Decision-Theoretic Monitoring of Cyber-Physical Systems

by

Andrey Yavolovsky
B.S. (Ivane Javakhishvili Tbilisi State University) 2006
M.S. (Ivane Javakhishvili Tbilisi State University) 2009

THESIS

submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2018

Chicago, Illinois

Defense Committee:
Miloš Žefran, Chair and Advisor, Electrical and Computer Engineering
A. Prasad Sistla, Advisor, Computer Science
Piotr Gmytrasiemicz, Computer Science
Venkat Venkatakrishnan, Computer Science
Mojtaba Soltanalian, Electrical and Computer Engineering

ACKNOWLEDGMENTS

This work would never be possible without the guidance and help from my advisors Prof. Miloš Žefran and Prof. A. Prasad Sistla. Your supportive guidance led me through the field that was totally new to me in the beginning but became a part of my common language in the end. Your suggestions and recommendations helped me to achieve the results that I could publish and present in this thesis.

I'm forever thankful for the continuous financial support to the University of Illinois At Chicago (UIC) Computer Science Department, UIC Electrical and Computer Engineering Department, National Science Foundation (NSF), Prof. Žefran and Prof. Sistla. I would never be able to complete this work without your input.

My interest in robotics has originated from the time when I was working in Tbilisi, Georgia for the EMCoS (Electromagnetic Compatibility Consulting and Software) company. I'm indebted to Prof. Roman Jobava for all the opportunities and skills that I have developed while I was part of EMCoS team. All of my motivation for joining the UIC for the Ph.D. program took a start at that time, and I'm exceptionally happy that I was able to make it happen.

Many thanks go to my friends from UIC, especially my labmates from Robotics Lab. Being a member of Robotics lab, where I got exposed to all the beauty of Robotics, a field that I was always truly admiring, was a delightful experience. I extremely value all the discussions that we had internally with Ehsan Noohi and Yao Feng, that helped me to move forward and find the answers to a number of hard problems. I thank all of my labmates: Wen Jiang, Sina

Parastegari, Maria Javaid and Bahareh Abbassi. Your support and help were very important to me.

I thank my parents, grandmother and my sister who always believed in me. Everything that I was able to achieve is due to all the efforts that you put in raising me with the love of technology and engineering in mind. To my wife, Natasha, for the unconditional love that made all of this possible. To my daughter who made me the happiest father in the world, and constantly reminded me that I need to take breaks from work and spend time with her. Being in love and being loved was the most important factor that helped me to constantly move forward towards my goals.

AY

CONTRIBUTION OF AUTHORS

A version of Chapter 3, 4, 5, 8 has been published in Runtime Verification Conference as:

Yavolovsky A., Žefran M., Sistla A.P. (2016) Decision-Theoretic Monitoring of Cyber-Physical Systems. In: Falcone Y., Sánchez C. (eds) Runtime Verification. RV 2016. Lecture Notes in Computer Science, vol 10012. Springer, Cham

For this paper [1] I was the first author and primary investigator, responsible for the development of methodology, implementation of the algorithm, design of the experiment and collection of data. Miloš Žefran and A. Prasad Sistla have been supervisory authors, who contributed in the concept formation and paper editing.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPS	Cyber-Physical System
DTMT	Decision-Theoretic Monitoring Tool
EHMM	Extended Hidden Markov Model
EU	Expected Utility
HMC	Hidden Markov Chain
HS	Hybrid System
IDE	Integrated Developmental Environment
IoT	Internet-of-Things
IRL	Inverse Reinforcement Learning
LTL	Linear Temporal Logic
MDP	Markov Decision Process
MEU	Maximum Expected Utility
NSF	National Science Foundation
OS	Operating System
PHA	Probabilistic Hybrid Automata
PHS	Probabilistic Hybrid System

LIST OF ABBREVIATIONS (Continued)

POMCP	Partially Observable Monte-Carlo Planning
POMDP	Partially Observable Markov Decision Process
PTSCC	Persistent Terminal Strongly Connected Component
ROS	Robot Operating System
RPM	Revolutions Per Minute
TSCC	Terminal Strongly Connected Component
QA	Quality Assurance
UCB1	Upper Confidence Bounds
UIC	University of Illinois at Chicago

SUMMARY

Cyber-physical systems (CPS) represent "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components". They can be found in such areas as aerospace, manufacturing, transportation, entertainment, healthcare, and automotive. For some of these systems the top priority is correct functioning; incorrect operation of systems like autonomous vehicles, medical devices or aircraft may lead to catastrophic consequences. But formally proving system correctness is a challenging problem. In recent years, runtime monitoring, where a monitor observes the outputs of the system and determines whether a system specification has been violated, has emerged as an attractive alternative.

In this thesis, we focus on the decision-theoretic approach to monitoring of safety properties in CPS. In particular, we formulate the monitoring problem as a Partially Observable Markov Decision Process (POMDP) whereby deciding whether a run is safe corresponds to executing the optimal policy of the monitoring POMDP. We show how Monte-Carlo planning algorithm (POMCP) can be used to compute the optimal policy of the monitoring POMDP. The monitoring POMDP reward structure is naturally described with four parameters and an important question is how it affects the monitoring performance, quantified through acceptance accuracy, rejection accuracy and monitoring time. We analyze the performance of the POMDP-based monitors for special choices of the reward structure and compare them with the performance of the traditional threshold-based monitors. Our results show that using POMDPs we can some-

times simultaneously improve both accuracies of the monitor while decreasing the monitoring time.

Further, we study POMDP-based monitors for systems with terminal strongly connected components. For this class of systems, we derive the expressions for the POMDP value function. The expressions allow us to demonstrate how the POMDP-policy can take advantage of the properties of the monitored system and thus provide an enhanced monitoring performance.

In order to study decision-theoretic monitors and experiment with them, we have developed a software called Decision-Theoretic Monitoring Tool (DTMT). This tool implements the architecture that allows easy integration of new monitoring decision rules and experimentation with an arbitrary user-defined system and property models.

Finally, we evaluate POMDP monitors on an experimental robotic system. We simulate the operation of a simplified transmission system on a mobile robot that was developed specifically for this purpose. The experiments confirm that POMDP-based monitors provide an improvement over traditional threshold-based approaches. Further, we show that POMDP-based monitors implemented through POMCP can be used online even for large problems and thus provide an attractive and flexible alternative to traditional threshold-based monitors.

CHAPTER 1

INTRODUCTION

1.1 Cyber-Physical Systems

The term *Cyber-Physical System* (CPS) became widely used in a recent decade. Cyber-physical systems represent "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components" [2].

Cyber-physical systems are all around us and more are being added every day. Examples include completely autonomous self-driving vehicles, complex surgical robotic systems, smart home systems, Internet of Things (IoT) devices and many more. Cyber-physical systems can be found in such areas as aerospace, manufacturing, transportation, entertainment, healthcare, and automotive.

Despite extensive research on the cyber and the physical aspects of systems, there are a lot of challenges that have to be addressed when these components are integrated into a single system. Physical and behavioral interactions involved at a system level combined with a number of theoretical models and formal representations make it exceptionally hard to verify safety and correctness. [3].

Generally, complexity that arises during a design, analysis, and validation of CPS is caused by several reasons [4]. An interactive and inter-dependent behavior of cyber and physical parts have to be considered when a system is designed. Also, typically, the semantic domains of

physical and cyber parts are very different. It is often the case that a physical system represents continuous dynamics modeled with differential equations. Meanwhile, a cyber part represents a discrete dynamics. Integration and abstraction of such systems pose a major challenge.

CPS technology is expected to transform the way people interact with engineered systems. With a development and better understanding of the benefits of "Smart Cities" and the IoT we are able to take our life to a totally new level, that once could be imagined only in science-fiction novels. Research advances in CPS promise to give answers on how to develop systems that have a faster response time, are precise, dependable, highly efficient and exceptionally reliable [2].

