the MDP depicted in Fig. 5.3. For simplicity, we assume that $c(q,a) = 2$ if $a \in \Sigma_{att}$. In this case, only one strategy remains in MDP M_{tr}, i.e., only one strategy reaches Obj with probability 0.4. It follows that $c^*_{M_{tr}}(Q_{abs}) = 2 \times 0.4 = 0.8$, which implies that $\mathbb{E}^{A^{multi}}[cost(s)] = 0.8$. The attack function A^{multi} is the same as A^{max} defined in Example **5.3**. This expected cost is the attacker's average cost to reach the critical state (set Obj) or to reach a state where the critical state is unreachable (state q_{abs}).*

5.6.3 Proof of theorem 5.2

Since M_{tr} is constructed directly from M, it is clear that any strategy in M_{tr} can be extended to a strategy in M. In order to fully use the results provided in Section 5.5 for M_{tr}, a technical assumption related to complete and consistent attack functions must be addressed.

For simplicity, we assume that an attacker stops its attack once a string in $U_{unr,A}$ (Eq. (5.9)) is executed, i.e., the attacker cannot reach the critical state at this point. We assume attack functions $A \in \Pi_A$ such that for any $s_1 s_2 \in \mathcal{L}(S_A/H)$, $s_1 \in U_{unr,A}$, and $s_2 \in \Sigma^*$, then $\hat{A}(s_1 s_2) = \hat{A}(s_1)s_2$. Namely, the attacker does not insert or delete events after a string in $U_{unr,A}$ is executed. Using this assumption, all results from Section 5.5 naturally extend to M_{tr}.

In order to prove Theorem **5.2**, we first show that there is a one-to-one map between the strings in $U_{unr,A} \cup U_{uns,A}$ to finite runs in M_{tr} that reach the absorbing set Q_{abs}, where

$$U_{uns,A} = U_{uns} \cap \mathcal{L}(S_A/H) \tag{5.43}$$

Next, we show that the expected cost of the attack function is equivalent to the expected cost of reaching Q_{abs}.

First, we define the set R^{π}_{abs} to be the set of all finite runs that reach Q_{abs}. Formally, this set is defined as:

$$R^{\pi}_{abs} = \{\rho \in Pref^{\pi}(M_{tr}) \mid q_{\rho}^{|\rho|} \in Q_{abs} \wedge q_{\rho}^{|\rho|-1} \notin Q_{abs}\} \tag{5.44}$$

Proposition 5.6. *Given a proper strategy $\pi \in \Pi_{M_{tr}}$ with respect to Q_{abs}, then there exists a bijection between R^{π}_{abs} and $U_{unr,A} \cup U_{uns,A}$. Conversely, given an attack function $A \in \Pi_A$ such that $win_A = win_{A^{max}}$, then there exists a bijection between $R^{\pi_A}_{abs}$ and $U_{unr,A} \cup U_{uns,A}$.*

Proof. Let us define the set $L^{\pi}_{abs} = \{s \in \mathcal{L}(H) \mid s = s_\rho \text{ for } \rho \in R^{\pi}_{abs}\}$. It is enough to show $L^{\pi}_{abs} = U_{unr,A_\pi} \cup U_{uns,A_\pi}$ since each $\rho \in R^{\pi}_{abs}$ generate a unique string in L^{π}_{abs}. The last statement follows since π is deterministic.

First, we show that $L^{\pi}_{abs} \subseteq U_{unr,A_\pi} \cup U_{uns,A_\pi}$. Assume that $s \in L^{\pi}_{abs}$ is generated by $\rho \in R^{\pi}_{abs}$ such that $q_\rho^{|\rho|} = q_{abs}$. Since $p^{\pi}_{M,q_{abs}}(Obj) = 0$ and $p^{\pi}_{M,q_\rho^{|\rho|-1}}(Obj) > 0$, it follows that $s \in U_{unr,A_\pi}$. Now, assume that $s \in L^{\pi}_{abs}$ is generated by $\rho \in R^{\pi}_{abs}$ such that $q_\rho^{|\rho|} \in Obj$. Then $q^{|\rho|}_{\rho,H} = x_{crit}$, and, by Def. **5.6**, $s \in U_{uns,A_\pi}$.

Showing that $U_{unr,A_\pi} \cup U_{uns,A_\pi} \subseteq L^{\pi}_{abs}$ follows the same steps. Let $s \in U_{unr,A_\pi} \cup U_{uns,A_\pi}$, then $\hat{A}_\pi(s)$ is bounded since π is proper. Using Proposition **5.2** and Def. **5.6**, $\exists \rho \in Pref^{\pi}(M_{tr})$ such that $s_\rho = s$ and $t_\rho = \hat{A}_\pi(s)$. If $s \in U_{unr,A_\pi}$, then $q_\rho^{|\rho|} = q_{abs}$ which implies that $s \in L^{\pi_A}_{abs}$. If $s \in U_{uns,A_\pi}$, then $\rho' = \rho\tau(q^{|\rho|}_{\rho,H}, q^{|\rho|}_{\rho,R}, \epsilon) \in R^{\pi}_{abs}$ and $s_{\rho'} = s_\rho\epsilon = s \in L^{\pi}_{abs}$.

The second equality $U_{unr,A} \cup U_{uns,A} \cap \mathcal{L}(S_A/H) = L^{\pi_A}_{abs}$ follows in the same manner. □

Let $c^{\pi}_{M_{tr}}(Q_{abs}) = \mathbb{E}^{\pi}_{M_{tr}}[cumul(Q_{abs}, \rho)]$ be the expected cost given strategy $\pi \in \Pi_{M_{tr}}$. Finally, we state a proposition which is a consequence of Propositions **5.3** and **5.6**.

Proposition 5.7. *Given a strategy $\pi \in \Pi_{M_{tr}}$ such that π is proper with respect to $Q_{abs} = Obj \cup \{q_{abs}\}$, then $c^{\pi}_{M_{tr}}(Q_{abs}) = \mathbb{E}^{A_\pi}[cost(s)]$. Conversely, given a solution of Problem **5.1** $A \in \Pi_A$, then $\mathbb{E}^{A}[cost(s)] = c^{\pi_A}_{M_{tr}}(Q_{abs})$.*

Proof. Given $\pi \in \Pi_{M_{tr}}$, π being a proper strategy, then since R^{π}_{abs} is countable:

$$c^{\pi}_{M_{tr}}(Q_{abs}) = \mathbb{E}^{\pi}_{M_{tr}}[cumul(Q_{abs}, \rho)] \tag{5.45}$$

$$\overset{p^{\pi}_{M_{tr}}(Q_{abs})=1}{=} \mathbb{E}^{\pi}_{M_{tr}}[\sum_{i=0}^{\min\{j:q^j_\rho \in Q_{abs}\}} c(q^i_\rho, a^i_\rho)] \tag{5.46}$$

$$\overset{\text{Eq.}(5.44)}{=} \sum_{\rho \in R^{\pi}_{abs}} \sum_{i=0}^{|\rho|-1} c(q^i_\rho, a^i_\rho) Pr^{\pi}_{M_{tr}}(< \rho >) \tag{5.47}$$

$$\overset{\text{Prop.}\mathbf{5.3}}{=} \sum_{\rho \in R^{\pi}_{abs}} \sum_{i=0}^{|\rho|-1} c(q^i_\rho, a^i_\rho) L_p(S_{A_\pi}/H)(s_\rho) \tag{5.48}$$

$$\overset{\substack{\text{Prop.}\mathbf{5.6} \\ \text{Eq.}(5.41)}}{=} \sum_{s \in L^{\pi}_{abs}} \sum_{i=1}^{|\hat{A}(s)|} w(e^i_{\hat{A}(s)}) L_p(S_{A_\pi}/H)(s) \tag{5.49}$$

$$\overset{\text{Eq.}(5.10)}{=} \sum_{s \in L^{\pi}_{abs}} cost(s) L_p(S_{A_\pi}/H)(s) \tag{5.50}$$

$$= \mathbb{E}^{A_\pi}[cost(s)] \tag{5.51}$$

The equality $\mathbb{E}^{A}[cost(s)] = c^{\pi_A}_{M_{tr}}(Q_{abs})$ follows by reversing the previous steps, i.e., starting from Eq. (5.51) to Eq. (5.45). Reversing the previous steps, we can show that for a given attack function A and its derived strategy π_A, then $\mathbb{E}^{A}[cost(s)] = c^{\pi_A}_{M_{tr}}(Q_{abs})$. □

Finally, the proof of Theorem **5.2** follows directly from Proposition **5.7** and Proposition **5.5**. Let $\pi^* \in \Pi_{M_{tr}}$ be a strategy such that $c^{\pi^*}_{M_{tr}}(Q_{abs}) = c^{*}_{M_{tr}}(Q_{abs})$, then $c^{\pi^*}_{M_{tr}}(Q_{abs}) = \mathbb{E}^{A_{\pi^*}}[cost(s)]$ and $p^{\pi^*}_{M}(Obj) = win_{A^{max}}$. It follows that for any $A \in \Pi_A$ such that $win_A = win_{A^{max}}$, then $\mathbb{E}^{A}[cost(s)] = c^{\pi_A}_{M_{tr}}(Q_{abs}) \geq c^{*}_{M_{tr}}(Q_{abs}) = c^{\pi^*}_{M_{tr}}(Q_{abs}) = \mathbb{E}^{A_{\pi^*}}[cost(s)]$. Consequently, $\mathbb{E}^{A^{multi}}[cost(s)] = \mathbb{E}^{A_{\pi^*}}[cost(s)]$ and $A^{multi} = A_{\pi^*}$ is a solution of Problem **5.2**.

5.7 Examples

We developed a tool to automatically construct the MDP M, as in Definition **5.4**, as input to the PRISM model checker [58] and to construct an attack function based on the optimal strategy out-

putted by PRISM. PRISM has support for solving both MDP Problems **5.3** and **5.4**. In fact, PRISM supports multi-objective strategy generation [59]. The tool includes efficient model checking engines, e.g., binary decision diagrams, and supports different MDP solution methods, e.g., linear programming, value iteration, etc. Our evaluation was done on a Linux machine with 2.2GHz CPU and 16GB memory. Our software tool and the complete models of the examples described in this section are available at: `https://gitlab.eecs.umich.edu/M-DES-tools/desops`.

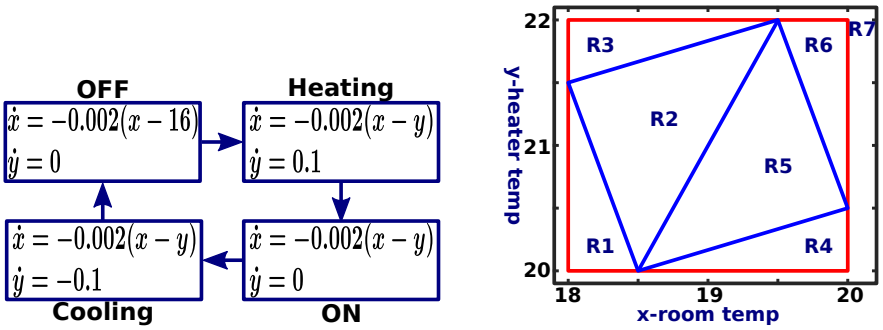
5.7.1 Temperature control

We consider the thermostat system described in [60]. The system has four modes with dynamics described by differential equations as depicted in Fig. 5.4((a)), where x denotes the room temperature and y denotes the temperature of the heater. Moreover, it transitions among the different modes in the order shown in 5.4((a)). This system is abstracted to a discrete model based on the partition plane with 7 regions as shown in Fig. 5.4((b)). The discrete model has 7 uncontrollable events (R1-R7) and 4 controllable events (OFF, Heating, ON, Cooling). The transition relation among the regions in each mode is obtained based on its dynamics.

Different from [60], we consider a probabilistic transition relation among these regions, where the probability captures the likelihood of transitioning between regions. In [60], a supervisor guarantees that the controlled system *never* reaches region 7 (R7). A simulation result illustrating a continuous implementation of this supervisor is shown in Fig. 5.5((a)). The system starts in the ON mode in region 6 (R6). Table 5.1 summarizes the number of states for H and R in this example.

Example	$\lvert X_H \rvert$	$\lvert X_R \rvert$	$\lvert Q \rvert$	$\lvert Q_{tr} \rvert$	$time_1(s)$	$time_2(s)$
Temp.	62	20	427	22	3.50	7.93
Robot	90	63	5193	63	167.1	716.58

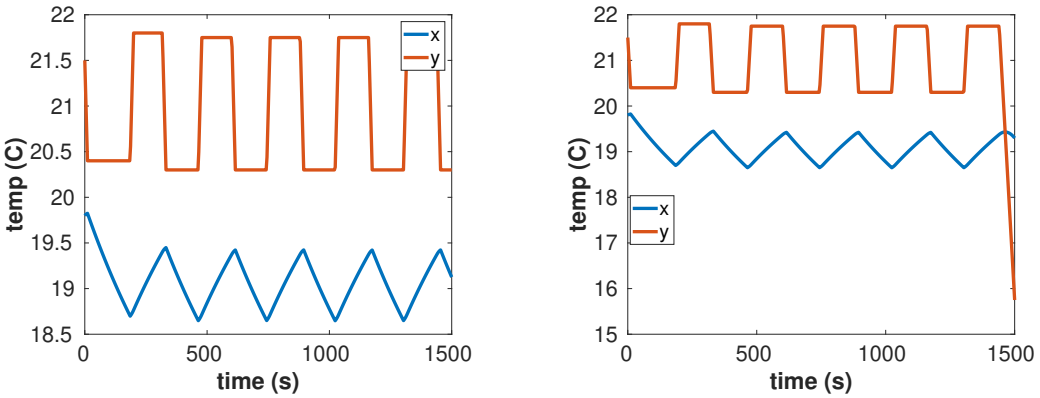
Table 5.1: Summary of results for the examples



(a) Thermostat system dynamics (b) State space partition

Figure 5.4: Four-mode thermostat system

First, we synthesize a maximal reachability attack function assuming that the attacker can manipulate both the room temperature and the heater temperature. As expected, there exists an attack function that reaches region 7 with probability one. One attack function is depicted in Fig. 5.6. The attacker waits for the system to reach region 4 to then delete the reading R4 and replace it by R5 until the system reaches R7. A simulation result illustrating a continuous implementation of this attacked system is shown in Fig. 5.5((b)). The attack starts at time 1375 and by time 1460 the controlled system has reached region 7.



(a) Simulation without attack (b) Simulation with max. reach. attack

Figure 5.5: Four-mode thermostat simulation

Next we consider the synthesis of an attack function based on multiple-objectives (Section 5.6). We consider that each attack modification has weight one, i.e., $w(e) = 1$ if $e \in \Sigma_{att}$. In this

manner, we can synthesize an attack function that reaches region 7 with probability one and with an expected cost of 8.5, i.e., it takes on average 8.5 edits until the system reaches region 7. As a comparison, if we switch infimum by supremum in Problem **5.2**, then we can synthesize an attack function with maximum expected cost. In this example, it takes on average 266.5 edits for this attack function reaches to region 7 with probability one.

Table 5.1 summarizes the results of this example. The MDP M has 427 states while M_{tr} has 22 states. Moreover, it takes 3.50 seconds to obtain a solution of Problem **5.1** while it takes 7.93 seconds to obtain a solution of Problem **5.2**.