1.2 Motivation

Both hardware and software parts of any CPS must be highly dependable and configurable. However, for some systems the top priority is correct functioning under any external or internal conditions. Inconsistent operation of systems like autonomous vehicles, weapons, medical devices, and aircrafts may lead to catastrophic consequences. In order to guarantee correct functioning of the developed systems a lot of engineering resources are used. For example, in aviation, over 50% of all required development resources are dedicated on certification [5]. A special attention is dedicated to the systems, which if failed may lead to conditions that are determined to be unacceptable. Such systems are called *safety-critical* systems [6].

It's crucial that correctness and safety are guaranteed for the safety-critical systems, but proving such properties is a challenging problem. Some systems might be exceedingly large and have great complexity caused by the number of different involved domains. At the same time,

another challenge is to define an approach that will be easy to configure and fast enough to be applicable for large systems.

1.3 Real-Time Monitoring

In some cases, a requirement that the system operation is correct in any circumstance may be slightly relaxed. Instead, a substantial list of test scenarios may be generated and used during the quality assurance (QA) process. Clearly, even though it may be shown that system performs correctly for these testing scenarios, it does not verify in general the complete system.

Formal verification techniques represent a class of approaches that may be used to prove the correctness of a system. However, those techniques might not be feasible for some large real-world applications, especially those including numerous controls and networked microprocessors. It has been shown that the problem of verification is in general undecidable [7].

An alternative way to guarantee correctness is to monitor the behavior in real-time. This is particularly practical for a sufficiently large number of systems that may be implemented using fail-safe procedures. A number of different challenges arise for the monitoring, e.g. whether the system is fully or partially observable, and whether transitions are non-deterministic.

In previous work, real-time safety monitoring of CPS is based on the *probability estimation* that an execution is "bad" [8–10]. In our work, we are looking to develop decision rules that are able to improve on existing monitoring approaches. We are interested in seeing the monitor as a decision system, which is penalized for acting incorrectly and rewarded for performing well. This represents a traditional decision-theoretic view of the monitoring problem. *Our goal*

is to define, solve and analyze properties of the decision-theoretic monitors for cyber-physical systems.

1.4 Hypothesis

Every monitoring technique leads to a certain expectation of the set of outcomes, which primarily are defined by the accuracy and monitoring time. In this thesis, the primary focus was towards decision-theoretic monitoring approach, specifically the application of Partially Observable Markov Decision Processes (POMDPs). The monitoring accuracy and time to correctly distinguish a failure are determined by the rewards and penalties used to guide the decision of the monitoring POMDP. The central claim of this thesis is:

> The decision-theoretic monitoring approach based on POMDPs may significantly improve performance measures of safety properties monitoring in CPS, when compared to threshold-based approach.

While threshold-based monitors employ the probability whether a perceived set of system outputs was generated in a "good" or "bad" execution, this might not capture certain aspects of monitored systems, and, therefore, would not take advantage of that. Our expectation is that POMDP-based monitors declare decision rule which is more advanced, as it considers a combination of current data together with predicted variants of the future.

1.5 Thesis Organization

Chapter 2 starts by listing the background topics and theoretical foundations that are needed for the further development of the thesis subject. In the same chapter, we list and discuss related work on the relevant subjects, and identify how our work is different.

In Chapter 3 we initiate a general discussion on the decision-theoretic approach for CPS. We define the set of actions for decision-theoretic monitors, and propose a set of rewards that are required in order to derive the decision rule of a rational decision-theoretic monitoring agent. We continue by describing the previously studied threshold-based monitors in the decision-theoretic framework and formally define a way to approximate such monitors using a finite set of computations.

POMDP-based monitoring is introduced in Chapter 4. First of all, we introduce the set of general building blocks used to specify a POMDP, and specifically indicate how a monitoring problem may be formally represented as a monitoring-POMDP. We present the decision rules of the monitoring-POMDP and continue with the search of the optimal POMDP policy that maximizes the total cost of each monitoring action. We show that monitoring problem defined in the form of the POMDP is rather complex while solving even small POMDPs exactly is known to be computationally intractable. Thus, we define an adapted Partially Observable Monte Carlo Planning (POMCP) algorithm for monitoring, which we propose as a recommended approach considering specifics of the monitoring problem. At this time we also explain that to make POMDP-based monitoring applicable it is required that systems are fully autonomous or a set of control inputs is fully given in advance.

In Chapter 5 we focus on the parametrization of POMDP-based monitors. We identify that POMDP-based monitoring decision depends on the cost of taking a specific action and a depth of POMDP horizon. We show how to reduce the total number of reward values participating in the monitoring-POMDP value function, and identify classes of reward assignments that lead to equivalent monitoring performance. Here we start comparing the class of POMDP-based monitors with a class of threshold-based monitors. In order to do so, we give a definition that formalizes how a pair of monitoring approaches can be compared. We show that POMDP-based monitors represent a class of conservative monitors, i.e. detect a failure eventually in every "bad" execution, when configured rationally to do so. We study POMDP-based monitors with horizons $h = 1$ and $h = 2$ and give the conditions on rewards that provably make POMDP-based monitors equivalent or less efficient when compared to an approximation of threshold-based monitors.

From considering a general set of problems we continue into a specific class of systems, which are characterized by the terminal strongly connected components (TSCCs). This is discussed in Chapter 6. We use the property of TSCC in conjunction with the specifics of the monitoring problem to simplify the decision rule of POMDP-based monitors. As a result, we obtain a set of expressions for the POMDP value functions, that are independent of the probability functions on the system outputs, which are commonly present in the case of general POMDP. We introduce further assumptions that allow us to reduce decision rules of POMDP-based monitor to a condition that compares a probability that system has not yet failed with some probability interval. We discuss how resulting decision rule is related to the decision procedure

of threshold-based monitors, and what are the conditions when POMDP-based approach may take the lead.

Chapter 7 is dedicated to a Decision-Theoretic Monitoring Tool (DTMT), which is a tool that was developed and implemented to support our experimental study. We present the architecture design of DTMT, and show how it may be used with an arbitrary decision rule, system, and a monitored property. We list the parameters and modes in which DTMT may be used and give the instructions on how the tool may be accessed on the Internet for a local deployment.

In Chapter 8 we present an experimental analysis of the POMDP-based monitor. To perform an experiment we came up with the model of a simplified transmission system. We present details about the way the experiment was conducted and the hardware components that have been built together to collect real sensor data. We show and discuss the experimental results, that confirm that POMDP-based monitors may be configured to perform better than tradition threshold-based approach.

Chapter 9 discusses few possible extensions to our work. In particular, we suggest investigation of the variation of threshold-based monitors such that multiple decision rules are combined together in a deterministic or non-deterministic fashion. We also include discussion about the Inverse Reinforcement Learning (IRL) technique as an approach for recovering the POMDP reward function from the policy.

Finally, we present a concluding summary of the thesis in Chapter 10.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Assembling Monitoring of CPS

In this section we present the notions necessary for a development of a core theory behind a formal representation of CPS and a monitored property. Most of the definitions, unless explicitly indicated, are based on the work done in [8–10].

2.1.1 Sequences

For the given set S, let S^* and S^ω represent the set of finite and infinite sequences over the elements of S respectively. The length of a finite sequence α over the set S is represented as $|\alpha|$. For a potentially infinite sequence $\sigma = s_0, s_1, \ldots$ of elements from S and for any arbitrary value of $i \geq 0$, $\sigma[0, i]$ represents the prefix of σ up to s_i. The concatenation of two sequences α_1 and α_2 is denoted as $\alpha_1\alpha_2$, where α_1 is a finite sequence, and α_2 is either a finite or an infinite sequence over S.