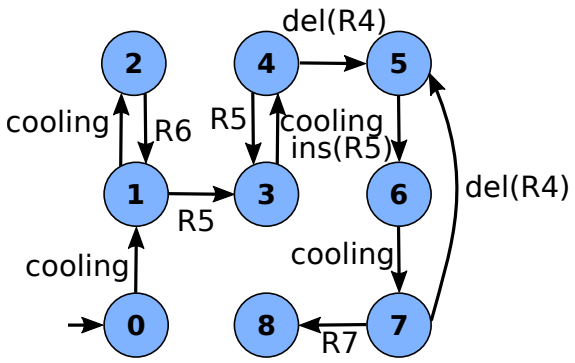
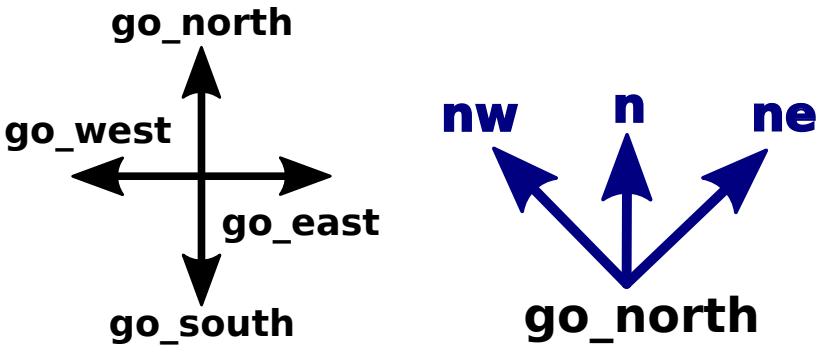


Figure 5.6: Max. reach. attack function encoded as automaton

5.7.2 Robot motion planning

We consider a robot is navigating an area that has been partitioned as a grid. The robot is assumed to have four different movement modes that are modeled as controllable events. Figure 5.7((a)) shows these four possible modes. Each mode represents the possible movements of the robot, e.g., go_north captures the action of commanding the robot to go north. After a mode is selected the robot moves to a new position and outputs its relative movement as a sensor reading. To capture possible disturbances, the relative movement is probabilistic and possibly different than the action selected. For example, Fig. 5.7((b)) depicts the possible sensor reading after mode go_north is selected. All other modes capture similar disturbances as shown in Fig. 5.7((b)). Therefore, the system has 8 uncontrollable events, i.e., $\Sigma_{uctr} = \{n, s, w, e, nw, ne, sw, se\}$.

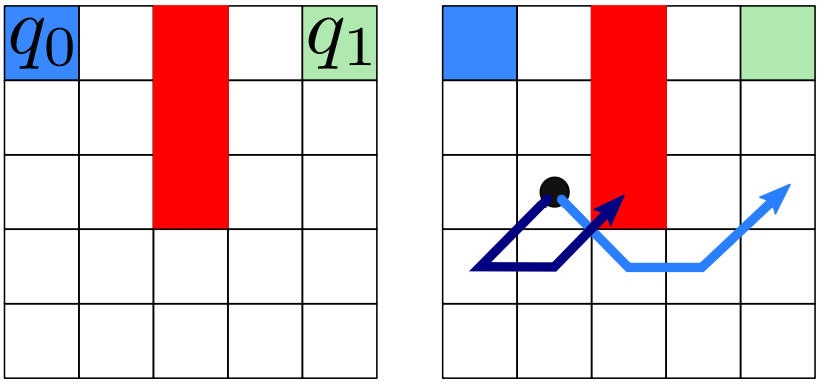


(a) Robot movement modes

(b) Robot possible sensor readings after mode go_north is selected

Figure 5.7: Robot model

The robot moves freely in the workspace shown in Fig. 5.8((a)). Its initial state is the blue cell denoted as q_0. Moreover, the red cells are considered to be obstacles. A supervisor is designed to enforce the following properties: (1) the robot can always access states q_0 and q_1; (2) avoid the obstacle.



(a) Robot workspace

(b) Robot simulation with max. reach. attack

Figure 5.8: Robot workspace and attack simulation

First, we synthesize a maximal reachability attack function assuming that the attacker can manipulate only events that represent disturbances, i.e., $\Sigma_{sen} = \{ne, nw, se, sw\}$ and its goal is to crash the robot into the obstacle. There exists a maximal reachability attack function that crashes the robot into the obstacle with probability one. Next, consider the multi-objective problem with weight function defined as: $w(e) = 1$ if $e \in \Sigma_{att}$, $w(e) = 0$ otherwise. We can synthesize an attack

function that reaches the obstacle state with probability one and with an expected cost of 10.25, i.e., it takes on average 10.25 edits until the robot crashes into the obstacle.

Based on the multi-objective attack function, we show a specific attack scenario of the attack function in Fig. 5.8((b)). The robot reaches the position marked by the black dot and it selects action go_south, but it moves to sw. At this point, the attacker replaces event sw by event se and waits for events e and ne. The trajectory in dark blue represents the exact trajectory that the robot executes, while the trajectory in light blue represents the fictitious trajectory received by the supervisor. This is one successful scenario for this multi-objective attack function.

Table 5.1 summarizes the results of this example. The MDP M has 5193 states while M_{tr} has 63 states. Moreover, it takes 167.1 seconds to obtain a solution of Problem **5.1** while it takes 716.58 seconds to obtain a solution of Problem **5.2**.

5.8 Conclusion

We have considered the problem of synthesis of attack functions for sensor deception attacks at the supervisory layer 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 are quantified by the likelihood of reaching the critical state. We investigated the problem of synthesizing attack functions with the maximum likelihood of reaching the critical state. In order to constrain the attacker behavior, we posed a second problem where each attack edit is costly. In the second problem, the attacker has two objectives: (i) reach the critical state with the maximum likelihood; (ii) minimize the expected cost while performing (i).

In order to solve these problems, we leveraged techniques from MDPs. Namely, we reduced these problem to two well known problems in MDPs: the probabilistic reachability problem and the stochastic shortest path problem. Finally, we presented two illustrative examples to demonstrate the results of our work.

CHAPTER VI

Synthesis of supervisors robust against sensor deception attacks

6.1 Introduction

The previous two chapters focus on the synthesis of attack strategies. They consider the attacker's perspective given a fixed and known supervisor. Although the results in the previous chapters show the possible vulnerabilities of closed-loop systems, they do not provide means of "fixing" these possible vulnerabilities. In order to address these vulnerabilities, we must change our attention from the attacker to the supervisor. In other words, we search for supervisors that are robust against sensor deception attacks. Herein, we consider the "dual" problem of synthesizing a supervisor that will be robust against sensor deception attacks; thus, this work focuses on *defense* strategies. Pictorially, the behavior generated by the feedback control system under attack should never intersect the critical region, as shown in Figure 6.1.

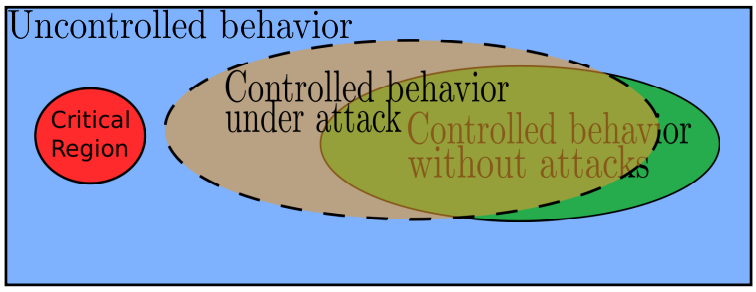


Figure 6.1: Robust controlled system under attack

We study two different methodologies to solve this problem: one uses graph-games methods together with supervisory control methods where the other uses only supervisory control methods.

There exists a trade-off between these two methodologies. While the method relying on supervisory control techniques is computationally more efficient than the graph-game technique, it does not provide flexibility on the selection of the robust supervisor. This flexibility on choosing a robust supervisor is feasible with the graph-game technique but the price for it is a more computationally "expensive" method.

The supervisory control methodology comprises of two steps. In the first step, we build an *augmented plant*, called attacked plant, to capture the interaction of the attacker under the constraints of the plant model. The attacked plant can be built in a manner that accounts for all possible attacks or can be based on a known attack model. The attacked plant captures at the same time the execution of events in the original plant and the information received by the supervisor. The second step poses a supervisory control problem for the attacked plant under the specification that states that cause damage to the plant should never be reached. Specifically, this *supervisory control problem* becomes an instance of supervisory control with *arbitrary control patterns* under partial observation, for which the existing theory of supervisory control of discrete event systems is leveraged [47, 61, 62]. We show that the solution of the supervisory control problem for the attacked plant provides a solution for the problem addressed.

The second solution methodology leverages techniques from games on automata under imperfect information and from supervisory control of partially-observed discrete event systems. It also comprises of two steps. We build a *game arena* to capture the interaction of the attacker and the supervisor, under the constraints of the plant model. The arena defines the solution space over which the problem of synthesizing supervisors with the desired robustness properties can be formulated. In this solution space, called *meta-system*, we use supervisory control techniques to enforce such robustness properties. We leverage the existing theory of supervisory control under partial observation [35, 63, 64, 65] to solve this *meta-supervisory control problem*. As formulated, the meta-supervisory control problem has a unique solution. This solution embeds *all* robust supervisors for the original plant, thereby providing a complete characterization of the problem addressed.

The remainder of this chapter is organized as follows. Section 6.2 formalizes the problem statement for this chapter. The supervisory control solution method is presented in Section 6.3. The second solution method is described in Section 6.4. Finally, we conclude this chapter in Section 6.5.

Related work

There is a vast literature on robust control in discrete event systems [66, 67, 39, 68, 38, 36, 37]. However, robustness in the previous literature is related to communication delays [38, 36], loss of information [37], or model uncertainty [66, 67, 68, 39]. Exceptions to that are [14, 13, 18, 28], where the problem of synthesizing supervisors robust against attacks was investigated. The results of [18] are related to actuator deception attacks.

Our work differs from [14, 13, 28] as we provide a general game-theoretical framework that solves the problem of synthesizing supervisors robust against general classes of sensor deception attacks. The solution methodology in [14, 13, 28] follows the standard supervisory control solution methodology, where only results about *one* robust supervisor against a *specific class* of sensor deception attacks is provided. Conditions on the existence of robust supervisors against a possible set of sensor deception attacks with a normality condition on the plant are provided in [14]. A methodology to synthesize the supremal controllable and normal robust supervisor against *bounded* sensor deception attacks is given in [13]. Finally, [28] provides a methodology to synthesize a maximal controllable and observable supervisor against *unbounded* sensor deception attacks.

The game-theoretical framework adopted in this work provides necessary and sufficient conditions for the problems of existence and synthesis of robust supervisors against general classes of sensor deception attacks. This game-theoretical approach provides a structure that incorporates all robust supervisors against sensor deception attacks. Different robust supervisors can be extracted from this structure, e.g., maximal controllable and observable, supremal controllable and normal, etc. In fact, the robust supervisors from [14, 28] are embedded in this structure. Moreover,

there is a natural extension of our solution methodology such that robust supervisors from [13] are embedded in this structure.

In summary, our work does not impose any normality condition as imposed in [14, 13] and studies synthesis and existence of robust supervisors against *any* sensor deception attack. Our approach considers both bounded and unbounded sensor deception attacks. Moreover, *necessary and sufficient* conditions are provided for the existence and synthesis of robust supervisors, whereas in [14] only existence conditions are provided and in [13] only a sufficient condition is provided. Finally, our synthesis solution methodology has double exponential worst-case complexity while the one in [13] has triple exponential worst-case complexity.

Note that our work also differs from the work of cyber-security on continuous/discrete-time control systems. There exists a vast literature on cyber-security problems for continuous-variable systems, e.g., see [69, 70] for recent results. Our work focuses on systems that have discrete-event models, i.e., systems that by nature are driven by events or time-driven systems that have been abstracted to a discrete-event model.

Our problem formulation is based on the following considerations. The attacker attacks whenever possible, by deleting plant events upon their occurrence or by inserting fictitious ones allowed by the supervisor. In this context, we wish to synthesize a supervisor that provably prevents the plant from reaching a critical state despite the fact that the information it receives from the compromised sensors may be inaccurate.

6.2 Problem formulation

In this chapter, we return to the standard supervisory control framework presented in Chapter III. The attacker hijacks the communication channel between the plant and the supervisor and it can modify the readings of events in Σ_{sen}. The attacker is modeled by an automaton A as described in Section 3.2.2. Based on Σ_{sen}, both the plant G and the supervisor R are augmented to included attacker actions. The augmented versions of the plant and the supervisor are denoted the attacked

plant G_a and the attacked supervisor R_a, Defs. **3.3** and **3.4**. In this manner, the attacked closed-loop system is defined as the parallel composition of A, G_a, and R_a. This framework is summarized in Fig. 6.2.

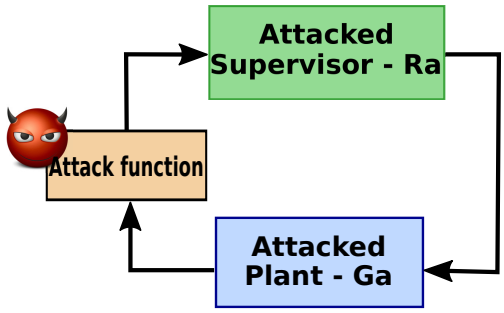


Figure 6.2: Augmented supervisory control framework under sensor deception attack

We investigate the problem of synthesizing a supervisor R robust against the attack strategy A. We assume that the plant G contains a set of *critical* states defined as $X_{crit} \subset X_G$; these states are unsafe in the sense that they are states where physical damage to the plant might occur. Although damage is defined in relation to the set X_{crit}, it could be generalized in relation to any regular language by state space refinement.

Definition 6.1. *Supervisor R is* robust *(against sensor deception attacks) with respect to G, X_{crit} and A, if for any $s \in \mathcal{L}(S_A/G)$ then $\delta_G(x_{0,G}, s) \notin X_{crit}$.*

The definition of robustness is dependent on the attack strategy A. Recall that the all-out strategy (Section 3.4.1) encompasses all other attack strategies [11]. Therefore, a supervisor that is robust against the all-out strategy is robust against any other A [28].

Problem 6.1 (Synthesis of Robust Supervisor)**.** *Given G, X_{crit} and an attack strategy A, synthesize a robust supervisor R, if one exists, with respect to G, X_{crit} and A.*

We are asking that the robust supervisor should prevent the plant from reaching a critical state regardless of the fact that it might receive inaccurate information. In other words, the supervisor reacts to every event that it receives, but since it was designed to be robust to A, the insertions and deletions that A performs will never cause G to reach X_{crit}. This will be guaranteed by the solution procedures presented in the next section.

6.3 Synthesis of robust supervisor via supervisory control theory

The left diagram of Figure 6.3 pictorially describes sensor deception attacks in the supervisory control framework, where the attacker intervenes in the communication channel between the plant's sensors and the supervisor. The attacker has the ability to observe the same observable events as the supervisor. Even more, it has the ability to alter some of the sensor readings in this communication channel, where "alter" means that it can insert or delete events. The subset of affected sensor readings is defined as the *compromised event set* and denoted by $\Sigma_{sen} \subseteq \Sigma_{obs}$.

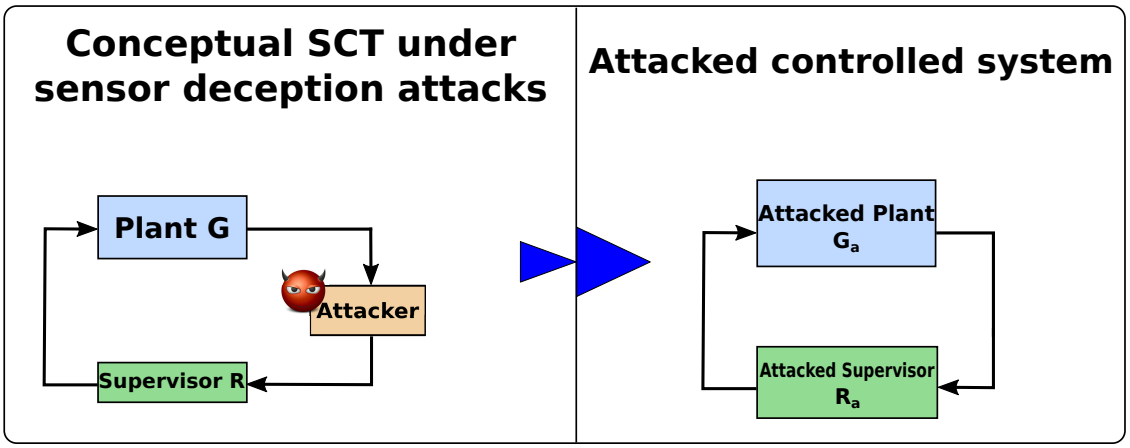


Figure 6.3: Sensor Deception Attack Framework

We recall the conceptual diagram of sensor deception attacks in the supervisory control framework, shown in Figure 6.3, as we have discussed in Chapter III. We transform the original controlled system with attacker assumptions, right diagram in Figure 6.3, into an attacked controlled system, left diagram in Figure 6.3. In this attacked controlled system, we perform our analysis and provide guarantees about robustness against sensor deception attacks.