2.1.2 Safety Properties

For any infinite sequence $\sigma \in S^\omega$ the set of all prefixes of σ is represented as prefixes(σ). If we consider any subset $C \subseteq S^\omega$, then the set of all prefixes of C is represented by a union $\text{prefixes}(C) = \cup_{\sigma \in C}\big(\text{prefixes}(\sigma)\big)$. A set $C \subseteq S^\omega$ is a *safety* property if for any $\sigma \in S^\omega$ satisfying the condition $\text{prefixes}(\sigma) \subseteq \text{prefixes}(C)$ it also holds that $\sigma \in C$.

2.1.3 Automata

A deterministic Streett automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, q_0, F)$. The set Q represents a set of states, while the set Σ defines an input alphabet. A transition function of the automaton is $\delta : Q \times \Sigma \to Q$, q_0 is the initial state and F gives a collection of pairs of subsets of states. Given an infinite input sequence $\sigma = \sigma_0, \ldots$ let $r = r_0, \ldots, r_i, \ldots$ represent a *run* of automaton over the σ, where $r_0 = q_0$ and $\forall i \geq 0$, $r_{i+1} = \delta(r_i, \sigma_i)$. An *accepting* run is such that for all the pairs $(C, D) \in F$ the existance of infinitely many i such that $r_i \in C$ implies the existance of infinitely many j such that $r_j \in D$. Acceptance of an input string by the automaton is defined by the existance of an accepting run on that string, which starts from the initial state q_0. $L(A)$ is defined to represent a set of input strings accepted by the automaton $\mathcal{A}$. A deterministic Streett automaton $\mathcal{A}$ is called a *safety automaton* if $F = \{(\{q_{error}\}, \emptyset)\}$, i.e. contains a single pair, where q_{error} is a sink state. All the transitions from state q_{error} go back to itself. The state q_{error} is called the *error* state. It follows from the definition that the infinite input sequence is *accepted* iff the run defined on these inputs does not contain the *error* state.

2.1.4 Markov Chains

A Markov chain is defined as a triple $G = (S, R, \phi)$. The set S represents a countable set of states, while $R \subseteq S \times S$ establishes a total binary relation on the set S. The function $\phi : R \to (0, 1]$ represents a probability that a system switches from the state s to state t in a single step. It is required that for any state $s \in S$, the sum $\sum_{(s,t)\in R} \phi\big((s, t)\big) = 1$. By considering a sequence of states $s_0, s_1, \ldots, s_n$ such that every pair of consequent states $(s_i, s_{i+1}) \in R$ for $0 \leq i < n$ we obtain a finite path p of the Markov chain. Provided that p is a non-zero length

path the probability of the path is calculated as $\phi(p) = \prod_{0 \leq i < n} \phi\big((s_i, s_{i+1})\big)$. The probability of zero length path is assumed to be $\phi(p) = 1$. The infinite paths of G are defined analogously. Stochastic systems given as Markov Chains [11] can be monitored for the satisfaction of a property given by an automaton or a temporal formula.

In order to specify properties over the sequences of states of a Markov chain G we use automata with input symbols from the state space of G. It has been shown that, for any automaton $\mathcal{A}$, $L(\mathcal{A})$ is measurable [12].

Monitors that we would like to define are supposed to react on the sequences of system states modeled by G. A monitor ensures that operation of a system satisfies the property given by an automaton $\mathcal{A}$. However, we consider systems where a system state is not directly observable. Instead, only the sequences of system outputs are available.

2.1.5 Hidden Markov Chains

A Hidden Markov Chain (HMC) [13] $H = (G, \Sigma, O, r_0)$ is a quadruple such that $G = (S, R, \phi)$ is a Markov chain, Σ is a set of output symbols, $O : S \to \Sigma$ is an output function, and $r_0 \in S$ is an initial state. Given the state $s \in S$ function $O(s)$ represents the output of the H when system is in that state. In the context of HMC an actual state is not observable. The only information that is available for the external observer is from a set of generated outputs. In the definition of the HMC given above there is no mention of the probabilistic observation model, that would define the probability of an observation from the set O for the given state. Such observation function can be frequently seen in traditional definitions of HMC considered in literature. Traditional HMC models can be converted to the representation with deterministic

observations by substituting the state space with the product $S \times \Sigma$ and adjusting the function ϕ of the underlying Markov chain.

Monitoring of systems modeled as HMC reduces to using the observed output symbols in order to infer the satisfiability of the system execution with respect to a given property. Although HMC represents a convenient model to represent the class of partially observable systems it does not extend to a more general scenario, where system state is hybrid and consists of continuous and discrete variables.

2.1.6 Extended Hidden Markov Model

An Extended Hidden Markov Model [9] is an extension over the traditional HMCs for the stochastic dynamic systems over discrete time with both discrete and continuous variables. Let $\mathbb{N}$ and $\mathbb{R}$ represent the set of natural and real numbers respectively.

Consider a vector $\sigma = (\sigma_0, \ldots, \sigma_{n-1})$, such that every $\sigma_i \in \{0, 1\}$. We define a hybrid domain $S_\sigma = T_0 \times \cdots \times T_{n-1}$, where $T_i = \mathbb{N}$ or $T_i = \mathbb{R}$ depending on whether $\sigma_i = 0$ or $\sigma_i = 1$ respectively.

A probability function over the hybrid domain S_σ is defined as follows. A function $\mu : S_\sigma \to [0, \infty]$ is a probability function if it is a measurable function and $\sum_{x_1 \in \mathbb{N}^{n_1}} \int_{\mathbb{R}^{n_2}} \mu(x_1, x_2) dx_2 = 1$, where x_1 is a vector of n_1 variables such that each is in $\mathbb{N}$, and x_2 is a vector of n_2 variables such that each is in $\mathbb{R}$. The integration is assumed to be done using a Lebesgue integral [14]. In the summation, at first, the function μ is integrated over the continuous variables, while discrete variables are fixed, and then everything is summed up over all the assignments of the discrete variables. Function μ represents a standard density function if $S_\sigma = \mathbb{R}^n$, i.e. $n_1 = 0$.

An Extended Hidden Markov Model is defined as follows. Let $n_1, n_2, m_1, m_2 \geq 0$ be integers and σ_1, σ_2 be the vectors $0^{n_1}1^{n_2}$ and $0^{m_1}1^{m_2}$, respectively. Intuitively, n_1, n_2 give the number of discrete and continuous state variables, while m_1, m_2 gives the number of discrete and continuous outputs of the system being described. An Extended Hidden Markov Model (EHMM) $\mathcal{H}$ of dimensions (n_1, n_2, m_1, m_2), is a triple (f, g, μ) defined as follows. The function $f : (S_{\sigma_1} \times S_{\sigma_1}) \to [0, \infty)$ is a *next state function.* For any fixed value $x \in S_{\sigma_1}$, function $f(x, y)$ represents a *probability function* on S_{σ_1} in y. A function $g : (S_{\sigma_1} \times S_{\sigma_2}) \to [0, \infty)$ is an *output function.* For any appropriate fixed value $x \in S_{\sigma_1}$, function $g(x, z)$ represents a probability function on S_{σ_2} in z. To simplify the notation, we will denote S_{σ_1} with S, and S_{σ_2} with Σ. Both functions $f(x, y)$ and $g(x, z)$ are measurable in both arguments, i.e. for a fixed value of first argument they these functions are measurable in the second argument, and vice versa. Finally, μ describes the probability of the initial state, i.e., it is a probability function on S_{σ_1}.

In our work we are mostly focused on systems that produce observations based on sensor readings. It's a common approach to acquire readings from those sensors using analog-to-digital converters. Therefore, we assume that the set of output symbols is discrete and finite. Thus, in the definition of EHMMs we assume that $m_2 = 0$, i.e. there are no observations that come from the continuous space.