Specifically, we pose a supervisory control problem for the attacked plant under the specification that states that cause damage to the plant should never be reached. This *supervisory control problem* becomes an instance of supervisory control with *arbitrary control patterns* under partial observation, for which the existing theory of supervisory control of discrete event systems is lever-

aged [47, 61, 62]. We show that the solution of the supervisory control problem for the attacked plant provides a solution for the problem addressed in this work. We also provide results on the existence of a supremal robust supervisor.

6.3.1 Supervisory control with control patterns

In this section, we need to leverage the partially observed supervisory control problem with arbitrary control patterns (SCP-AP) studied in [47, 61, 62]. In this problem, the supervisor S is defined based on an arbitrary set $C \subseteq \Gamma$, i.e., $S : P_{\Sigma\Sigma_{obs}}(\mathcal{L}(G)) \rightarrow C$. Now, we recall a result from [62].

Proposition 6.1. *[62] Let $K \subseteq \mathcal{L}(G)$ be a non-empty and prefix-closed language and let $C \subseteq \Gamma$ be the available control patterns set. There exists a supervisor $S : P_{\Sigma\Sigma_{obs}}(\mathcal{L}(G)) \rightarrow C$ such that $\mathcal{L}(S/G) = K$ if and only if K satisfies the following condition: for any $s \in K$ and $t \in P^{-1}_{\Sigma\Sigma_{obs}}[P_{\Sigma\Sigma_{obs}}(s)] \cap K$, there exists a control pattern $\gamma \in C$ such that $\gamma \cap \Sigma_{\mathcal{L}(G)}(t) = \Sigma_K(t)$, where $\Sigma_K(s) = \{e \in \Sigma \mid se \in K\}$ is the one event continuation of string $s \in K$.*

Proposition **6.1** reduces to the standard controllability and observability conditions when $C = \Gamma$ [62]. When K does not satisfy Proposition **6.1**, the set $\mathcal{CO}(K) = \{K' \subseteq \mathcal{L}(G) \mid K' = pr(K') \subseteq K$ s.t. K' satisfies Proposition **6.1**$\}$ is defined. Note that, $\mathcal{CO}(K)$ is non-empty since $\emptyset \in \mathcal{CO}(K)$. Similarly to the standard partially observed supervisory control problem, there does not exist in general a supremal element in $\mathcal{CO}(K)$.

6.3.2 Reducing the robust supervisor problem

Inspired by the technique used in [11], the model G_a[1] defined in Chapter III becomes the system at the center of our study. Since the attacked plant G_a captures the behavior of plant G under a sensor deception attack over Σ_{sen}, our solution technique poses a supervisory control problem

[1]Definition **3.3** describes G_a under attack of the all-out attack strategy. If there exists prior knowledge of the attacker, described by A, then G_a is redefined as $G_a := G_a||A$, where the G_a on the right is the one defined in Def. **3.3**.

directly at the attacked plant G_a. Although it appears that we could directly apply standard supervisory control techniques, the events in Σ_{att} prevent us to do so because we cannot assign them to the controllable/uncontrollable and the observable/unobservable partitions. Next, we provide a solution to this problem starting with the controllability issue.

The controllability of the events in G_a must be defined carefully since the supervisor cannot directly disable insertions nor deletion events. However, if the compromised event is controllable then the supervisor can disable this event and indirectly disable its insertion and deletion events.

Recall that in the construction of R_a, deletion events are self-loops in the states of R_a where the legitimate events are defined. They are self-loops since the supervisor does not receive any information. On the other hand, insertion events are defined in parallel to their legitimate events $e \in \Sigma_{sen}$ [2]. Therefore if a compromised event is enabled in R, then both its insertion and deletion events are enabled in R_a. This control pattern is denoted as C_a based on events $e \in \Sigma_{sen}$ and defined as:

$$C_a = \{\gamma \in 2^{\Sigma_{all}} \mid \gamma \subseteq \Sigma_{uctr} \wedge (\forall e \in \gamma \cap \Sigma_{sen}, ins(e) \in \gamma \wedge del(e) \in \gamma)\}. \tag{6.1}$$

The control patterns created by Σ_{att} solve the issue related to controllability of events in Σ_{att} but they do not solve the issue related to observability. P^S projects attacked strings from G_a to strings executed in G, but it does not capture how attacked strings are perceived by the supervisor. Even though the system R_a/G_a captures exactly the closed-loop behavior of the attacked system, we are interested in synthesizing a supervisor R that has Σ^*_{obs} as input. In other words, we use R_a/G_a to test if a supervisor R with Σ^*_{obs} as input is safe.

Let us analyze G_a with the projection P^S. Recall that Definition **3.3** adds self-loops of insertions events at each state of G. It also adds transitions with deletion events parallel to transitions with their legitimate event. Therefore, if we use the projection P^S in G_a, then the insertion events are mapped to their legitimate events and deletion events are mapped to the empty string. This

[2]For simplicity let us assume that the self-loops with insertion events are not defined. This is without loss of generality since these insertion events do not change neither the state of the plant nor the state of the supervisor.

operation generates a nondeterministic automaton, which we want to avoid. We prefer a partially observed system to be consistent with standard supervisory control.

We consider the following procedure to eliminate the nondeterminism generated by P^S and analyze strings from G_a as they are seen by the supervisor.

Definition 6.2. *Given G_a, we define $G_m := (X_{G_m}, \Sigma_{all}, \delta_{G_m}, x_{0,G_m})$ as:*

$$X_{G_m} := X_{G_a} \cup (X_{G_a} \times \{ins(e) \mid e \in \Sigma_{sen}\})$$

$$\delta_{G_m}(x,e) := \begin{cases} \delta_{G_a}(x,e) & \textit{if } x \in X_{G_a} \textit{and } e \notin \Sigma_{ins} \\ (x,e) & \textit{if } x \in X_{G_a} \textit{and } e \in \Sigma_{ins} \\ x_1 & \textit{if } x \in X_{G_a} \times \{ins(\sigma) \mid \sigma \in \Sigma_{sen}\},\ x = (x_1, e_1),\ e = \mathcal{M}(e_1) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

$$x_{0,G_m} := x_{0,G_a}$$

Given the automaton G_m, we can know discuss the observable events of this system. After the execution of an event $ins(e)$, the legitimate counterpart e is executed in G_m. The execution of an event $del(e)$ is still considered unobservable in G_m. Therefore, we can now specify the events in Σ_{att} as unobservable. Moreover, the events that are unobservable in G continue to be unobservable in G_m. Thus, the unobservable event set for G_m is $\Sigma_{m,uo} := \Sigma_{uobs} \cup \Sigma_{att}$ and the observable set is $\Sigma_{m,o} := \Sigma_{obs}$. We also define $P_{\Sigma_{all}\Sigma_{m,o}}$ as the projection operation of strings in Σ_{all} to $\Sigma_{m,o}$. The following proposition shows that the language observed by a supervisor R through G_a is the same as the one observed by G_m.

Proposition 6.2. $P_{\Sigma\Sigma_{obs}}(P^S(\mathcal{L}(G_a))) = P_{\Sigma_{all}\Sigma_{m,o}}(\mathcal{L}(G_m))$

Proof. It follows from the definitions of G_m, P^S and $P_{\Sigma_{all}\Sigma_{m,o}}$. □

The attacked system G_m has $\Sigma_{m,o} = \Sigma_{obs}$ as the set of observable events in G_m and C_a as the set of control patterns. Therefore, we search for a supervisor that only selects control decisions

with respect to the control pattern C_a instead of the entire set Γ. Fortunately, the work in [62] provides the necessary results on obtaining maximal controllable and observable sublanguages with constrained control patterns. To be able to use these results, we need the following proposition.

Proposition 6.3. *The set C_a is closed under union.*

Proof. Let $\gamma_j \in C_a$ and $c_j = \gamma_j \cap \Sigma_{sen}$ for $j \in \{1,2\}$. Since $\gamma_j \subseteq \Sigma_{uctr}$ for $j \in \{1,2\}$, then $\gamma_1 \cup \gamma_2 \subseteq \Sigma_{uctr}$. Also $\forall e \in c_j$, we have that $ins(e) \in \gamma_j$ and $del(e) \in \gamma_j$, $j \in \{1,2\}$. Thus, $\forall e \in c_1 \cup c_2$, we have that $ins(e) \in \gamma_1 \cup \gamma_2$ and $del(e) \in \gamma_1 \cup \gamma_2$. In this manner, we conclude that $\gamma_1 \cup \gamma_2 \in C_a$. □

Since C_a is closed under union, the results in [62] are applicable to G_m. Let $K = \mathcal{L}(Ac(G_m, X_{crit}))$ be the language specification on $\mathcal{L}(G_m)$, where $Ac(G_m, X_{crit})$ is the *accessible* subautomaton of G_m after deleting states $X_{crit} \subset X_{G_m}$. From [62], it follows that for any $K' \in \mathcal{CO}(K)$, if $K' \neq \emptyset$, there exist a supervisor $S : P_{\Sigma\Sigma_{obs}}(\mathcal{L}(G_m)) \to C_a$ such that $\mathcal{L}(S/G_m) = K'$. Also, the supremal element of $\mathcal{CO}(K)$ does not exist in general. For this reason, we search for a supervisor that generates a maximal K'.

Problem 6.2 (**SCP-AP**-Supervisory Control with Arbitrary Control Patterns)**.** *Given the attacked system G_m, and $K = \mathcal{L}(Ac(G_m, X_{crit}))$. Let $K' \in \mathcal{CO}(K)$ be a maximal sublanguage in $\mathcal{CO}(K)$. Synthesize a supervisor $S : P_{\Sigma\Sigma_{obs}}(\mathcal{L}(G_m)) \to C_a$ that satisfies $\mathcal{L}(S/G_m) = K'$.*

Since the languages $\mathcal{L}(G_m)$ and K are regular languages, the supervisor S can be encoded by a DFA R_m. Next, we define a supervisor R for the Problem **6.1** based on the supervisor R_m. Using this supervisor, we prove that Problem **6.1** is reducible to the **SCP-AP** problem.

Definition 6.3. *Assume that $\mathcal{CO}(K) \neq \emptyset$ in the **SCP-AP** problem. Let the supervisor R_m be a solution for the **SCP-AP** problem. Construct the supervisor $R = (X_R, \Sigma, \delta_R, x_{0,R})$ in the following manner.*

$$X_R := X_{R_m}$$

$$\delta_R(x,e) := \begin{cases} \delta_{R_m}(x,e) & \textit{if } x \in X_R \textit{ and } e \in \Sigma \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

$$x_{0,R} := x_{0,R_m}$$

Definition **6.3** constructs the supervisor R by simply removing the events in Σ_{att}. The transitions with events in Σ_{att} on R_m are self-loops since they are unobservable on the **SCP-AP** problem. We now provide the connection between the Problem **6.1** and the **SCP-AP** problem.

Theorem 6.1. *Problem **6.1** is reducible to the **SCP-AP** problem.*

Proof. The reduction algorithm starts with the construction of the DFA G_a, G_m, and with the set C_a. We first show that if the **SCP-AP** problem does not have a solution, then Problem **6.1** also does not have a solution. The second part of the proof shows that the supervisor R defined in Definition **6.3** is a solution for the **SCP-AP** problem.

The first part of the proof can be shown by contradiction.

For the second part, we assume that $\mathcal{CO}(K) \neq \emptyset$ and that R_m encodes the solution for the **SCP-AP** problem. Let R be the supervisor constructed as in Definition **6.3**. We prove that R is a solution for Problem **6.1**, namely R is robust. We show this result by contradiction.

Assume that R is not robust, which means that $\exists s \in \mathcal{L}(R_a/G_a)$ such that

$$\delta_G(x_{0,G}, P_{\Sigma\Sigma_{obs}}(P^G(s))) \in X_{crit}.$$

Let $v = P_{\Sigma\Sigma_{obs}}(P^S(s))$, then the sequence $c = c_0 c_1 \dots c_{|v|}$, where $c_j = En_R(\delta_R(x_{0,R}, v^j))$, is the sequence of control decisions made by R over the string s until it reaches some state in X_{crit}. In other words, these are the control decisions that allow the string $P^G(s)$ to be executed. Similarly, we find the control decisions based on R_a that allows s in G_a. Namely, the sequence of control decisions $c' = c'_0 c'_1 \dots c'_{|v|}$, where $c'_j = \Gamma_{R_a}(\delta_{R_a}(x_{0,R_a}, v^j))$.

Next, we show that there exist a string $t \in \mathcal{L}(G_m)$ where $P_{\Sigma_{all}\Sigma_{m,o}}(t) = P_{\Sigma\Sigma_{obs}}(P^S(s)) = v$. To construct string t, we apply the same transformation we used in the construction of G_m based

on G_a. String t is a copy of string s with the following modification: for each insertion event e in s, we replace it by $e\mathcal{M}(e)$. In this manner, $P_{\Sigma_{all}\Sigma_{m,o}}(t) = P_{\Sigma\Sigma_{obs}}(P^S(s)) = v$.

The sequence of control decisions made by R_m over t is c' since the control decisions at each state of R_m are the same at the corresponding state in R_a and $P_{\Sigma_{all}\Sigma_{m,o}}(t) = P_{\Sigma\Sigma_{obs}}(P^S(s)) = v$. Therefore, $t \in \mathcal{L}(R_m/G_m)$ since $s \in \mathcal{L}(R_a/G_a)$. Finally, string s leads the attacked system G_a to the critical state. By construction of G_m and t, string t leads the G_m to the critical state. This concludes our proof. Therefore, Problem **6.1** is reducible to the **SCP-AP** problem.

Since $s \in \mathcal{L}(R_a/G_a)$, we have that $t \in \mathcal{L}(R_m/G_m)$. □

6.3.3 Computing a maximal robust supervisor

Although Theorem **6.1** provides the reduction of Problem **6.1** to the partially observed supervisory control problem with arbitrary control patterns, an algorithm to synthesize such supervisor was not provided in [62]. For this reason, we provide a modified version of the *VLP-PO* algorithm presented in [71][3]. We provide necessary modifications to compute maximal control policies given a set of control patterns.

Algorithm 6 provides the modifications to compute the maximal control policies given a set of control patterns $C \subseteq \Gamma$. The variables ACT and NS are global variables and store the current issued control decision and the current state estimate. Given that the system executed an observable event e, these two variables update accordingly.

We use four functions in Algorithm 6 that we have not yet defined, namely $UR_{+\gamma}$, SmContPat, V and Elist.pop(). We define them based on automaton G and specification K. The $UR_{+\gamma}$ finds the set of states reached via a string of unobservable events or via a string of unobservable events concatenated with *one* observable event.

$$UR_{+\gamma}(X \subseteq X_G) := UR_\gamma(X) \cup \{x \in X_G \mid \exists y \in UR_\gamma(X) \text{ and } e \in (\Sigma_{obs} \cap \gamma), x = \delta_G(y,t)\}$$

[3]The *VLP-PO* algorithm computes online and offline maximal control policies. However, any algorithm in the literature that solves the standard supervisory control problem with partial observation can be adapted as we do for VLP-PO.

The function $SmContPat(\gamma \in \Gamma, C)$ returns a smallest control decision $\gamma' \in C$ such that $\gamma \subset \gamma'$. The cost function $V : X_G \to \{0, \infty\}$ holds the information obtained by solving the fully observed supervisory control problem, i.e., $V(x) := \infty$ if x has an uncontrollable trace starting at state x that violates spec K. Lastly, the function *EList.pop()* output and removes the first element of the EList.

Algorithm 6 VLP-PO with Control Patterns

function VLP-POCP($e \in \Sigma_{obs} \cup \{\epsilon\}$)
  $NS = \delta_G(PS, e)$
  $ACT =$ Control-Actions(NS, Event-Ordering, C)
  $PS = UR_{\text{ACT}}(NS)$
function CONTROL-ACTIONS($S \subseteq X_G$, **EList: Ordered-**Σ_{ctr}, Control Pattern Set C)
  $ACT = SmContPat(\Sigma_{uctr}, C)$; $Pt = 1$;
  while EList$\neq \emptyset$ **do**
    if $Pt >$ |EList| **then**
      $ACT = ACT \cup$ EList; Return;
    $e =$ EList[Pt]
    $\gamma = SmContPat(\{e\}, C)$
    if $UR_{+ACT\cup\{\gamma\}}(S) = UR_{+ACT}(S)$ **then**
      $Pt = Pt + 1$; GOTO 7
    for all $x \in UR_{+ACT\cup\{\gamma\}}(S)$ **do**
      if $V(x) = \infty$ **then**
        EList = EList-γ; GOTO 7
    $ACT = ACT \cup \{\gamma\}$; EList = EList-$\gamma$; $Pt = 1$;
    GOTO 7

Similarly as in the original VLP-PO algorithm, the following initialization of VLP-POCP is necessary: PS = $\{x_{G,0}\}$ prior to the first call, Event-Ordering is a total ordering on Σ_{ctr}, and VLP-POCP(ϵ) must be used to determined the first control action.