2.1.7 Probabilistic Hybrid Systems

A hybrid system (HS) is a dynamic system whose evolution is characterized by both symbolic (discrete) and continuous variables. We are interested in a particular subclass of hybrid systems, *probabilistic hybrid systems(PHS)* [9, 15].

Formally, a probabilistic hybrid system $\mathcal{A}$ is a tuple $(Q, V, \Delta t, \mathcal{E}, \mathcal{T}, c_0)$. The set Q represents a countable set of discrete modes, i.e. discrete states of $\mathcal{A}$. The set V represents all the state variables, output variables and the variables representing noise processes. For every mode $q \in Q$ in $\mathcal{A}$ the function $\mathcal{E}$ defines a set of equations. The evolution of the continuous state is given with discrete-time state equations. The value of output and noise variables is given through corresponding equations at time step $t + \Delta t$, where Δt represents the sampling time. The transitions are given as pairs (ϕ, p), such that ϕ is a measurable predicate that is computed over a subset of continuous or discrete state variable, while p is a probability distribution over Q. The function $\mathcal{T}$ associates a set of transition pairs to every mode $q \in Q$. Finally, a discrete mode and an initial continuous probability distribution on state variables is given by c_0. It is expected that the equations defining continuous state variables also employ the noise variables, while the set of transition pairs for every mode declare guard conditions that are mutually exclusive and exhaustive.

For every PHS it is possible to define a corresponding EHMM. Indeed, for every mode q the dynamic evolution of the PHS is given by the set of difference equations. A transition from any mode q to a mode q' is probabilistic and occurs with a probability distribution p when a specific guard condition ϕ is satisfied, i.e. $(\phi, p) \in \mathcal{T}(q)$.

The semantics of PHS $\mathcal{A}$ may be represented in terms of the corresponding EHMM $\mathcal{H}_{\mathcal{A}}$. This is easy to conclude due to the lack of deterministic resets to variables in the transitions.

2.1.8 Monitors

Let Σ be a set of output symbols generated by the monitored system modeled as an EHMM $\mathcal{H}$. Formally [8], a monitor $M : \Sigma^* \to \{0, 1\}$ is a function such that for any $\alpha \in \Sigma^*$, if $M(\alpha) = 0$ then $M(\alpha\beta) = 0$ for every $\beta \in \Sigma^*$. A finite sequence $\alpha \in \Sigma^*$ is called to be rejected by the monitor M if $M(\alpha) = 0$. Otherwise, if $M(\alpha) = 1$ then we say that M accepts α. By definition, every extension of rejected sequence α is also rejected by M. If α represents a prefix of infinite sequence $\sigma \in \Sigma^\omega$ and α is rejected by M then we say that M rejects σ. If σ has no prefix rejected by M then it is accepted by the monitor.

2.1.9 Accuracy Measures

Let $\mathcal{P}$ be a safety automaton on the states of EHMM $\mathcal{H}$. The *acceptance accuracy* [8] of the monitor M for the property $\mathcal{P}$ with respect to the EHMM $\mathcal{H}$, defined as $AA(M, \mathcal{H}, \mathcal{P})$, is "the conditional probability that an output sequence generated by the system is accepted by M, given that it is in $L(\mathcal{P})$" [8]. The *rejection accuracy* of M for $\mathcal{P}$ with respect to $\mathcal{H}$, defined by $RA(M, \mathcal{H}, \mathcal{P})$ represents "the probability that an output sequence generated by the system is rejected by M, given that it is not in $L(\mathcal{P})$" [8]. Intuitively, the *acceptance accuracy* represents the probability that good runs generated by $\mathcal{H}$ are accepted by the monitor M, while the *rejection accuracy* is the probability that bad runs are rejected.

2.1.10 Monitoring Time

In addition to the accuracy measures of a monitor, it is important to consider the time it takes to detect a failure in a system execution. Definition of the monitoring time can be given

only for the case when a safety property is considered, i.e. there is a valid definition of "good" and "bad". Monitoring time [10] is formally defined as follows.

Let σ be a bad execution of EHMM $\mathcal{H}$. A monitor M rejects the execution σ according to a safety property $\mathcal{P}$. Any prefix of σ that is rejected by $\mathcal{P}$ is called a bad prefix of σ. In order to represent the monitoring time of the monitor M it is necessary to consider the smallest bad prefix of σ, i.e. such a prefix of σ that the failure has just occurred. Assume that $\sigma[0:t]$ is the smallest bad prefix of σ, however, the time instance when the failure was detected by the monitor is t'. The difference $t' - t$ represents the elapsed time before the monitor signaled a failure. The monitoring time $MTIME(M)$ denotes an expectation of this elapsed time under the condition that an execution was rejected correctly, i.e. the execution was bad.

Note, that this definition does not assume or enforce that the monitor M rejects the execution σ immediately at or after time instance t, i.e. it is possible that $t' < t$. In that case the difference $t' - t$ may be negative, and also the expectation $MTIME(M)$ might be negative.

2.1.11 Monitorability

As we are concerned about the values of accuracy measures that can be achieved by applying a certain monitor, we are also interested whether those values may be arbitrary high. An answer to this question is partially given by the definition of *monitorability* [8].

A system $\mathcal{H}$ is called to be *strongly monitorable* with respect to the property $\mathcal{P}$ if there exists a monitor M such that $AA(M, \mathcal{H}, \mathcal{P}) = RA(M, \mathcal{H}, \mathcal{P}) = 1$. Strong monitorability essentially declares that there exists a monitor that is able to reject all bad execution and accept all good executions. Such property is in general difficult to satisfy. A system $\mathcal{H}$ is called to be

monitorable if for any $x \in [0,1)$ there exists a monitor M, such that $AA(M,\mathcal{H},\mathcal{P}) \geq x$ and $RA(M,\mathcal{H},\mathcal{P}) \geq x$. Intuitively, a system is *monitorable* if given enough observations the monitor can distinguish a good execution from a bad execution with arbitrary high probability.

It has been shown that in [16], the problem of checking monitorability for finite state systems is decidable in polynomial time.

2.1.12 Threshold-Based Monitors

A threshold-based approach for monitoring of a cyber-physical system with respect to a safety or a liveness property was given and studied in [8–10]. The operation of the threshold-based monitor is controlled by the selected value z of a probability threshold and is given as follows.

We define a monitor for the system specified as EHMM $\mathcal{H}$ with respect to a property $\mathcal{P}$. At every step of a system execution a new output symbol is observed. Let α be a finite sequence of outputs seen from the beginning of the execution until now. The acceptance probability [8] $AccProb(\alpha)$ represents the conditional probability that the system execution that originally generated the output sequence α is accepted by the property $\mathcal{P}$. Similarly, a rejection probability $RejProb(\alpha)$ represents a conditional probability that the system execution is rejected by the property. According to these definitions for every finite output sequence α it is obvious to see that $AccProb(\alpha) + RejProb(\alpha) = 1$.

The threshold-based monitors work by comparing a value of $RejProb(\alpha)$ with a threshold-value $z \in [0,1]$. The monitor rejects an execution that generated an output sequence α when $RejProb(\alpha) \geq z$. In this case z represents a *rejection threshold*. Similarly, the threshold-based

monitors can be defined with respect to the $AccProb(\alpha)$. Then the monitor rejects an execution if $AccProb(\alpha) \leq 1 - z$, where $1 - z$ represents an *acceptance threshold*.