Instead of online control decisions, we are interested in obtaining an offline representation of a maximal supervisor. For that, we initialize VLP-POCP and use breadth first search to compute the realization of a maximal supervisor. Since we are only interested in the offline solution, we use $R =$VLP-POCP(G, G_{spec}, C) as the realization of a maximal supervisor computed offline by VLP-POCP, where G_{spec} is an automaton such that $\mathcal{L}(G_{spec}) = K$ and $C \subseteq \Gamma$.

Proposition 6.4. *Let $R_m =$ VLP-POCP(G_m, K, C_a) be the supervisor realization generated by the VLP-POCP algorithm, where $K = \mathcal{L}(Ac(G_m, X_{crit}))$. If $R = \emptyset$ then $\mathcal{CO}(K) = \{\emptyset\}$.*

Proof. It follows from the correctness of the VLP-PO algorithm. □

Theorem 6.2. *If $\mathcal{CO}(K) \neq \{\emptyset\}$ where $K = \mathcal{L}(Ac(G_m, X_{crit}))$, then R_m =VLP-POCP(G_m, K, C_a) is a supervisor realization that satisfies the **SCP-AP** problem.*

Proof. It follows from the correctness of the VLP-PO algorithm. □

The final result about VLP-POCP is related to its complexity. The original VLP-PO algorithm computes control decisions in $O(|\Sigma_{ctr}|^2|X_G|)$. The modified version maintains the same runtime since we only introduce the modification on line 6, which can be done in constant time. Let N denote $|X_G|$. The next proposition states the complexity of our method for Problem **6.1**.

Proposition 6.5. *Problem **6.1** is solved in $O\big((|\Sigma_{ctr}|^2(N + N|\Sigma_{sen}|))2^{N+N|\Sigma_{sen}|}\big)$ time.*

Proof. It follows from the complexity of the VLP-POCP algorithm and G_m. □

Example 6.1. *Let us get back to the intersection example to obtain a robust supervisor for it. The models of G and G_a are shown in Fig. 6.4. We assume that $\Sigma_{obs} := \{R_{int}, B_{int}\}$ and $\Sigma_{sen} := \{R_{int}\}$.*

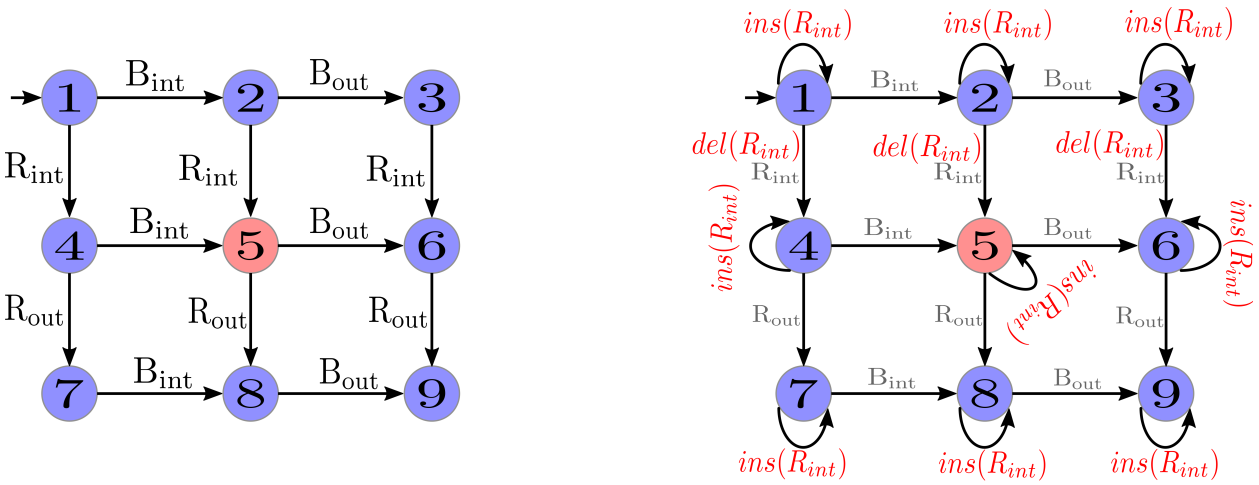


Figure 6.4: Plant G for intersection example (left) and its attacked plant model G_a (right)

*Given G_a, we construct automaton G_m as defined in Def. **6.2**. In this case, G_m will have 18 states since we add a newer state for each state of G while constructing G_m. The specification for G_m is obtained by removing state 5. At this point, we use G_m, the specification and the control*

*pattern C_a as inputs to the modified VLP-PO algorithm. Figure 6.5 depicts a supervisor output of the modified VLP-PO algorithm. This supervisor is then transformed by removing transitions with attack events as described in Def. **6.3**. The robust supervisor obtained is the same as R_2 described in Example **2.4**. There is one more maximal solution for this example, this second solution will be evaluated in the next section.*

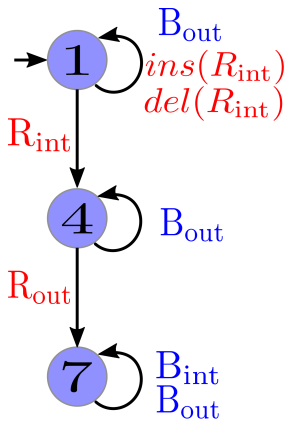


Figure 6.5: Robust supervisor R_m: output of Algorithm 6

Remark 6.1. *Even if G is fully observable, the robust supervisor problem is always transformed to a partially observed supervisory control problem. This is the case since insertion and deletion events are unobservable. In the next section, we provide conditions for the existence of the supremal robust supervisor.*

6.3.4 Condition for the existence of the supremal K'

In the previous sections, we showed how to solve the general problem of synthesizing robust supervisors. We showed that we can map this problem to the **SCP-AP** problem (Problem **6.2**). Since there does not exist in general a supremal supervisor for SCP-AP, then there does not exist in general a supremal robust supervisor. In this section, we investigate sufficient conditions for the existence of such a supremal robust supervisor.

The first condition, also known as normality condition, is well known in the partially observed supervisory control problem, $\Sigma_{ctr} \subseteq \Sigma_{obs}$. It is known that when $\Sigma_{ctr} \subseteq \Sigma_{obs}$, then the observ-

ability plus the controllability conditions are equivalent to the normality condition [34]. Since the supremal controllable and normal sublanguage always exists, then, with this condition imposed, the supremal controllable and observable also exists.

The condition $\Sigma_{ctr} \subseteq \Sigma_{obs}$ is not enough to guarantee the existence of the supremal robust supervisor. Therefore, we must impose a second condition that together with the first condition enforces the existence of the supremal robust supervisor.

In cyber-physical system models, we normally assign actuators signals to be controllable and observable events while sensors signals are uncontrollable. It is based on this assumption that the normality condition was first introduced. Since we study *sensor* deception attacks, we can assume that the set of compromised events is a subset of the uncontrollable events, besides being a subset of observable events. Thus, the second condition is $\Sigma_{sen} \subseteq \Sigma_{obs} \cap \Sigma_{uctr}$.

Corollary 6.1. *If $\Sigma_{ctr} \subseteq \Sigma_{obs}$ and $\Sigma_{sen} \subseteq \Sigma_{uctr}$, then there exists a supremal element $K^{\uparrow} \in \mathcal{CO}(K)$ and R_m =VLP-POCP(G_m, R, C_a) is the supervisor realization such that $\mathcal{L}(R_m/G_m) = K^{\uparrow}$.*

Proof. The proof follows from the fact that $C_a = \Gamma$ in this case. Therefore, we are under the standard partially observed supervisory control problem with $\Sigma_{ctr} \subseteq \Sigma_{obs}$. □

6.3.5 Summary

We leveraged techniques from supervisory control with arbitrary control patterns of partially-observed discrete event systems to develop a solution methodology to prevent damage to the system when some sensor readings may be edited by the attacker. We showed how this problem can be reduced to a partially observed supervisory control problem with arbitrary control patterns. Our solution methodology has single exponential complexity over the number of states of the plant and events. It is more computationally efficient than the previous method in the literature [13].

Since the robust supervisor problem might not have a unique solution, we need to find a flexible method to select robust supervisors. The solution approach shown in this section does not provide

flexibility in the selection of a robust supervisor. It only selects maximal robust supervisors; and Algorithm 6 does not provide flexibility for this selection. In Algorithm 6, the event-ordering determines which maximal solution the algorithm outputs, but this ordering does not provide intuition and much flexibility on the supervisor output. For this reason, we investigate a second method to solve Problem **6.1**; one with more flexibility and intuition on the robust supervisor selection.

6.4 Synthesis of robust supervisor via graph-games

6.4.1 Meta-Supervisor problem

In this section, we present our second approach to solve Problem **6.1**. We briefly explain the idea of this new approach. Figure 6.6 shows the connection of the problem formulation space (left box) and the solution space (right box). The connection between these two spaces is given by the arrows that cross the two boxes. These arrows are labeled by results provided in this section.

In the left box of Fig. 6.6, we have the problem formulation space where the supervisor R is unknown. Based on G, Σ_{obs}, Σ_{ctr} and A, we construct a *meta-system*, called $\mathcal{A}$, in a space where all supervisors are defined. This construction is given in Definition **6.4**. The meta-system is part of the proposed solution space and it is represented in the right box of Fig. 6.6.

Although all supervisors are defined in $\mathcal{A}$, which is shown by Proposition **6.6**, we are only interested in robust supervisors. In order to obtain robust supervisors, we use techniques of partially observed supervisory control theory [64, 65] in the meta-system. The structure $\mathcal{A}^{\text{sup}}$ is obtained via Definition **6.6** and it contains all robust supervisors against sensor deception attacks on Σ_{sen}.

Finally, to return to our problem formulation space, we extract one supervisor, if one exists, from $\mathcal{A}^{\text{sup}}$. Such extraction is given by Algorithm 7.

6.4.1.1 Definition

Inspired by the techniques of two-player reachability games, we construct an arena as it is constructed in these games. In the arena, player 1 represents the supervisor while player 2 represents

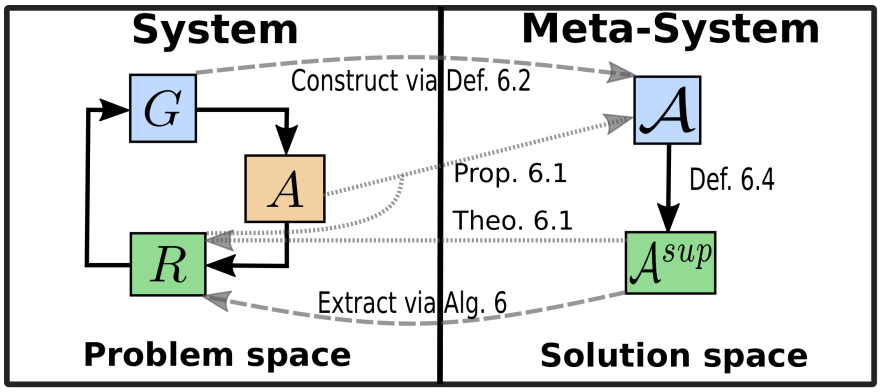


Figure 6.6: Relation of the system and the meta-system

the adversarial environment. The arena exhaustively captures the game between the supervisor and the environment, where the supervisor selects control decisions (Γ) and the environment executes events ($\Sigma_{obs} \cup \Sigma_{att}$). In the arena, player 1's transitions record a control decision made by the supervisor. On the other hand, player 2's transitions represent actions of the plant G or actions of the attacker A. Formally, the arena is defined as follows.

Definition 6.4. *Given plant G and attack function A, we define the arena $\mathcal{A}$ as 4-tuple:*

$$\mathcal{A} := (Q_1 \cup Q_2, A_1 \cup A_2, h_1 \cup h_2, q_0) \tag{6.2}$$

where

- $Q_1 \subseteq 2^{X_G} \times X_A$ *is the set of states where the supervisor issues a control decision. Its states have the form of (S_1, S_2), where S_1 is the estimate of the states (as it is executed by the plant) of G and S_2 is the attacker's state. For convenience we define the projection operators $I_i((S_1, S_2)) = S_i$ for $i \in \{1, 2\}$;*

- $Q_2 \subseteq 2^{X_G} \times X_A \times \Gamma \times (\{\epsilon\} \cup \Sigma_{sen})$ *is the set of states where the adversarial environment issues a decision. Its states have the form $(S_1, S_2, \gamma, \sigma)$, where S_1 and S_2 are defined as in Q_1 states, γ is the last control decision made by the supervisor, and σ is related to inserted events. The event σ is equal to $e \in \Sigma_{sen}$ if the last transition was $ins(e) \in \Sigma_{ins}$, otherwise it is equal to ϵ. We use the same projection operators I_i for states in Q_2 for $i \in \{1, 2\}$;*

- $A_1 := \Gamma$ *and* $A_2 := \Sigma_{obs} \cup \Sigma_{att}$ *are respectively the actions/decisions of player 1 and player 2;*

- $h_1 : Q_1 \times A_1 \to Q_2$ *is built as follows: for any* $q_1 = (S_1, S_2) \in Q_1$ *and* $\gamma \in A_1$

$$h_1(q_1, \gamma) := \big(UR_\gamma(S_1), S_2, \gamma, \epsilon\big) \tag{6.3}$$

- $h_2 : Q_2 \times A_2 \to Q_1 \cup Q_2$ *is built as follows for any* $q_2 = (S_1, S_2, \gamma, \sigma) \in Q_2$*:*

Let $e \in \Sigma_{obs}$*:*

$$h_2(q_2, e) := \begin{cases} \big(\delta_G(S_1, e), \delta_A(S_2, e)\big) & \textit{if } (e \in En_G(S_1) \cap \gamma) \wedge \\ & \quad (e \in En_A(S_2)) \wedge (\sigma = \epsilon) \\ \big(S_1, S_2\big) & \textit{if } (\sigma = e) \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{6.4}$$

Let $e \in \Sigma_{sen}$*:*

$$h_2(q_2, ins(e)) := \begin{cases} \big(S_1, \delta_A(S_2, ins(e)), \gamma, e\big) & \textit{if } (ins(e) \in En_A(S_2)) \wedge (\sigma = \epsilon) \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{6.5}$$

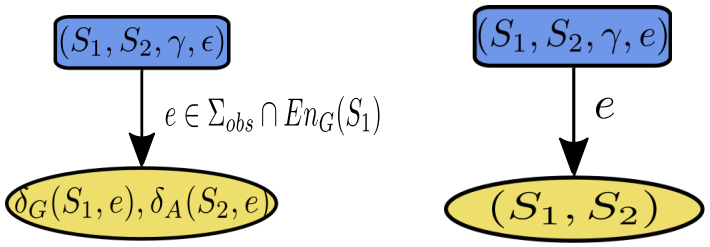
$$h_2(q_2, del(e)) := \begin{cases} \big(UR_\gamma(\delta_G(S_1, e)), \delta_A(S_2, del(e)), \gamma, \epsilon\big) & \textit{if } (e \in En_G(S_1) \cap \gamma) \wedge \\ & \quad (del(e) \in En_A(S_2)) \wedge (\sigma = \epsilon) \\ \textit{undefined} & \textit{otherwise} \end{cases} \tag{6.6}$$

- $q_0 \in Q_1$ *is the initial S-state:* $q_0 := (\{x_{0,G}\}, x_{0,A})$.

We explain the definition of the transition functions h_1 and h_2 in detail. The definition of h_1 is simple and it defines a transition from player 1 to player 2, which records a control decision made by the supervisor, and it updates G's state estimate according to this decision. On the other hand,

h_2 is more complex since player 2 has two types of transitions.