Formally the threshold-based monitors are defined as follows:

$$M(\alpha) = \begin{cases} 0 & RejProb(\alpha) \geq z \\ 1 & RejProb(\alpha) < z \end{cases} = \begin{cases} 0 & AccProb(\alpha) \leq 1 - z \\ 1 & AccProb(\alpha) > 1 - z \end{cases} \tag{2.1}$$

From the definition it is clear that by selecting different values of the rejection threshold z the monitor will have different acceptance and rejection accuracy (AA and RA). Specifically, for the case of $z = 0$ the monitor will reject every execution of the system $\mathcal{H}$, and, therefore, $RA = 1$, however $AA = 0$, since all good executions will be rejected. On the other side, for the case of $z = 1$ the monitor will accept any execution, and imply the accuracies $RA = 0$ and $AA = 1$. It has been shown [8] that given the system that is monitorable with respect to the property the accuraccies AA and RA approach 1 as z is approaching 1.

The value of the $RejProb(\alpha)$ can be calculated for certain systems. E.g., if $\mathcal{H}$ and $\mathcal{P}$ are finite systems, then their product can be constructed and $RejProb(\alpha)$ can be obtained using standard techniques [17]. However, the above approach would not work for the case of infinite systems, and also may not be efficient for the case of finite systems. Due to these reasons, implementation of the threshold-based monitors presented in [8–10] is done by the approximation of $RejProb(\alpha)$ with a lower bound. The lower bound is defined by the probability that the current state of the system violates the property.

2.2 Assembling a Decision-Theoretic Approach

2.2.1 Intelligent Agents and Decision Theory

When referring to the term of intelligent agents we will assume the definition given in [18]. A notion of an intelligent agent is one of the basics of a modern approach to the Artificial Intelligence. Every agent simply represents something that acts, but a rational agent is acting to achieve the best possible outcome. When there is no way to evaluate an outcome exactly, e.g. under uncertainty, the rational agent acts by maximizing the expected outcome given the information about an environment. In a case when there is a limited amount of information available to the agent, or time to make all the necessary computations is bounded the agent is said to have a *limited rationality*.

Rational agents can be built by implementing a system that follows the rules defined by the decision theory. While the probability theory describes how agent's belief is related to a perception of an environment, the utility theory specifies goals that the agent wants to achieve. The decision theory combines probability theory and utility theory together to describe what the rational agent should do [18].

A goal of the utility theory is to define a way to distinguish between the values of different possible outcomes of various actions. This leads to the definition of a fundamental idea of the decision theory: a principle of a maximum expected utility (MEU). Let a represent agent's action and $EU(a)$ be the expected utility function, which quantifies numerically the desirability of an action using a probabilistic model of the outcomes possible in the future. According to the

MEU principle, the rational agent should choose the action that results in the highest expected utility.

$$action = \arg\max_{a} EU(a) \tag{2.2}$$

The simplest decision theoretic agent will choose an action based on the attractiveness of the immediate outcomes only. In this case, expected utility function for a given action is simply a weighted sum over the immediately obtained utilities. Alternatively, by considering known information about the model it is possible to obtain the expectation of utility function over the finite or infinite future horizon (assuming convergence for the expected utility value).

2.2.2 Sequential Decision Problems

When the decision of an agent on which action to select does not depend on the decisions made in the past, the problem is called to be an *episodic decision problem*. In that case, the operation of an agent is separated into atomic independent episodes with its own percepts and a single selected action. However, in some cases, the current decision may affect all the future decisions. In that case, a problem of action selecting is called a *sequential decision* problem.

For the fully observable environments with Markovian state transitions the sequential decision problems are commonly defined with a formal model called *Markov Decision Process* (MDP) [18]. Formally MDP is defined as a tuple $\{S, A, T, R, \gamma\}$, where S is a finite set of states, A is a finite set of actions, T is a probabilistic transition function, R is a reward function and γ is a discount factor. For any given pair of states $s, s' \in S$ and action $a \in A$ the transition function $T(s, a, s')$ defines the probability $Pr(s'|s, a)$, i.e. probability that by executing action a from the state s the agent will end up in the state s'. The reward function $R : S \times A \to \mathbb{R}$ de-

fines an immediate cost of executing an action from the given state. A discount factor $\gamma \in [0, 1]$ defines the importance of future rewards in choosing which action to execute next.

During the sequential decision process, at each step, an agent has to decide which action to perform. A function $\pi : S \to A$ is a *policy* of the MDP that defines the corresponding action for every state from S. The policy is characterized by the value function $V(\pi)$, that represents the expected utility gained by the agent for every state when the policy π is followed.

$$V(\pi) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in S} T\big(s, \pi(s), s'\big) V^{\pi}(s') \tag{2.3}$$

The goal of an agent, and the solution of the MDP, is an optimal policy π^* that would maximize the value function V for every state $s \in S$. The optimal value function is

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right] \tag{2.4}$$

Optimal value function is calculated by employing value-iteration [19] using the Bellman equation:

$$V_{t+1}(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_t(s') \right] \tag{2.5}$$

While quantifying the expected utility for every policy there is a question of consideration of a finite or infinite horizon to make a decision. With a finite horizon, an agent is calculating the expected reward only for the finite number of future steps. This makes an optimal policy function for the finite horizon to be a *nonstationary* function, which can change over time. An

optimal policy for a case of the infinite horizon is, however, a stationary function, since agent's decision is the same independently from the time.

The value-iteration algorithm computes the optimal value function by calculating a sequence of optimal finite-horizon value functions $V_0^*, V_1^*, \ldots, V_t^*$ until the convergence condition $\max_{s \in S} \left| V_{t+1}(s) - V_t(s) \right| < \epsilon$ is satisfied [20].

The value-iteration algorithm leads into the implicit MDP optimal policy defined with a Q-function as follows:

$$\begin{aligned} Q(s,a) &= R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \\ \pi^*(s) &= \arg\max_a Q(s,a) \end{aligned} \tag{2.6}$$

2.2.3 Sequential Decision Problems in Partially Observable Environments

In the case of partially observable systems, a sequential decision process requires an extension of the MDP model. A Partially Observable Markov Decision Process (POMDP) [18] is aimed to take limited observability of the state into consideration. POMDP is defined as a tuple $(S, A, T, R, O, Z, \gamma)$. Set of states S, set of actions A, transition function T, reward function R and discount factor γ are assumed to be same as in fully observable MDP. For every state s the agent may perceive an observation from the finite set O, but since the environment is partially observable this perception is nondeterministic. An observation function $Z(o, s, a)$ defines the probability $Pr(o|s, a)$, i.e. the probability that observation o is perceived after executing action a from the state s.

At any given time the agent is not able to observe and identify the exact state of the underlying POMDP model. The only information that is observable and available for the use of the agent at time t is a history or a trajectory, which may be represented as a sequence of pairs $< A_i, O_i >$.

$$H =< A_0, O_0 >, < A_1, O_1 >, < A_2, O_2 >, ..., < A_t, O_t > \tag{2.7}$$

However, as time goes forward this representation of the history grows. Therefore, it is convenient to compress the information about the past history into the belief state - a probability distribution over the states of the POMDP. It has been shown that such a representation of the belief state is sufficient to summarize the observable history of a POMDP without the loss of generality [21]. Simply saying, the belief state of a POMDP represents the set of states that an agent might be in and their probability. Usually, it is assumed that an initial belief state b_0, that represents information about the states before the first action is executed, is given. For the given belief state and any state $s \in S$, a value of the $b(s)$ represents agent's belief (probability) of occupying the state s.

Propagation of the belief state forward, essentially, represents a filtering task. Calculation of the new belief state given the current belief, chosen action and observation is usually called the *forward update*. Let $b(s)$ represent the agent's belief of occupying the state $s \in S$, then if

an action a is executed and the observation o is perceived we may compute the next belief state b' for every choice of next state $s' \in S$ as

$$b'(s') = \beta Z(o, s', a) \sum_{s \in S} T(s, a, s') b(s) \tag{2.8}$$

Normalizing coefficient β is chosen to make $\sum_{s' \in S} b'(s') = 1$.

Every POMDP can be converted into a belief-state MDP. The state space of the belief-state MDP is continuous and consists of possible belief states of the underlying POMDP. However, belief states are fully observable, and, therefore, satisfy requirements of the MDP. Belief-state MDP is formally a tuple $(B^s, A, T^b, R^b, \gamma)$. The set of actions A and the discount factor γ are the same as in the POMDP. B^s is the continuous space of beliefs over the state space S (the set of all possible probability distributions over S). The transition function $T^b : B^s \times A \to B^s$ is defined as:

$$\begin{aligned} T^b(s) = Pr(b'|b, a) &= \sum_{o \in O} Pr(b'|b, o) Pr(o|b, a) = \\ &= \sum_{o \in O} Pr(b'|b, a, o) \sum_{s' \in S} Z(o, s', a) \sum_{s \in S} T(s, a, s') b(s) \end{aligned} \tag{2.9}$$

Here $Pr(b'|b, a, o)$ is assumed to be 1 iff $b^o_a = b'$, where b^o_a is the belief state after the action a has been executed from the belief state b and the observation o was perceived. Reward function $R^b : B \times A \to \mathbb{R}$ is

$$R^b(b, a) = \sum_{s \in S} b(s) R(s, a) \tag{2.10}$$

From the constructed belief-state MDP it may be seen that choosing an action of the POMDP depends only on the current observable belief. The optimal POMDP policy is then $\pi^* : B \rightarrow A$. Similarly to the MDP, the implicit policy of the POMDP can be extracted from the following equations:

$$\begin{aligned} Q(b,a) &= \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in O} Pr(o|a,b) V^*(b_o^a) \\ \pi^*(b) &= \arg\max_{a \in A} Q(b,a) \end{aligned} \tag{2.11}$$

A complete decision cycle of the POMDP can be described with the following steps:

1. For the given belief b execute the action $a = \pi^*(b)$.
2. Perceive an observation o.
3. Forward update the current belief state b given the action a and the observation o.

POMDP value iteration is the MDP value iteration reduced to the continuous belief-state MDP. The value of the belief state b for the horizon $t+1$ is recursively calculated based on the value at the horizon t:

$$\begin{aligned} V_{t+1}(b) &= \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in O} Pr(o|a,b) V_t(b_o^a) \right] \\ Pr(o|a,b) &= \sum_{s' \in S} Z(s',a,o) \sum_{s \in S} T(s,a,s') b(s) \end{aligned} \tag{2.12}$$

The main problems of the exact POMDP solvers are defined as a "*curse of dimensionality*", and a "*curse of history*" [22]. Both problems are directly related to the complexity of the

POMDP policy search which is influenced by a dimensionality of the belief state, i.e. a number of states, and an exponential growth of the search space with the increase of a planning horizon.

2.3 Related Work

In our work we focus on the broad class of CPS, however, systems we are mostly interested in are commonly modeled as hybrid systems. A lot of literature has been focused on the modeling and control of hybrid systems [23–26]. Often, safety requirements are described by specifying states which are permissible or are forbidden.

Safety and liveness verification for hybrid systems has been extensively studied [7, 27]. It was shown that this verification problem is in general undecidable [7]. The problem of fault detection and diagnosis of hybrid automata has been studied in [15, 28–31]. In these works the goal was to identify when the automaton enters a fail state, which is very different from the monitoring a system with respect to a property given in an expressive formalism such as Linear Temporal Logic (LTL) [32].

Control synthesis for stochastic discrete-event systems has been studied in [33, 34] but only finite-state systems with directly observable state have been considered. In like manner, only the case of deterministic finite-state systems has been considered in the works on diagnosability of partially-observable discrete-event systems. Runtime monitoring for software programs modeled as (finite-state) HMMs has been studied in [36].

The question if the safety requirements can be intergrated into the design process was studied in [37–39]. Such an approach would make the verification or monitoring unnecessary.

However, these methods are not yet able to reasonably address systems with complex continuous dynamics, and are not suitable for stochastic systems.

An approach for monitoring and checking quantitative and probabilistic properties of real-time systems has been given in [40, 41]. A game-theoretic framework was employed for the monitoring interfaces for faults in [42], and in [43, 44] a conservative runtime monitors were proposed. However, none of these works are meant to work for the case of hybrid systems.

Issues of safety have been studied using the POMDP formalism [45–47]. However, in all of these works safety was considered to be a part of the system internally, i.e., actions need to be chosen to avoid or reduce the risk of failures. This significantly differs from the decision-theoretical analysis that we employ. Instead of controlling the system in a safe way, we design a monitoring system that needs to decide timely and with a reasonable accuracy if the system operation has failed. Thus monitoring POMDP identifies a policy that drives the action of the monitor and not the monitored system. Further, we consider systems with continuous state spaces.

CHAPTER 3

DECISION-THEORETIC MONITORING OF CPS

3.1 Decisions in Run-Time Monitoring Problems

The definition of the monitor was given in Section 2.1.8, however, no actions or decisions have been mentioned yet. It is obvious that one action that is important for every monitor is to raise an *Alarm* when a violation of the property is detected. Another action is the opposite of the first one and is essentially defined as not raising the *Alarm*. We call it *Continue*, since a monitor allows the system to continue execution.

In our work we see a decision-theoretic monitor as an intelligent *monitoting agent* that is required to pick one of the actions from the set $A = \{Alarm, Continue\}$. Even though this might not be the only way to define actions for a decision-theoretic monitor we find that every monitor should have at least these two actions and will thus not discuss this question further.

Note that actions of the monitoring agent have a different effect on the system execution. The action *Continue* allows the system execution to proceed, while the action *Alarm* is a terminal action: the system state entered after the action *Alarm* has been executed is treated as terminal.

A rational agent chooses an action that maximizes the expected utility for that action. Let α be a sequence of outputs perceived from the EHMM $\mathcal{H}$. Then, in general, the decision rule of a rational monitoring agent may be formulated as:

$$action = \arg\max_{a \in A} EU_a(\alpha) \tag{3.1}$$

The value of $EU_a(\alpha)$ is commonly defined by considering the uncertainties and non-determinism of a monitored system, as well as the numerical values of rewards associated with certain actions and system states.

3.2 Monitoring Safety Properties

Let $\mathcal{A}$ be a PHS (with the associated EHMM $\mathcal{H}_{\mathcal{A}}$) and assume that the property that has to be monitored is defined by the (deterministic) safety automaton $\mathcal{P}$. Let $\mathcal{B}$ be the product, $\mathcal{B} = \mathcal{A} \times \mathcal{P}$ (see [9] for details).

Intuitively, the execution of $\mathcal{B}$ corresponds to the run of $\mathcal{A}$ with $\mathcal{P}$ being simultaneously driven by the sequence of states of $\mathcal{A}$. Finally, let $\mathcal{H}_{\mathcal{B}}$ be the EHMM associated with $\mathcal{B}$.

Threshold-based monitors for safety properties, as defined in [9], compute the probability that a sequence of observed outputs is generated by the execution that is rejected by the property $\mathcal{P}$; this probability is approximated by the probability (belief) that the current state of $\mathcal{B}$ is bad (the second component of the combined state of $\mathcal{B}$ is a bad state of $\mathcal{P}$). If this probability is greater than the given threshold, the monitor raises an alarm.

We generalize the algorithm and extend it by substituting the threshold-based approach with a generic decision procedure. Figure 1 demonstrates the basic flow and building blocks of the monitoring process.