The first type is transitions from player 2 to player 1 which characterizes the visible decision made by the environment and is related to events in Σ_{obs}. These transitions are defined in Eq. (6.4), and they are illustrated in Fig. 6.7. In Fig. 6.7(a), an event $e \in \Sigma_{obs}$ that is feasible in G from some state in S_1 is selected; thus, both the state estimate and the attacker's state are updated. In Fig 6.7(b), $q_2 = (S_1, S_2, \gamma, e) \in Q_2$ is reached after an insertion since $e \neq \epsilon$; thus, G's state estimate and the attacker's state remain unchanged. They remain unchanged because similarly as in Def. **6.2**, our insertion encoding is divided in two steps. The transition in Fig 6.7(b) just reports the legitimate event to the supervisor. The first step, explained in the next paragraph, deals with the state estimate updates.



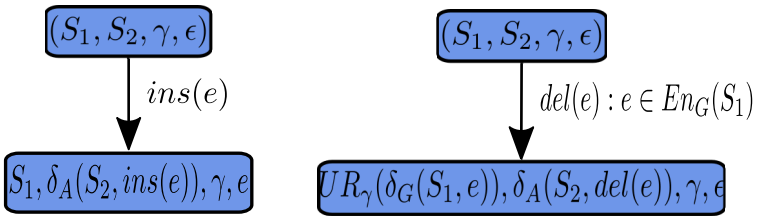
(a) First transition of Eq. (6.4)

(b) Second transition of Eq. (6.4)

Figure 6.7: Transition function h_2 from player 2 to player 1

Transitions from player 2 to itself characterize invisible, from the supervisor's perspective, decisions. They are only defined for events in Σ_{att} . These transitions are defined by Eqs. (6.5-6.6). An attacker can insert any event in $e \in \Sigma_{sen}$, as long as $ins(e)$ is allowed in the current attacker's state. The inserted events $e \in \Sigma_{ins}$ are not going to be seen by the supervisor, as only the attacker knows it decided to insert the event. Insertions will be seen by the supervisor are genuine events. But $ins(e)$ represents here (in the context of the game arena) the intention of the attacker to insert. Equation (6.5) (depicted in Fig. 6.8(a)) is the unobservable part, where an insertion decision was selected and the attacker's state and the fourth component of $q_2 \in Q_2$ are updated. The observable part is shown by Fig. 6.7(b). In the case of a deleted event, from the supervisor's perspective, it is seen as an ϵ event as well. That is, the supervisor cannot change its control decision when the

attacker deletes an event, as shown in Fig. 6.8(b).



(a) Transition of Eq. (6.5)

(b) Transition of Eq. (6.6)

Figure 6.8: Transition function h_2 from player 2 to player 2

Remark 6.2. *The elements of Q_1 and Q_2 are defined such that they incorporate the "sufficient information" (in the sense of* information state *in system theory) that each player needs to make its respective decision. Equations (6.3-6.6) guarantee by construction that the updates of the information states are consistent with the plant dynamics and the actions of the attacker. The unobservable events, Σ_{att}, are needed to capture what the supervisor player sees since we do not have an explicit state for the supervisor in the information state. Basically, the supervisor sees the observable part of the game arena. Overall, the arena constructed thereby captures the possible attacks and all possible supervisors in a finite structure. We prove both results later on.*

Example 6.2. *We return to our illustrative intersection example to show results on the construction of the arena. In this example, we assume that the system is partially observed with $\Sigma_{obs} = \{R_{int}, R_{out}\}$. We construct $\mathcal{A}$ for system G, $X_{crit} = \{5\}$ and the all-out attack strategy for $\Sigma_{sen} = \{R_{int}\}$. Since we construct $\mathcal{A}$ for the all-out attack strategy, we can omit the attacker state. The arena has a total of* 31 *states; for this reason, we do not show the entire arena but just part of it. Figure 6.9 illustrates a part of $\mathcal{A}$ constructed with respect to G and the all-out attack strategy. We specify the missing transitions in gray. We can observe the encoding of insertion and deletion in this arena. For example, at state $\{1\}, \{R_{int}\}, \epsilon)$ the transition $ins(R_{int})$ goes to state $\{1\}, \{R_{int}\}, R_{int}$ and then transition R_{int} goes to state $\{1\}$.*

For convenience, we extend the definition of h_2 based on a given control decision. Namely, we

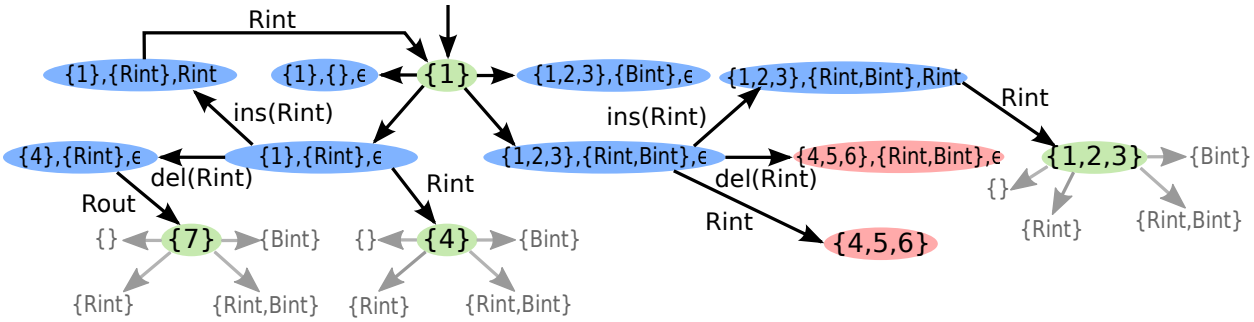


Figure 6.9: Part of $\mathcal{A}$. States in green are Q_1 states while states in blue are Q_2. Uncontrollable events are omitted from the control decisions, i.e., $\{\} := \{R_{out}, B_{out}\}$. For simplicity, transitions from Q_1 states are not labeled since the label is retrievable from the Q_2 state. States in red are critical states.

define a transition function H_2 that always starts and ends in Q_2 states. This notation simplifies walks in $\mathcal{A}$.

Definition 6.5. *We define the function* $H_2 : Q_2 \times \Sigma_{obs} \cup \Sigma_{att} \times \Gamma \to Q_2$ *as:*

$$H_2(q, e, \gamma) := \begin{cases} h_1(h_2(q, e), \gamma), & \text{if } e \in \Sigma_{obs} \\ h_2(q, e) & \text{if } e \in \Sigma_{del} \\ h_1(h_2(h_2(q, e), \mathcal{M}(e)), \gamma), & \text{if } e \in \Sigma_{ins} \\ \textit{undefined}, & \textit{otherwise} \end{cases} \tag{6.7}$$

Function H_2 *can be recursively extended for strings* $s \in (\Sigma_{obs} \cup \Sigma_{att})^*$ *given a sequence of control decisions* $\gamma_1 \dots \gamma_{|s|}$*, i.e.,*

$$H_2(q, s, \gamma_1 \dots \gamma_{|s|}) = H_2(H_2(q, s^{|s|-1}, \gamma_1 \dots \gamma_{|s|-1}), e_s^{|s|}, \gamma_{|s|}).$$

Example 6.3. *We return to our illustrative example to demonstrate function* H_2*. Let* $q = (\{1\}, \{R_{int}\}, \epsilon)$*, then*

$$H_2(q, ins(R_{int}), \{R_{int}\}) = q$$

and

$$H_2(q, ins(R_{int})del(R_{int}), \{R_{int}\}\{R_{int}\}) = (\{4\}, \{R_{int}\}, \epsilon).$$

6.4.1.2 Properties

For a fixed supervisor R and attacker A, we obtain the language $\mathcal{L}(G_a||R_a||A)$ which contains the possible string executions in the attacked system, e.g., strings of events in Σ_{all}. For convenience, let $\Sigma_{obs,att} := \Sigma_{obs} \cup \Sigma_{att}$. Given a string $s \in P_{\Sigma_{all}\Sigma_{obs,att}}(\mathcal{L}(G_a||R_a||A))$, we can find the state estimate of G_a after execution of s, i.e., the state estimate of G_a under supervision of R_a and attack strategy A. Formally, this state estimate is

$$RE(s) = \{x \in X_{G_a} \mid x = \delta_{G_a}(x_{0,G_a}, t) \text{ for } t \in P^{-1}_{\Sigma_{all}\Sigma_{obs,att}}(s) \cap \mathcal{L}(G_a||R_a||A)\} \tag{6.8}$$

In the construction of $\mathcal{A}$, we allow the attacker to insert events that are not allowed by the current control decision (see Eq. (6.5)). Therefore, given a supervisor R, we need to define its control decisions for all $s \in \Sigma^*_{obs}$, differing from the usual definition only for $s \in P_{\Sigma\Sigma_{obs}}(\mathcal{L}(G))$. For this reason, we extend the function δ_R to be a complete function in Σ_{obs}.

$$\Delta_R(x, e) = \begin{cases} \delta_R(x, e) & \text{if } e \in En_R(x) \\ x & \text{otherwise} \end{cases} \tag{6.9}$$

for $x \in R$ and $e \in \Sigma_{obs}$. Intuitively, Δ_R extends δ_R by simply ignoring the events that are not defined in δ_R. The function Δ_R is extended to $s \in \Sigma^*_{obs}$ as δ_R is extended. Lastly, we define the control decision of R for any $s \in \Sigma^*_{obs}$ as:

$$\mathcal{C}_R(s) = En_R(\Delta_R(x_{0,R}, s)) \tag{6.10}$$

Based on H_2 and $\mathcal{C}_R$, we show that the arena $\mathcal{A}$ computes the same state estimates based on the supervisor R and attacker A as the ones computed based on $G_a||R_a||A$. This result is shown in Proposition **6.6**.

Proposition 6.6. *Given a system G, a supervisor R, an attack function A and arena $\mathcal{A}$, then for*

any $s \in P_{\Sigma_{all}\Sigma_{obs,att}}(\mathcal{L}(G_a||R_a||A))$*, we have that*

$$H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})! \tag{6.11}$$

$$I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s) \tag{6.12}$$

$$I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s) \tag{6.13}$$

where $x_0 = h_1(q_0, \mathcal{C}_R(\epsilon))$ *and* $\gamma_i = \mathcal{C}_R(P^S(s^i))$.

Proof. The result is proved by induction on the string $s \in P_{\Sigma_{all}\Sigma_{o,m}}(\mathcal{L}(G_a||R_a||A))$, where $\Sigma_{o,m} = \Sigma_{obs} \cup \Sigma_{att}$.

Before we start the induction proof, we state two important results. First, we define $\mathcal{C}_{R_a}$ in the same manner as $\mathcal{C}_R$, but the control decisions of R_a are defined over Σ_{all}. It can be shown by induction that the following equality holds for any $s \in \mathcal{L}(G_a||R_a||A)$:

$$\mathcal{C}_R(P^S(s)) = \mathcal{C}_{R_a}(s) \cap \Sigma \tag{6.14}$$

Intuitively, Eq. (6.14) follows since R_a is a copy or R with insertion and deletions events added based on P^S.

Second, the function RE can be computed recursively for $s \in P_{\Sigma_{all}\Sigma_{o,m}}(\mathcal{L}(G_a||R_a||A))$ and $e \in \Sigma_{o,m}$:

$$RE(se) = \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_{R_a}(se))^*\} \tag{6.15}$$

The set Σ_{uobs} defines the unobservable events of G_a. Then, only supervisor R_a disables events in Σ_{uobs} to be executed in G_a since A is defined over $\Sigma_{o,m}$. For this reason, Eq. (6.15) is equivalent to Eq. (6.8).

Induction basis: $s = \epsilon$.

We have that $H_2(x_0, \epsilon, \epsilon) = h_1(q_0, \mathcal{C}_R(\epsilon))$ is well defined since h_1 is complete with respect to Γ. It also follows that $\delta_A(x_{0,A}, \epsilon) = I_2(x_0)$ since $\epsilon \in \mathcal{L}(G_a||R_a||A)$ and $I_2(x_0) = x_{0,A}$. We have

that $I_1(H_2(x_0, \epsilon, \epsilon)) = UR_{\mathcal{C}_R(\epsilon)}(x_{0,G})$.

$$RE(\epsilon) \overset{\text{Eq.(6.8)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(x_{0,G_a}, t) \text{ for } t \in P^{-1}_{\Sigma_{all}\Sigma_{o,m}}(\epsilon) \cap \mathcal{L}(G_a||R_a||A)\} \tag{6.16}$$

$$\overset{\text{Def.3.3}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for } t \in \Sigma^*_{uobs} \cap \mathcal{L}(G_a||R_a||A)\} \tag{6.17}$$

$$\overset{\text{Def.3.4}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_{R_a}(\epsilon))^*\} \tag{6.18}$$

$$\overset{\text{Eq.(6.14)}}{=} \{x \in X_G \mid x = \delta_G(x_{0,G}, t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_R(\epsilon))^*\} \tag{6.19}$$

$$\overset{\text{Eq.(2.4)}}{=} UR_{\mathcal{C}_R(\epsilon)}(x_{0,G}) \tag{6.20}$$

Induction hypothesis:

$H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})!$, $I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s)$ and $I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s)$ for all $s \in \mathcal{L}(G_a||R_a||A)$ and $|s| = n$.

Induction step:

Let $e \in \Sigma_{o,m}$, $s \in \mathcal{L}(G_a||R_a||A)$, $|s| = n$ and $se \in \mathcal{L}(G_a||R_a||A)$.

The induction hypothesis gives us $H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})!$, $I_1(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = RE(s)$, and $I_2(H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})) = \delta_A(x_{0,A}, s)$. Let $q = H_2(x_0, s, \gamma_1 \dots \gamma_{|s|})$.

Since $se \in \mathcal{L}(G_a||R_a||A)$ then it follows that $H_2(q, e, \gamma_{|se|})!$. Moreover, it follows that $I_2(H_2(x_0, se, \gamma_1 \dots \gamma_{|se|})) = \delta_A(x_{0,A}, se)$ by construction of $\mathcal{A}$.

For equality of Eq. (6.12), we divide the event e into three cases.

First, $e \in \Sigma_{del}$. Then, $\gamma_{|se|} = \gamma_{|s|}$. Based on the construction of $\mathcal{A}$, we have that

$I_1(H_2(q, e, \gamma_{|se|})) = UR_{\gamma_{|s|}}(NX_{P^G(e)}(I_1(q)))$.

$$RE(se) \overset{\text{Eq.(6.15)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_{R_a}(se))^*\} \tag{6.21}$$

$$\overset{\substack{\text{Def.3.3}\\ \text{Eq.(6.14)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_R(P^S(se)))^*\} \tag{6.22}$$

$$\overset{\gamma_{|s|}=\gamma_{|se|}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for } t \in (\Sigma_{uobs} \cap \gamma_{|s|})^*\} \tag{6.23}$$

$$\overset{\text{Eq.(2.4)}}{=} UR_{\gamma_{|s|}}(\delta_G(RE(s), P^G(e))) \tag{6.24}$$

$$= UR_{\gamma_{|s|}}(\delta_G(I_1(q), P^G(e))) \tag{6.25}$$

Let $e \in \Sigma_{ins}$. Based on the construction of $\mathcal{A}$, we have that $I_1(H_2(q, e, \gamma_{|se|})) = UR_{\gamma_{|se|}}(I_1(q))$.

$$RE(se) \overset{\text{Eq.(6.15)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_{R_a}(se))^*\} \tag{6.26}$$

$$\overset{\substack{\text{Def.3.3}\\ \text{Eq.(6.14)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_R(P^S(se)))^*\} \tag{6.27}$$

$$\overset{P^G(e)=\epsilon}{=} \{x \in X_G \mid x = \delta_G(RE(s), t) \text{ for } t \in (\Sigma_{uobs} \cap \gamma_{|se|})^*\} \tag{6.28}$$

$$\overset{\text{Eq.(2.4)}}{=} UR_{\gamma_{|se|}}(RE(s)) \tag{6.29}$$

$$= UR_{\gamma_{|se|}}(I_1(q)) \tag{6.30}$$

Lastly, $e \in \Sigma_{obs}$. Based on the construction of $\mathcal{A}$, we have that $I_1(H_2(q, e, \gamma_{|se|})) =$

$UR_{\gamma_{|se|}}(\delta_G(I_1(q), e))$.

$$RE(se) \overset{\text{Eq.(6.15)}}{=} \{x \in X_{G_a} \mid x = \delta_{G_a}(\delta_{G_a}(RE(s), e), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_{R_a}(se))^*\} \tag{6.31}$$

$$\overset{\substack{\text{Def.3.3} \\ \text{Eq.(6.14)}}}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), P^G(e)), t) \text{ for } t \in (\Sigma_{uobs} \cap \mathcal{C}_R(P^S(se)))^*\} \tag{6.32}$$

$$\overset{P^G(e)=e}{=} \{x \in X_G \mid x = \delta_G(\delta_G(RE(s), e), t) \text{ for } t \in (\Sigma_{uobs} \cap \gamma_{|se|})^*\} \tag{6.33}$$

$$\overset{\text{Eq.(2.4)}}{=} UR_{\gamma_{|se|}}(\delta_G(RE(s), e)) \tag{6.34}$$

$$= UR_{\gamma_{|se|}}(\delta_G(I_1(q), e)) \tag{6.35}$$

This concludes our proof. □

Recall that in the left box of Fig. 6.6 the supervisor is unknown. Equation (6.11) tells us that the arena captures all possible interactions between any supervisor R and attack function A with the plant G. It captures all possible interactions since Proposition **6.6** is true regardless of the supervisor R and of the attack function A. This is one of the main benefits of constructing the arena $\mathcal{A}$. It defines a space where all supervisors and attacker actions based on A for the plant G exist.

Moreover, Eqs. (6.12-6.13) says that the arena correctly captures the interaction between the attacker, supervisor and plant. Equation (6.12) computes the state estimate of G based on the modified string $s \in P_{\Sigma_{all}\Sigma_{obs,att}}(\mathcal{L}(G_a||R_a||A))$ and the control decisions taken by R along the observed string. These estimates capture an agent that has full knowledge of the modification on the string s and the decisions taken by R. On the other hand, Eq. (6.13) establishes the correct state of the attacker A in the construction of $\mathcal{A}$.

6.4.1.3 Solution of the meta-control problem

Our approach to solve Problem **6.1** is to consider the above-constructed arena $\mathcal{A}$ as the *uncontrolled system* in a *meta-control problem*, which is posed as a supervisory control problem for a partially-

observed discrete event system, as originally considered in [63]. For that reason, we will refer to $\mathcal{A}$ as the *meta-system*. As will become clear in the following discussion, this supervisory control approach naturally captures our synthesis objectives, and moreover supervisory control theory provides a complete characterization of the solution. Such a methodology was previously used in [30, 72] for instance; however, in these works the meta-control problem is a control problem under full observation. The same situation does not apply in our case, where events in Σ_{att} are unobservable (from the supervisor's perspective).

To formally pose the meta-control problem, we need a specification for the meta-system. In fact, the specification emerges from the corresponding specification in Problem **6.1**, which states that the controlled system should never reach any state in X_{crit}. The same specification is to be enforced in $\mathcal{A}$, where the state estimate of G represents the reachable states of G. Thus, the specification for the meta-control problem is that the meta-controlled system should never reach any state $q \in Q_1 \cup Q_2$ such that $I_1(q) \cap X_{crit} \neq \emptyset$.

The next step in the meta-control problem formulation is to specify the controllable and observable events in the meta-system $\mathcal{A}$. We already mentioned that all $e \in \Sigma_{att}$ are unobservable events. In fact, they are the only unobservable events in $\mathcal{A}$ since they are moves of the attacker that the supervisor does not directly observe. In regard to the controllable events, the supervisor makes decisions in order to react to the decisions made by the environment. Therefore, the events in $A_1 \setminus \{\Sigma_{uctr}\}$ are controllable, while those in $A_2 \cup \{\Sigma_{uctr}\}$ are uncontrollable. Note that, we explicitly exclude the control decision composed only of uncontrollable events as a meta-controllable event; the supervisor should always be able to at least enable the uncontrollable events, otherwise it would not be admissible. In this way, the supervisor can always issue at least one control decision, i.e., enable all uncontrollable plant events. We are now able to formulate the meta-control problem.

Definition 6.6. *Given $\mathcal{A}$ constructed with respect to G and A, with events $E := A_1 \cup A_2$, $E_c := A_1 \setminus \{\Sigma_{uctr}\}$ as the set of controllable events and $E_{uobs} := \Sigma_{att}$ as the set of unobservable events. Let $\mathcal{A}^{trim} := Ac(\mathcal{A}, M)$ be the specification automaton, where $M := \{q \in Q_1^{\mathcal{A}} \cup Q_2^{\mathcal{A}} \mid I_1(q) \cap$*

$X_{crit} \neq \emptyset\}$[4]. *Calculate the supremal controllable and normal sublanguage of the language of* $\mathcal{A}^{trim}$ *with respect to the language of* $\mathcal{A}$*, and let this supremal sublanguage be generated by the solution-arena denoted by* $\mathcal{A}^{\sup}$.

Note that all controllable events in the meta-control problem are also observable, i.e., $E_c \subseteq E_o$. Therefore, the controllability and observability conditions are equivalent to the controllability and normality conditions. Hence, in this case, the supremal controllable and observable sublanguage exists and is equal to the supremal controllable and normal sublanguage; see, e.g., §3.7.5 in [34]. *As consequence a supremal and unique solution of the meta-control problem exists.* This solution is the language generated by the solution-arena $\mathcal{A}^{\sup}$.

The state structure of $\mathcal{A}^{\sup}$ will depend on the algorithm used to compute the supremal controllable and normal sublanguage of $\mathcal{A}^{trim}$. One example of the structure of $\mathcal{A}^{\sup}$ is provided.

Example 6.4. *We return to our running example. Based on* $\mathcal{A}$*, we obtain* $\mathcal{A}^{\sup}$ *using an integrated (for controllability and normality) iterative algorithm to compute the supremal controllable and normal sublanguage that is based on preprocessing the input automata to satisfy simultaneously a strict sub-automaton [34] condition and a State Partition Automaton [73] condition. As part of the algorithm, one needs to refine* $\mathcal{A}$ *so that its observer is a state partition automaton (using the algorithm in [73]), i.e., to compute* $\mathcal{A}||Obs(\mathcal{A})$*, where* Obs *is the observer operation with respect to* E_{uo} *(Def. 2.5). The resulting* $\mathcal{A}^{\sup}$ *is depicted in Fig. 6.10. Each state in* $\mathcal{A}^{\sup}$ *is a tuple, where the first component is a state in* $\mathcal{A}$ *and the second component is a state in* $Obs(\mathcal{A})$*, where*

$$obs_1 := \{(\{1\}, \{R_{int}\}, \epsilon), (\{1\}, \{R_{int}\}, R_{int}), (\{4\}, \{R_{int}\}, \epsilon), (\{4\}, \{R_{int}\}, R_{int})\}$$
$$obs_2 := \{(\{7,8,9\}, \{B_{int}, R_{int}\}, \epsilon), (\{7,8,9\}, \{B_{int}, R_{int}\}, R_{int})\}$$
$$obs_3 := \{(\{7,8,9\}, \{B_{int}, R_{int}\}, \epsilon), (\{7,8,9\}, \{R_{int}\}, R_{int})\}.$$

Regardless of the algorithm to obtain $\mathcal{A}^{\sup}$, it has a structure with Q_1-like states and Q_2-like

[4]We use superscripts to differentiate the different arena structures, e.g., $\mathcal{A}$, $\mathcal{A}^{trim}$, etc.

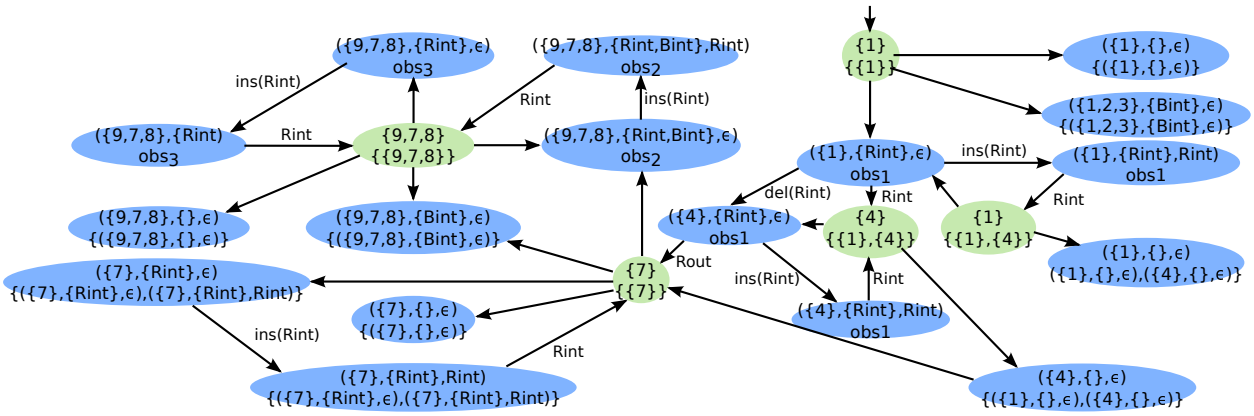


Figure 6.10: $\mathcal{A}^{\sup}$

states, since it accepts a sublanguage of $\mathcal{A}^{trim}$. Namely, it has states where only control decisions are allowed (Q_1 states) and states where only transitions with events in $\Sigma_{obs,att}$ are defined (Q_2 states). Thus, we can use the functions previously defined for $\mathcal{A}$ in $\mathcal{A}^{\sup}$.

The way $\mathcal{A}$ is constructed is such that it embeds the set of all supervisors for the original plant G. Therefore, the uniqueness of the language generated by $\mathcal{A}^{\sup}$ and the fact that it is the supremal solution of the meta-control problem means that the structure $\mathcal{A}^{\sup}$ embeds a family of supervisors S, where the controlled behavior generated by each member of that family does not reach any state in X_{crit}. Moreover, since $\mathcal{A}$ is constructed taking into account the attack function A, this family of supervisors is robust with respect to A. This leads us to the following result.

Theorem 6.3. *A supervisor R is a robust supervisor with respect to A if and only if $(\forall s \in P_{\Sigma_{all}\Sigma_{obs,att}}(\mathcal{L}(G_a||R_a||A)))[H_2^{\mathcal{A}^{\sup}}(x_0, s, \gamma_1 \dots \gamma_{|s|})!]$, where $x_0 = h_1(q_0^{\mathcal{A}^{\sup}}, \mathcal{C}_R(\epsilon))$ and $\gamma_i = \mathcal{C}_R(P^S(s^i))$.*

Proof. We start with the only if part. Let R be a robust supervisor. Proposition **6.6** guarantees that $H_2^{\mathcal{A}}(x_0, s, \gamma_1 \dots \gamma_{|s|})$ is defined for any $s \in \mathcal{L}(G_a||R_a||A)$ and for any attack function representation A. To analyze the meta-system $\mathcal{A}$, we have to define some notation for it.

We are analyzing $\mathcal{A}$ as a meta-system, namely as an automaton. The function h is a the combination of the functions h_1 and h_2. Let $E = A_1 \cup A_2$ be the event set of $\mathcal{A}$, $E_o = E \setminus \Sigma_{att}$ the observable event set, $E_c = A_1 \setminus \{\Sigma_{uctr}\}$ the controllable event set. Moreover, the function

$\eta : E^* \to \Sigma_{obs}^*$ projects strings in E^* to strings in Σ_{obs}^*. Intuitively, for any $s \in \mathcal{L}(\mathcal{A})$ the function $\eta(s)$ returns the string that is observed by the supervisor.

We construct the language $L \subset \mathcal{L}(\mathcal{A})$ recursively as:

1. $\epsilon \in L$
2. $s \in L \wedge h(q_0, s) \in Q_2^{\mathcal{A}} \Rightarrow se \in L, \forall e \in \Gamma_{\mathcal{A}}(h(q_0, s))$
3. $s \in L \wedge h(q_0, s) \in Q_1^{\mathcal{A}} \wedge \big(e = \{\Sigma_{uctr}\} \vee e = \mathcal{C}_R(\eta(s))\big) \Rightarrow se \in L$

The language L is by construction controllable w.r.t. E_c and $\mathcal{L}(\mathcal{A})$; we show that L is normal w.r.t. E_o and $\mathcal{L}(\mathcal{A})$. The result is shown by contradiction. Assume that L is not normal, then there exists a shortest $s \in L$ and $t \in \mathcal{L}(\mathcal{A}) \setminus L$ s.t. $P_{EE_o}(s) = P_{EE_o}(t)$. In the construction of L, player 2 is not constrained, meaning that the shortest strings that belong to $\mathcal{L}(\mathcal{A}) \setminus L$ end with an event in A_1 (control decisions). For this reason, $e_s^{|s|} = e_t^{|t|}$ and $P_{EE_o}(s^{|s|-1}) = P_{EE_o}(t^{|t|-1})$. It implies that $\eta(s^{|s|-1}) = \eta(t^{|t|-1})$ and $\mathcal{C}_R(\eta(s^{|s|-1})) = \mathcal{C}_R(\eta(t^{|t|-1}))$. By the definition of L, $t \in L$. This contradicts our assumption.

It is also true that $L \subseteq \mathcal{L}(\mathcal{A}^{trim})$, otherwise R would not be a robust supervisor. Intuitively, the actions made by player 2 are not constrained in the construction of L. This guarantees that L embeds all actions of attacker A. The actions of player 1 are constrained based on R and $\{\Sigma_{uctr}\}$. R is robust and changing any of its control actions for any string by $\{\Sigma_{uctr}\}$ will preserve robustness since $\{\Sigma_{uctr}\} \subseteq \gamma$ for any $\gamma \in \Gamma$.

Definition **6.6** defines $\mathcal{L}(\mathcal{A}^{\sup})$ to be the supremal controllable and normal sublanguage of $\mathcal{L}(\mathcal{A}^{trim})$ w.r.t. E_c, E_o and $\mathcal{L}(\mathcal{A})$. Since $L \subseteq \mathcal{L}(\mathcal{A}^{trim})$ and it is controllable and normal, then $L \subseteq \mathcal{L}(\mathcal{A}^{\sup})$. Therefore, $H_2^{\mathcal{A}^{\sup}}(x_0, s, \gamma_1 \dots \gamma_{|s|})!$ holds for all $s \in P_{\Sigma_m \Sigma_{o,e}}(\mathcal{L}(G_a || R_a || A))$.

For the if part, the result follows from the construction of $\mathcal{A}$, $\mathcal{A}^{trim}$ and the properties of $\mathcal{A}^{\sup}$. □

Corollary 6.2. *$\mathcal{A}^{\sup} = \emptyset$ if and only if there does not exist any robust supervisor R with respect to attacker A.*

Corollary **6.2** gives a necessary and sufficient condition for the existence of a solution for Problem **6.1**. Given that there exists a robust supervisor, we provide an algorithm[5] to extract a supervisor that solves Problem **6.1**. First, we define function H_1 as we defined H_2.

Definition 6.7. *Let the function* $H_1 : Q_1 \times \Sigma_{obs,att} \times \Gamma \to Q_1$ *be defined as:*

$$H_1(q,e,\gamma) := \begin{cases} h_2(h_1(q,\gamma),e), & \textit{if } e \in \Sigma_{obs} \\ q & \textit{if } e \in \Sigma_{del} \\ h_2(h_2(h_1(q,\gamma),e),\mathcal{M}(e)), & \textit{if } e \in \Sigma_{ins} \\ \textit{undefined,} & \textit{otherwise} \end{cases} \tag{6.36}$$

Algorithm 7 Robust Supervisor Extraction

Require: $\mathcal{A}^{\sup}$
Ensure: $R_r = (X_{R_r}, \Sigma, \delta_{R_r}, x_{0,R_r})$
1: $x_{0,R_r} = q_0^{\mathcal{A}^{\sup}}$
2: $X_{R_r} \leftarrow \{x_{0,R_r}\}$, $\delta_{R_r} \leftarrow \emptyset$
3: Expand(x_{0,R_r})
4: **procedure** EXPAND(x)
5: select $\gamma \in \Gamma_{\mathcal{A}^{\sup}}(x)$ s.t. $\forall \gamma' \in \Gamma_{\mathcal{A}^{\sup}}(x) : \gamma \not\subset \gamma'$
6: **for all** $e \in \Sigma \cap \gamma$ **do**
7: **if** $e \in \Sigma_{obs}$ **then**
8: $y = H_1^{\mathcal{A}^{\sup}}(x,e,\gamma)$, $\delta_{R_r} \leftarrow \delta_{R_r} \cup (x,e,y)$
9: $X_{R_r} \leftarrow X_{R_r} \cup \{y\}$
10: **if** $y \notin X_{R_r}$ **then**
11: $Expand(y)$
12: **else**
13: $\delta_{R_r} \leftarrow \delta_{R_r} \cup (x,e,x)$

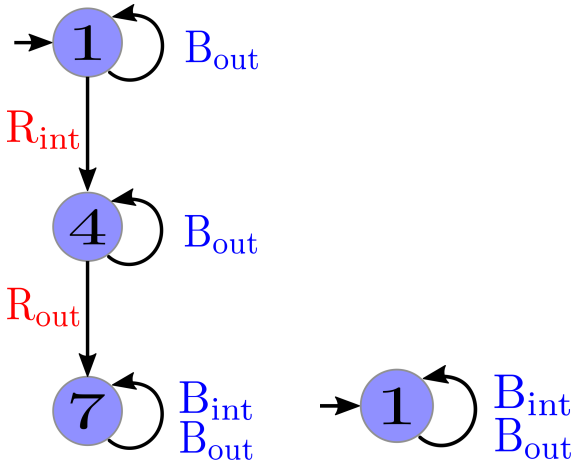
Algorithm 7 starts at the initial state of $\mathcal{A}^{\sup}$ and performs a Depth First Search by selecting only the largest control decisions at each state that it visits. By largest, we mean that it selects a control decision that is not a subset of any other control decision defined at state x, as described by line 5. Note that, it is possible to have more than two decisions that satisfy this condition. In this case, the algorithm selects one of the possible decisions in a nondeterministic manner. The

[5]There are different manners for a designer to extract a robust supervisor.

algorithm terminates since $\mathcal{A}^{\text{sup}}$ is finite. Moreover, the algorithm only traverses player 1 states, where the control decisions are defined.