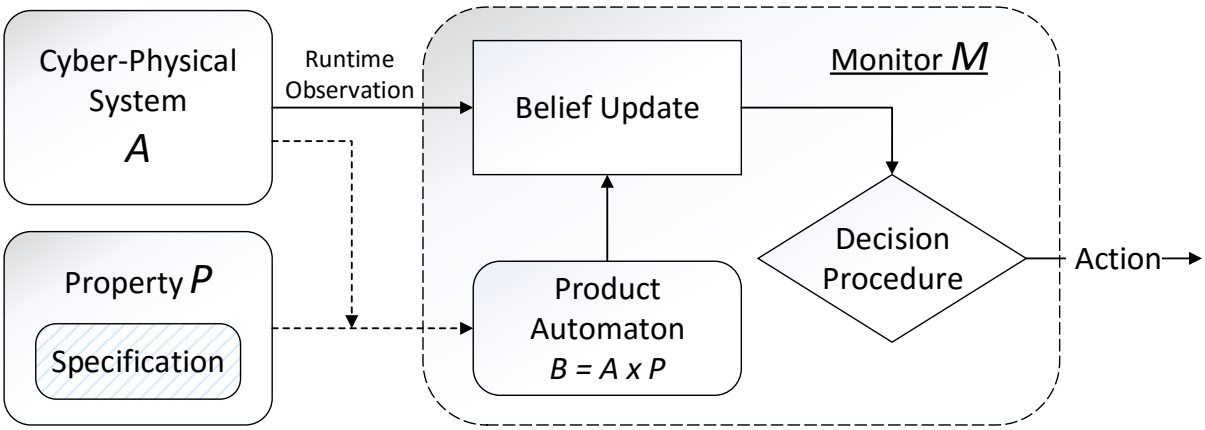

Figure 1. Monitoring algorithm.

Run-time outputs that are generated by the system $\mathcal{A}$ are used to maintain the belief state of the product automaton $\mathcal{B}$. This belief state is used by the decision procedure to select an action.

3.3 Monitoring Liveness Properties

Monitoring of properties specified as a liveness automata can be done by approximating it with safety automata using methods given in [43, 44]. Consider a Büchi automaton $\mathcal{A}$ that specifies a liveness property. We define a safety automaton $\mathcal{A}'$ by introducing the parametric

value T', which defines a value of timeout. An original automaton $\mathcal{A}$ is modified by the addition of the failure state, and transitions to that state happen if an accepting state is not reached within an interval of time T' after it was last visited or from the start. It's obvious that automaton A' might reject more executions than A. However, the automaton A' will reject every execution that would be rejected by A.

3.4 Monitoring Rewards

Rationality of the monitoring agent is driven by the costs associated with the execution of a particular action. We define a *reward function* for all $s \in S$ and $a \in A$ as follows. Assume that we can represent the set of states S (of $\mathcal{H}_{\mathcal{B}}$) as a union $S_{good} \cup S_{bad}$, where S_{bad} is a set of states that represent a failure, and S_{good} is a complement of S_{bad}. The reward function is

$$R(s,a) = \begin{cases} R_g^c \in \mathbb{R}_{\geq 0} & s \in S_{good}, a = Continue \\ R_g^a \in \mathbb{R}_{\leq 0} & s \in S_{good}, a = Alarm \\ R_b^c \in \mathbb{R}_{\leq 0} & s \in S_{bad}, a = Continue \\ R_b^a \in \mathbb{R}_{\geq 0} & s \in S_{bad}, a = Alarm \end{cases} \tag{3.2}$$

where $R_g^c \geq 0 \geq R_b^c, \quad R_b^a \geq 0 \geq R_g^a$.

The values of rewards $(R_g^c, R_b^c, R_g^a, R_b^a)$ define the monitor and will affect its performance in terms of an acceptance and rejection accuracies, a monitoring time. Given a state s, the optimal action is to *Continue* if the state is *good*, and *Alarm* if the state is *bad*. However, when system states are not fully observable, choosing any action entails certain risks. Those

risks are quantified by the values of rewards so that choosing a wrong action will gain a smaller reward. To emphasize undesirability of wrong actions we assign a negative value to R_b^c and R_g^a. Note that every time when a failure has not been detected a penalty in the form of R_b^c will be accumulated. Note also that a penalty for the false alarm will only be assigned once since the action *Alarm* is terminal.

We suggest that for the analysis of function calculated with the rewards it is convenient to substitute the rewards $(R_g^c, R_b^c, R_g^a, R_b^a)$ as follows:

$$\begin{aligned} R_g^c &= G^c \qquad R_b^c = -L^c \\ R_b^a &= G^a \qquad R_g^a = -L^a \end{aligned} \tag{3.3}$$

In the Equation 3.3 G^c and G^a represent the gain of executing actions *Continue* and *Alarm* respectively, while L^c and L^a represent a value of loss. Each of the G^c, G^a, L^c, L^a is a non-negative real number, and this explains the appearance of a negative sign in the definition of R_b^c and R_g^a.

3.5 Approximation to Threshold-Based Monitors

A general definition of a threshold-based approach for the monitoring safety properties of CPS was introduced in Section 2.1.12. Let $a_{tr} \in [0, 1)$ be the *acceptance threshold*, then a threshold-based monitor would reject an execution that initially generated an observed finite output sequence α if the following condition holds:

$$AccProb(\alpha) \leq a_{tr} \tag{3.4}$$

Alternative representation is through a rejection probability $RejProb(\alpha)$ compared with a rejection threshold $r_{tr} = 1 - a_{tr}$, such that output sequence is rejected once condition $RejProb(\alpha) \geq r_{tr}$ is satisfied.

There are multiple ways to approximate the value of $AccProb$ by the upper-bound, or $RejProb$ by the lower-bound. One such way was introduced and used in [8–10], and here we extend it to obtain a better estimated value. For any state $s \in S$, let $Pr(s|\alpha)$ represent the probability density function over the system states after the output sequence α is observed. Also, let $Pr(s'|s)$ define the probability that given a current system state s the next state is s', assuming that system continues its normal execution. Let functions $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ represent the approximation to the acceptance and rejection probabilities calculated with the horizon h. The approximation of $AccProb$ and $RejProb$ that was implemented in [8–10] considers value of horizon $h = 1$. Obviously,

$$\begin{aligned} \lim_{h\to\infty} AccProb^h(\alpha) &= AccProb(\alpha) \\ \lim_{h\to\infty} RejProb^h(\alpha) &= RejProb(\alpha) \end{aligned} \tag{3.5}$$

Formally, the functions $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ are defined by limiting the length of a potential future execution after the output sequence α has been observed to the value of h. And, therefore, mathematically are expressed as follows:

$$\begin{aligned}
AccProb^h(\alpha) &= \int_{s_1 \in S_{good}} \int_{s_2 \in S_{good}} \cdots \int_{s_h \in S_{good}} Pr(s_1|\alpha) Pr(s_2, s_1) \ldots Pr(s_h, s_{h-1}) \, \mathrm{d}s_1 \ldots \mathrm{d}s_h \\
RejProb^h(\alpha) &= \int_{s_1 \in S_{bad}} Pr(s_1|\alpha) \mathrm{d}s_1 + \int_{s_1 \in S_{good}} \int_{s_2 \in S_{bad}} Pr(s_1|\alpha) Pr(s_2, s_1) \, \mathrm{d}s_1 \, \mathrm{d}s_2 + \cdots + \\
&+ \int_{s_1 \in S_{good}} \cdots \int_{s_{h-1} \in S_{good}} \int_{s_h \in S_{bad}} Pr(s_1|\alpha) \ldots Pr(s_h, s_{h-1}) \, \mathrm{d}s_1 \ldots \mathrm{d}s_h \\
AccProb^h(\alpha) &+ RejProb^h(\alpha) = 1
\end{aligned} \tag{3.6}$$

Clearly for every finite value of horizon h,

$$\begin{aligned}
AccProb(\alpha) &\leq AccProb^h(\alpha) \\
RejProb(\alpha) &\geq RejProb^h(\alpha)
\end{aligned} \tag{3.7}$$

Given the explicitly defined state transition probability distributions of the monitored system, the values of $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ may be calculated numerically. However, such probability functions may not be available for systems modeled as PHS. And, therefore, calculations might be hard or even not feasible [9]. Alternatively, we may compute $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ by employing Monte-Carlo simulations.