Corollary 6.3. *A supervisor R_r constructed by Algorithm 7 is a solution for Problem **6.1**.*

Example 6.5. *To conclude this section, we provide two supervisors extracted via Algorithm 7. These two supervisors are depicted in Fig. 6.11.*



(a) Robust supervisor 1 (b) Robust supervisor 2

Figure 6.11: Robust supervisors with respect to all-out attack strategy

6.4.2 Selecting supervisors in the robust arena

Algorithm 7 provides one way of extracting robust supervisors from $\mathcal{A}^{\text{sup}}$. As we explained before, it selects maximal control decisions in the Q_1 states that the algorithm visits. Example **6.5** shows that this extraction does not provide specific information about the language generated by supervised system once a supervisor is selected, other than the fact that we are choosing a locally maximal control decision. While supervisor R_1 in Fig. 6.11(a) generates a live language, supervisor R_2 in Fig. 6.11(b) is blocking. Nonetheless, the space defined in $\mathcal{A}^{\text{sup}}$ provides maximum flexibility in extracting different supervisors since all robust supervisors are embedded in $\mathcal{A}^{\text{sup}}$. The methods in [14, 13, 28] do not provide the flexibility of $\mathcal{A}^{\text{sup}}$ since they exploit algorithms of supervisory control theory where only *one* supervisor can be obtained at a time. In fact, when

explicit comparisons can be made, the supervisors obtained by their methods are embedded in the corresponding $\mathcal{A}^{\text{sup}}$.

Another benefit of the construction of $\mathcal{A}^{\text{sup}}$ is the ability to exploit results in the area of turn-based two-player graph-games. Results from these areas can be leveraged to study different manners of extracting robust supervisors, for example, to study quantitative versions of the robust supervisor problem under some cost model [41, 74, 31].

We provide an example of a supervisor extraction algorithm based on a quantitative measure. First, we define a measure over the supervised system R/G, i.e., over the states of the automaton $G||R$. Let the set $X_{dead} = \{x \in X_{G||R} \mid \Gamma_{G||R}(\delta_{G||R}(x, s)) = \emptyset \text{ for } s \in \Sigma^*_{uobs}\}$ be the set of states in $G||R$ that can reach a deadlock state via an unobservable string. We define $r : X_{G||R} \to [0, +\infty) \cup \{-\infty\}$ to be a reward function for any $(x, y) \in X_{G||R}$ as:

$$r((x,y)) = \begin{cases} -\infty & \text{if } x \in X_{crit} \\ 0 & \text{if } (x,y) \in X_{dead} \\ c & \text{otherwise} \end{cases} \tag{6.37}$$

where $c \in [0, \infty)$.

The reward function r punishes states from where the system $G||R$ might deadlock. Based on the reward function r, we define the following total reward for the supervised system $G||R$.

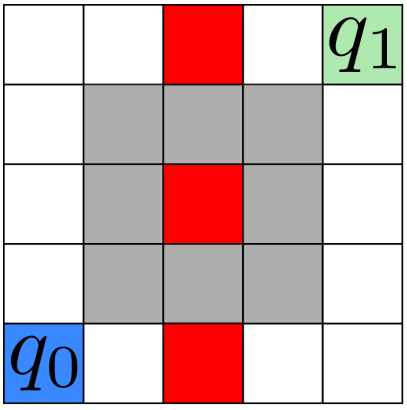
$$Reward(R, G) = \sum_{x \in X_{G||R}} r(x) \tag{6.38}$$

We can generalize Algorithm 7 to incorporate this quantitative measure such that it extracts a supervisor from $\mathcal{A}^{\text{sup}}$ that maximizes the measure $Reward(R, G)$. In our running example, this new method would extract robust supervisor 1. We can even go further in this extraction method by assuming that the attacker tries to minimize $Reward(R, G)$. In this scenario, we would pose a $\min\max$ problem in order to select a supervisor from $\mathcal{A}^{\text{sup}}$. We leave these extensions for future work.

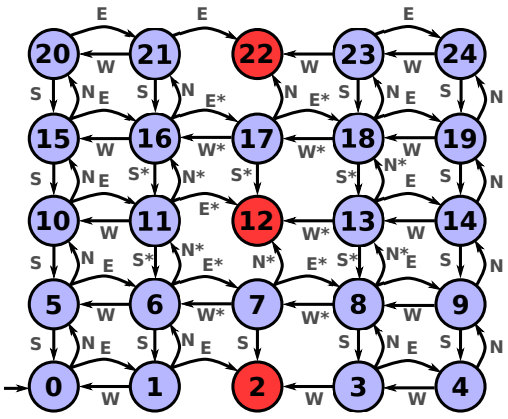
6.4.3 Robot motion planning example

We developed a tool[6] to automatically construct $\mathcal{A}$, as in Definition **6.4**, and to compute $\mathcal{A}^{\text{sup}}$. Moreover, Algorithm 7 is also implemented in our tool. Our evaluation was done on a Linux machine with 2.2GHz CPU and 16GB memory.

We consider a robot moving in a possibly hostile environment. The robot is assumed to have four different movement modes: North, South, East, West.The robot moves freely in the workspace shown in Fig. 6.12(a). Its initial state is the blue cell denoted as q_0 and the red cells are considered to be obstacles. Moreover, the shaded region is assumed to be hostile and the sensor readings of the robot could be under attack. We differentiate the sensor reading in the shaded area by marking them with an *, i.e., $\Sigma = \Sigma_{ctr} = \Sigma_{obs} = \{N, S, E, W, N^*, S^*, E^*, W^*\}$. This uncontrolled system is modeled by the automaton depicted in Fig. 6.12(b). We want to design a robust supervisor that enforces the following properties: (1) the robot must avoid the obstacles; (2) the robot can always access states q_0 and q_1.



(a) Robot workspace: the robot starts in the blue cell denoted by q_0; the shaded area is considered hostile and the sensor readings in that area are compromised.



(b) Model of the robot in the workspace: $\Sigma = \Sigma_{ctr} = \Sigma_{obs} = \{E, W, N, S, E*, W*, N*, S*\}$

Figure 6.12: Robot workspace and attack simulation

The set of compromised events is $\Sigma_{sen} = \{E^*, W^*, N^*, S^*\}$ since we consider that the sensor readings in the shaded area might be under attack. First, we construct the arena $\mathcal{A}$ considering the

[6] https://gitlab.eecs.umich.edu/M-DES-tools/desops

all-out attack strategy. The number of states in $\mathcal{A}$ is 18649 states. The state space explosion is due to the number of control decisions: there are 256 possible control decisions.

After constructing $\mathcal{A}$, we obtain $\mathcal{A}^{\text{sup}}$ as described in Definition **6.6**. To compute the supremal controllable and normal sublanguage, we use the algorithm described in Example **6.4**; it follows that $\mathcal{A}^{\text{sup}}$ has 65358 states. Note that $\mathcal{A}^{\text{sup}}$ has more states than $\mathcal{A}$ due to necessary preprocessing done by the iterative algorithm for the computation of the supremal controllable and normal sublaguage.

Finally, any supervisor selected from $\mathcal{A}^{\text{sup}}$ is robust against the all-out attack strategy, i.e., it satisfies property (1). Therefore, we must select a supervisor that satisfies property (2). For this reason, we modify Algorithm 7 to extract a supervisor from $\mathcal{A}^{\text{sup}}$ such that property (2) is satisfied. This supervisor is depicted in Fig. 6.13.

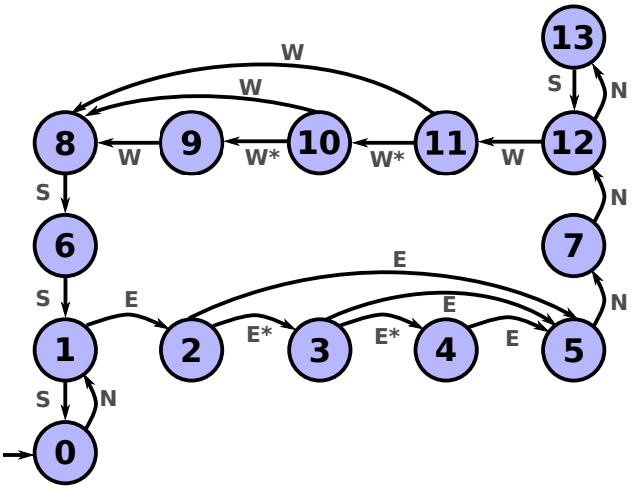


Figure 6.13: Robust supervisor for robot in a hostile environment

6.4.4 Summary

By formulating the problem at the supervisory control layer of a cyber-physical system, we have been able to leverage techniques from games on automata under partial information and from supervisory control of partially-observed discrete event systems to develop a solution methodology to prevent damage to the system when some sensor readings may be edited by the attacker. The

space defined in $\mathcal{A}^{\text{sup}}$ provides maximum flexibility in extracting different supervisors since all robust supervisors are embedded in $\mathcal{A}^{\text{sup}}$. As discussed in section 6.4.2, it would be interesting to investigate methods to extract supervisors from $\mathcal{A}^{\text{sup}}$ in order to satisfy additional constraints, such as optimality with respect to some quantitative criterion [74, 41]. Finally, identifying ways to reduce the state space of the arena by exploiting a suitable notion of state equivalence is another important research direction.

6.5 Conclusion

We considered a class of problems in cyber-security where sensor readings in a feedback control system may be manipulated by a malicious attacker. By formulating the problem at the supervisory control layer of a cyber-physical system, we have been able to leverage techniques from games on automata under partial information and from supervisory control of partially-observed discrete event systems to develop a solution methodology to prevent damage to the system when some sensor readings may be edited by the attacker. Our problem formulation is parameterized by an attack strategy over a set of compromised events. In this manner, synthesis of robust supervisors against sensor deception attack strategies is considered, e.g., bounded attack strategies, replacement attack strategies, etc. Moreover, if there is no prior information about the attacker strategy, then we consider the general all-out attack strategy over the set of compromised events. A supervisor robust against the all-out attack strategy is robust against any other sensor deception attack strategy over the same set of compromised events.

We presented two sound and complete solutions to the investigated problem, where one is more scalable than the other. The space defined in $\mathcal{A}^{\text{sup}}$ provides maximum flexibility in extracting different supervisors since all robust supervisors are embedded in $\mathcal{A}^{\text{sup}}$. On the other hand, the output of Algorithm 6 provides us with one maximal robust supervisor solution.

CHAPTER VII
Conclusion

7.1 Contribution

In this dissertation, we studied how disruptive and dangerous sensor deception attacks at the supervisory control layer of cyber-physical systems can be. Moreover, we investigated how to shield these systems against sensor deception attacks. Given that we studied these problems at the supervisory control layer of cyber-physical systems, our work leveraged the field of supervisory control theory of partially observed discrete event systems. In fact, we enhanced the supervisory control framework such that it takes into account sensor deception attacks.

Sensor deception attacks in the supervisory control framework were modeled as *attack functions*, i.e., the attacker can modify/edit events generated by the plant (physical process) before they reach the supervisor. In this manner, the supervisor might take control decisions based on incorrect information. We formally defined this enhanced supervisory control framework and completely characterized the closed-loop behavior of the controlled system under sensor deception attacks.

Once we characterized the supervisory control framework under sensor deception attacks, we focused on two important components of this framework: the attacker and the supervisor. Intuitively, we concentrated on two questions: "Can we find attack strategies that cause damage to the controlled system without raising any suspicion?"; "Can we find a supervisor that does not allow any attacker to cause damage to the controlled system?".

First, we focused on the attacker by putting ourselves into the attacker's shoes. In this context, we formulated the problem of synthesizing stealthy sensor deception attacks, i.e., an attack that it causes damage to the system without detection by an existing supervisor. Our solution procedure

is game-based and relies on the construction of a discrete structure called the AIDA, which is used to solve the synthesis problem for different attack scenarios.

The problem of synthesizing stealthy sensor deception attacks was posed in a qualitative manner, i.e., it is either possible to synthesize a stealthy attack or it is not. Next, we considered the problem of synthesis of attack functions for sensor deception attacks at the supervisory layer of feedback control systems, where the system is modeled as a probabilistic finite-state automaton controlled by a given deterministic supervisor. We investigated two problems: the problem of synthesizing attack functions with the maximum likelihood of reaching the critical state and a second problem where each attack edit is costly. In the second problem, the attacker has two objectives: (i) reach the critical state with the maximum likelihood; (ii) minimize the expected cost while performing (i). In order to solve these problems, we leveraged techniques from MDPs. Namely, we reduced these problem to two well known problems in MDPs: the probabilistic reachability problem and the stochastic shortest path problem.

Finally, we moved on to address the second question: "Can we find a supervisor that does not allow any attacker to cause damage to the controlled system?". We investigated the problem of synthesizing robust supervisors against sensor deception attack strategies. Our problem formulation is parameterized by an attacker strategy over a set of compromised events. Based on this given attack strategy, we synthesize a robust supervisor against it. Therefore, our robustness criterion is defined based on the type of attack strategy, e.g., bounded attack strategy, replacement attack strategy, etc. Nonetheless, we provide the all-out attack strategy that includes any other attack strategy over the same set of compromised events. Robustness against the all-out attack strategy implies robustness against any other attack strategy. Finally, we presented two sound and complete solutions to the investigated problem, where one is more scalable than the other.

In summary, this dissertation introduced a novel supervisory control theory framework that addressed the problem of sensor deception attacks in controlled systems. Moreover, it investigated problems from the attacker's perspective as well as the supervisor's perspective. On the attacker's perspective, we provided both qualitative and quantitative methods to synthesize attack strategies.

On the other hand, only qualitative results were obtained for the synthesis of robust supervisors.

7.2 Future work

There are several potential research directions for future work. Here we mention a few of these potential research directions.

This dissertation only investigates sensor deception attacks. Actuator deception attacks are also an important class of attacks that must be investigated. Recently, actuator deception attacks were investigated in isolation, similarly as our work [16, 18]. A potential area of research is to study the supervisory control framework under both sensor and actuator deception attacks.

In Chapters IV and V, we assume that the attacker has full knowledge of the controlled system (Assumption **4.1**). Relaxing this assumption is a possible extension of our work. For example, assuming that the attacker has partial information about the controlled system. The attacker has to resolve two issues in order to synthesize a successful attack strategy. It needs to learn the controlled system while it tries to synthesize an attack strategy. Therefore, this potential area would require both synthesis and learning techniques. This combination of learning and synthesis techniques would fit nicely in the framework developed in Chapter V. Reinforcement learning methods could provide the means to investigate this problem.

With respect to the defense techniques explored in Chapter VI, we provide one way of robustifying the controlled system. The synthesized controlled system is robust by limiting its uncontrolled behavior considering sensor deception attacks. Is it possible to obtain a robust controlled system that is more permissive than the one obtained by our methods in Chapter VI? One possible solution is to assume an active defense mechanism, i.e., a controlled system that actively prevents sensor deception attacks. This notion of active defense mechanism is exploited in problems of opacity enforcement [75, 24]. Enhancing the system of interest with a mechanism that actively modifies its observational behavior provides leverage enforcing opacity with permissiveness. Another possible technique is the Moving Target Defense paradigm [76], where the supervisor ac-

tively changes its actions in order to create uncertainty for attackers [77]. Using such defense mechanisms would allow a more permissive controlled system.

Lastly, this dissertation focused on the important class of cyber-physical systems. However, investigating these techniques and problems in cyber systems is also of critical importance. Network protocols are the essence of cyber systems since these systems are a collection of computerized networks. These protocols must be resilient against both malicious planned attacks and benign malfunction [78]. A model-based approach to study the security aspects of these protocols is the first step in understanding their vulnerabilities against attacks. This model-based approach would integrate modeling techniques from this dissertation to investigate security problems in cyber systems.

BIBLIOGRAPHY

[1] Farwell, J. P. and Rohozinski, R., "Stuxnet and the Future of Cyber War," *Survival*, Vol. 53, No. 1, 2011, pp. 23–40.

[2] Slay, J. and Miller, M., "Lessons Learned from the Maroochy Water Breach," *Critical Infrastructure Protection*, edited by E. Goetz and S. Shenoi, Springer US, Boston, MA, 2008, pp. 73–82.

[3] Kshetri, N. and Voas, J., "Hacking Power Grids: A Current Problem," *Computer*, Vol. 50, No. 12, December 2017, pp. 91–95.

[4] Greenberg, A., "The Jeep Hackers are back to prove car hacking can get much worse," *Wired*, 2016.

[5] Council of Economic Advisers, "The Cost of Malicious Cyber Activity to the U.S. Economy," *Executive Office of the President of the United States*, Feb 2018.

[6] Rashidinejad, A., Wetzels, B., Reniers, M., Lin, L., Zhu, Y., and Su, R., "Supervisory Control of Discrete-Event Systems under Attacks: An Overview and Outlook," *2019 18th European Control Conference (ECC)*, June 2019, pp. 1732–1739.

[7] Cardenas, A. A., Amin, S., and Sastry, S., "Secure Control: Towards Survivable Cyber-Physical Systems," *2008 The 28th International Conference on Distributed Computing Systems Workshops*, June 2008, pp. 495–500.

[8] Teixeira, A., Pérez, D., Sandberg, H., and Johansson, K. H., "Attack Models and Scenarios for Networked Control Systems," *Proceedings of the 1st International Conference on High Confidence Networked Systems*, HiCoNS '12, ACM, New York, NY, USA, 2012, pp. 55–64.

[9] Carvalho, L. K., Wu, Y. C., Kwong, R., and Lafortune, S., "Detection and prevention of actuator enablement attacks in supervisory control systems," *13th International Workshop on Discrete Event Systems*, May 2016, pp. 298–305.

[10] Lima, P. M., Carvalho, L. K., and Moreira, M. V., "Detectable and Undetectable Network Attack Security of Cyber-physical Systems," *14th IFAC Workshop on Discrete Event Systems WODES 2018*, Vol. 51, 2018, pp. 179 – 185.

[11] Carvalho, L. K., Wu, Y. C., Kwong, R., and Lafortune, S., "Detection and mitigation of classes of attacks in supervisory control systems," *Automatica*, Vol. 97, 2018, pp. 121 – 133.

[12] Lima, P. M., Alves, M. V. S., Carvalho, L. K., and Moreira, M. V., "Security Against Communication Network Attacks of Cyber-Physical Systems," *Journal of Control, Automation and Electrical Systems*, Vol. 30, No. 1, Feb 2019, pp. 125–135.

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

[14] Wakaiki, M., Tabuada, P., and Hespanha, J. P., "Supervisory Control of Discrete-Event Systems Under Attacks," *Dynamic Games and Applications*, Sep 2018.

[15] Wang, Y. and Pajic, M., "Supervisory Control of Discrete Event Systems in the Presence of Sensor and Actuator Attacks," *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 5350–5355.

[16] Lin, L., Thuijsman, S., Zhu, Y., Ware, S., Su, R., and Reniers, M., "Synthesis of Supremal Successful Normal Actuator Attackers on Normal Supervisors," *2019 American Control Conference (ACC)*, 2019, pp. 5614–5619.

[17] Zhu, Y., Lin, L., and Su, R., "Supervisor Obfuscation Against Actuator Enablement Attack," *2019 18th European Control Conference (ECC)*, 2019, pp. 1760–1765.

[18] Lin, L., Zhu, Y., and Su, R., "Towards Bounded Synthesis of Resilient Supervisors," *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 7659–7664.

[19] Wang, Z., Meira-Góes, R., Lafortune, S., and Kwong, R., "Mitigation of Classes of Attacks using a Probabilistic Discrete Event System Framework," *15th IFAC Workshop on Discrete Event Systems WODES 2020*, 2020.

[20] Thorsley, D. and Teneketzis, D., "Intrusion Detection in Controlled Discrete Event Systems," *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 6047–6054.

[21] Paoli, A., Sartini, M., and Lafortune, S., "Active Fault Tolerant Control of Discrete Event Systems Using Online Diagnostics," *Automatica*, Vol. 47, No. 4, April 2011, pp. 639–649.

[22] Meira-Góes, R., Keroglou, C., and Lafortune, S., "Towards probabilistic intrusion detection in supervisory control of discrete event systems," *21st IFAC World Congress (to appear)*, 2020.

[23] Zhang, Q., Li, Z., Seatzu, C., and Giua, A., "Stealthy Attacks for Partially-Observed Discrete Event Systems," *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1, 2018, pp. 1161–1164.

[24] Wu, Y.-C., Raman, V., Rawlings, B. C., Lafortune, S., and Seshia, S. A., "Synthesis of Obfuscation Policies to Ensure Privacy and Utility," *Journal of Automated Reasoning*, Vol. 60, No. 1, Jan 2018, pp. 107–131.

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

[26] Meira-Góes, R., Kang, E., Kwong, R. H., and Lafortune, S., "Synthesis of sensor deception attacks at the supervisory layer of CyberPhysical Systems," *Automatica*, Vol. 121, 2020, pp. 109172.

[27] Meira-Góes, R., Lafortune, S., and Marchand, H., "Synthesis of Supervisors Robust Against Sensor Deception Attacks," *under review*, 2019.

[28] Meira-Góes, R., Marchand, H., and Lafortune, S., "Towards resilient supervisors against sensor deception attacks," *2019 IEEE 58th Annual Conference on Decision and Control (CDC)*, Dec. 2019.

[29] Yin, X. and Lafortune, S., "Synthesis of Maximally-Permissive Supervisors for the Range Control Problem," *IEEE Transactions on Automatic Control*, Vol. 62, No. 8, 2017, pp. 3914–3929.

[30] Yin, X. and Lafortune, S., "A Uniform Approach for Synthesizing Property-Enforcing Supervisors for Partially-Observed Discrete-Event Systems," *IEEE Transactions on Automatic Control*, Vol. 61, No. 8, Aug 2016, pp. 2140–2154.

[31] Meira-Góes, R., Kwong, R., and Lafortune, S., "Synthesis of Sensor Deception Attacks for Systems Modeled as Probabilistic Automata," *2019 American Control Conference (ACC)*, July 2019.

[32] Meira-Góes, R., Kwong, R., and Lafortune, S., "Synthesis of optimal multi-objective attack strategies for controlled systems modeled by probabilistic automata," *under review*, 2020.

[33] Hopcroft, J. E., Motwani, R., and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

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

[35] Ramadge, P. J. and Wonham, W. M., "Supervisory Control of a Class of Discrete Event Processes," *SIAM J. Control Optim.*, Vol. 25, No. 1, Jan. 1987, pp. 206–230.

[36] Alves, M. V. S., Basilio, J. C., da Cunha, A. E. C., Carvalho, L. K., and Moreira, M. V., "Robust Supervisory Control Against Intermittent Loss of Observations," *IFAC Proceedings Volumes*, Vol. 47, No. 2, 2014, pp. 294 – 299.

[37] Lin, F., "Control of Networked Discrete Event Systems: Dealing with Communication Delays and Losses," *SIAM Journal on Control and Optimization*, Vol. 52, No. 2, 2014, pp. 1276–1298.

[38] Rohloff, K., "Bounded Sensor Failure Tolerant Supervisory Control," *IFAC Proceedings Volumes*, Vol. 45, No. 29, 2012, pp. 272 – 277.

[39] Xu, S. and Kumar, R., "Discrete event control under nondeterministic partial observation," *2009 IEEE International Conference on Automation Science and Engineering*, Aug 2009, pp. 127–132.

[40] Yin, X., "Supervisor Synthesis for Mealy Automata with Output Functions: A Model Transformation Approach," *IEEE Transactions on Automatic Control*, Vol. PP, No. 99, 2016.

[41] Cassez, F., Dubreil, J., and Marchand, H., "Synthesis of opaque systems with static and dynamic masks," *Formal Methods in System Design*, Vol. 40, No. 1, 2012, pp. 88–115.

[42] Lin, F., "Opacity of Discrete Event Systems and Its Applications," *Automatica*, Vol. 47, No. 3, March 2011, pp. 496–503.

[43] Saboori, A. and Hadjicostis, C. N., "Notions of security and opacity in discrete event systems," *46th IEEE Conference on Decision and Control*, Dec 2007, pp. 5056–5061.

[44] Meira-Góes, R., Rawlings, B. C., Recker, N., Willett, G., and Lafortune, S., "Demonstration of Indoor Location Privacy Enforcement using Obfuscation," *14th IFAC Workshop on Discrete Event Systems WODES 2018*, Vol. 51, No. 7, 2018, pp. 145 – 151.

[45] Grädel, E., Thomas, W., and Wilke, T., editors, *Automata Logics, and Infinite Games: A Guide to Current Research*, Springer-Verlag, Berlin, Heidelberg, 2002.

[46] Paoli, A. and Lafortune, S., "Safe Diagnosability for Fault-tolerant Supervision of Discrete-event Systems," *Automatica*, Vol. 41, No. 8, Aug. 2005, pp. 1335–1347.

[47] Golaszewski, C. H. and Ramadge, P. J., "Control of discrete event processes with forced events," *26th IEEE Conference on Decision and Control*, Vol. 26, Dec 1987, pp. 247–251.

[48] de Alfaro, L., "Computing Minimum and Maximum Reachability Times in Probabilistic Systems," *CONCUR'99 Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 66–81.

[49] Bertsekas, D. P., *Dynamic Programming and Optimal Control, Vol. II*, Athena Scientific, 3rd ed., 2007.

[50] Kumar, R. and Garg, V. K., "Control of stochastic discrete event systems modeled by probabilistic languages," *IEEE Transactions on Automatic Control*, Vol. 46, No. 4, Apr 2001, pp. 593–606.

[51] Garg, V. K., Kumar, R., and Marcus, S. I., "A probabilistic language formalism for stochastic discrete-event systems," *IEEE Transactions on Automatic Control*, Vol. 44, No. 2, Feb 1999, pp. 280–293.

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

[53] Pantelic, V., Postma, S. M., and Lawford, M., "Probabilistic Supervisory Control of Probabilistic Discrete Event Systems," *IEEE Transactions on Automatic Control*, Vol. 54, No. 8, 2009, pp. 2013–2018.

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

[55] Kushner, H., *Introduction to Stochastic Control*, Holt, Rinehart and Winston, 1971.

[56] Kolobov, A., Mausam, and Weld, D. S., "A Theory of Goal-oriented MDPs with Dead Ends," *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI'12, 2012, pp. 438–447.

[57] Lacerda, B., Parker, D., and Hawes, N., "Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications," *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI15, AAAI Press, 2015, p. 15871593.

[58] Kwiatkowska, M., Norman, G., and Parker, D., "PRISM 4.0: Verification of Probabilistic Real-Time Systems," *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 585–591.

[59] Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., and Qu, H., "Quantitative Multi-objective Verification for Probabilistic Systems," *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 112–127.

[60] Liu, J., Ozay, N., Topcu, U., and Murray, R. M., "Synthesis of Reactive Switching Protocols From Temporal Logic Specifications," *IEEE Transactions on Automatic Control*, Vol. 58, No. 7, July 2013, pp. 1771–1785.

[61] Li, Y., Lin, F., and Lin, Z. H., "A generalized framework for supervisory control of discrete event systems," *International Journal of Intelligent Control and Systems*, Vol. 2, 1998, pp. 139–160.

[62] Takai, S., "Supervisory control of partially observed discrete event systems with arbitrary control patterns," *International Journal of Systems Science*, Vol. 31, No. 5, 2000, pp. 649–656.

[63] Lin, F. and Wonham, W., "On observability of discrete-event systems," *Information Sciences*, Vol. 44, No. 3, 1988, pp. 173 – 198.

[64] Cieslak, R., Desclaux, C., Fawaz, A. S., and Varaiya, P., "Supervisory control of discrete-event processes with partial observations," *IEEE Transactions on Automatic Control*, Vol. 33, No. 3, March 1988, pp. 249–260.

[65] Cho, H. and Marcus, S. I., "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation," *Mathematics of Control, Signals and Systems*, Vol. 2, No. 1, 1989, pp. 47–69.

[66] Lin, F., "Robust and adaptive supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, Vol. 38, No. 12, Dec 1993, pp. 1848–1852.

[67] Cury, J. and Krogh, B., "Robustness of supervisors for discrete-event systems," *IEEE Transactions on Automatic Control*, Vol. 44, No. 2, 1999, pp. 376–379.

[68] Takai, S., "Maximizing robustness of supervisors for partially observed discrete event systems," *Automatica*, Vol. 40, No. 3, 2004, pp. 531 – 535.

[69] Pasqualetti, F., Dorfler, F., and Bullo, F., "Control-Theoretic Methods for Cyberphysical Security: Geometric Principles for Optimal Cross-Layer Resilient Control Systems," *IEEE Control Systems Magazine*, Vol. 35, No. 1, Feb 2015, pp. 110–127.

[70] Chong, M. S., Sandberg, H., and Teixeira, A. M. H., "A Tutorial Introduction to Security and Privacy for Cyber-Physical Systems," *2019 18th European Control Conference (ECC)*, June 2019, pp. 968–978.

[71] Hadj-Alouane, N. B., Lafortune, S., and Lin, F., "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *Discrete Event Dynamic Systems*, Vol. 6, No. 4, Oct 1996, pp. 379–427.

[72] Wu, Y. C. and Lafortune, S., "Synthesis of Optimal Insertion Functions for Opacity Enforcement," *IEEE Transactions on Automatic Control*, Vol. 61, No. 3, March 2016, pp. 571–584.

[73] Jirásková, G. and Masopust, T., "On Properties and State Complexity of Deterministic State-Partition Automata," *Theoretical Computer Science*, edited by J. C. M. Baeten, T. Ball, and F. S. de Boer, Berlin, Heidelberg, 2012, pp. 164–178.

[74] Ji, Y., Yin, X., and Lafortune, S., "Mean Payoff Supervisory Control Under Partial Observation," *2018 IEEE Conference on Decision and Control (CDC)*, Dec 2018, pp. 3981–3987.

[75] Wu, Y.-C., Raman, V., Lafortune, S., and Seshia, S. A., "Obfuscator Synthesis for Privacy and Utility," *Proceedings of the 8th International Symposium on NASA Formal Methods - Volume 9690*, NFM 2016, Springer-Verlag New York, Inc., New York, NY, USA, 2016, pp. 133–149.

[76] Jajodia, S., Ghosh, A. K., Swarup, V., Wang, C., and Wang, X. S., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Springer Publishing Company, Incorporated, 1st ed., 2011.

[77] Meira-Góes, R. and Lafortune, S., "Moving Target Defense based on Switched Supervisory Control: A New Technique for Mitigating Sensor Deception Attacks," *15th IFAC Workshop on Discrete Event Systems WODES 2020*, 2020.

[78] von Hippel, M., Vick, C., Tripakis, S., and Nita-Rotaru, C., "Automated Attacker Synthesis for Distributed Protocols," *Computer Safety, Reliability, and Security*, edited by A. Casimiro, F. Ortmeier, F. Bitsch, and P. Ferreira, Springer International Publishing, 2020, pp. 133–149.