It is easier to see how to apply Monte-Carlo simulations for the approximation of $AccProb^h(\alpha)$. Every simulation starts by sampling a state out of the probability distribution $Pr(s|\alpha)$. Further, the system has to simulate state execution forward $h-1$ times, and a note should be made

whether the final state on the end of the horizon is in S_{good}. Say, a total number of simulations is N, while only N_{good} end up in a good state at the horizon h. Then provided that the number of simulations N is sufficiently large a value of $AccProb^h(\alpha)$ is approximated as:

$$AccProb^h(\alpha) \approxeq \frac{N}{N_{good}} \tag{3.8}$$

Alternative view on the $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ may be derived from the relation with a probability that system is in a bad state now or will be within $h-1$ steps, given the output sequence α. We introduce functions $Bel_{good}(\alpha)$ and $Bel_{bad}(\alpha) = 1 - Bel_{good}(\alpha)$, which represent a belief (total probability) that system is in a bad state assuming that output sequence α has been observed so far. Formally,

$$\begin{aligned} Bel_{good}(\alpha) &= \int\limits_{S_{good}} Pr(s|\alpha)\,\mathrm{d}s \\ Bel_{bad}(\alpha) &= \int\limits_{S_{bad}} Pr(s|\alpha)\,\mathrm{d}s \end{aligned} \tag{3.9}$$

We may see immediately that $AccProb^1(\alpha) = Bel_{good}(\alpha)$ and $RejProb^1(\alpha) = Bel_{bad}(\alpha)$. From the definition of $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ it may be shown that these probability measures are equivalent to the expected probability that a systems execution which generated

the output sequence α will end up to be in a good or bad state respectively. The expectation is computed over all possible extentions of α with additional h outputs. I.e.

$$\begin{aligned} AccProb^h(\alpha) &= \mathbb{E}_{o_1,o_2,\ldots,o_{h-1}\in O}\left[Bel_{good}(\alpha o_1 o_2 \ldots o_{h-1})\right] \\ RejProb^h(\alpha) &= \mathbb{E}_{o_1,o_2,\ldots,o_{h-1}\in O}\left[Bel_{bad}(\alpha o_1 o_2 \ldots o_{h-1})\right] \end{aligned} \tag{3.10}$$

Representation of $AccProb^h(\alpha)$ and $RejProb^h(\alpha)$ in Equation 3.10 shows formally the relation between an approximation of $AccProb(\alpha)$ and $RejProb(\alpha)$ with the belief probability function.

CHAPTER 4

POMDP-BASED MONITORING

(Previously published as Yavolovsky A., Žefran M., Sistla A.P. (2016) Decision-Theoretic Monitoring of Cyber-Physical Systems. In: Falcone Y., Sánchez C. (eds) Runtime Verification. RV 2016. Lecture Notes in Computer Science, vol 10012. Springer, Cham)

4.1 Monitor Design

We define the POMDP-based monitor on the top of the EHMM associated with the product automaton $\mathcal{B} = \mathcal{A} \times \mathcal{P}$, where $\mathcal{A}$ is a system modeled as PHS and $\mathcal{P}$ is a monitored safety property automaton. For the automaton B we construct the corresponding EHMM $\mathcal{H}_\mathcal{B}$, such that $S_{\mathcal{H}_\mathcal{B}}$ is the set of states of $\mathcal{H}_\mathcal{B}$, Σ is the set of outputs, $f(x, y)$ and $g(x, z)$ represent the *next state function* and *output function* respectively. Let $S_{bad} \subset S_{\mathcal{H}_\mathcal{B}}$ represent a set of states of the product automaton such that the component representing the property $\mathcal{P}$ characterizes a failure. Sets S_{bad} and $S_{good} = S_\mathcal{B} \setminus S_{bad}$ represent the sets of good and bad states of the EHMM $\mathcal{H}_\mathcal{B}$ with respect to the property.

We define the set of actions of the POMDP-monitor as $A = \{Alarm, Continue\}$. The set of states S of the POMDP-monitor is a union $\mathcal{S}_{\mathcal{H}_\mathcal{B}} \cup \{s_{terminal}\}$. A special state $s_{terminal}$ represents a condition after the terminal action $Alarm$ is executed.

When the action *Continue* is executed, the transition function T is completely defined by the *next state function* f of the EHMM $\mathcal{H}_{\mathcal{B}}$.

$$T(s, Continue, s') = \begin{cases} f(s, s') & s, s' \in S \setminus \{s_{terminal}\} \\ 0 & s \neq s' \text{ and } s' = s_{terminal} \\ 1 & s = s' = s_{terminal} \end{cases} \tag{4.1}$$

Transition probabilities under the effect of action *Alarm* are defined for $\forall s \in S$ as follows:

$$T(s, Alarm, s') = \begin{cases} 0 & s' \in S \setminus \{s_{terminal}\} \\ 1 & s' = s_{terminal} \end{cases} \tag{4.2}$$

The reward function R of the POMDP is based on the rewards tuple (G^c, G^a, L^c, L^a) as defined in Section 3.4, with additional consideration of the terminal state:

$$R(s, a) = \begin{cases} G^c & s \in S_{good}, a = Continue \\ -L^a & s \in S_{good}, a = Alarm \\ -L^c & s \in S_{bad}, a = Continue \\ G^a & s \in S_{bad}, a = Alarm \\ T^{a,c} = 0 & s = s_{terminal}, \forall a \in A \end{cases} \tag{4.3}$$

An additional parameter $T^{a,c} = 0$ is defined to represent the value of the reward that is given by execution of any action from the terminal state $s_{terminal}$. By assigning a zero value to it we make sure that no rewards are gained after action *Alarm* is executed.

The POMDP-monitor inherits the observation function in the form of the *output function* from the underlying EHMM and, consequently, from the monitored PHS, and extends it to consider the case of the state $s_{terminal}$. To make the POMDP model consistent we may define a new observation $o_{terminal}$, which is deterministically observed while in the state $s_{terminal}$.

4.2 Monitoring Decision Rule

Every history, i.e., a sequence of actions and observations, may be compactly represented in the form of the belief state - a conditional probability function of the current state given a past history that can be computed recursively using Bayes belief propagation [18]. We will represent a belief state of a POMDP-monitor as b_α, where α is a set of perceived output symbols. Note that in POMDP-monitors a history is fully represented by a sequence of perceived observations, while the actions are always *Continue*.

The policy of the POMDP-monitor defines an action that has to be taken for every belief state b_α. The optimal policy corresponds to an action that maximizes the expected utility

gained by the immediate execution and by following the optimal strategy over a future time horizon: $\pi^*(b_\alpha) = \arg\max_{a \in A} Q(b_\alpha, a)$, where according to the Bellman equation [18]

$$\begin{aligned} Q(b_\alpha, \mathit{Alarm}) &= Bel_{bad}(b_\alpha)G^a - Bel_{good}(b_\alpha)L^a \\ Q(b_\alpha, \mathit{Continue}) &= -Bel_{bad}(b_\alpha)L_c + Bel_{good}(b_\alpha)G^c + \\ &+ \gamma \sum_{o \in O} Pr(o|\mathit{Continue}, b_\alpha)V^*(b_{\alpha o}) \\ V^*(b_\alpha) &= \max_{a \in A} Q(b_\alpha, a) \end{aligned} \tag{4.4}$$

In the equations above, $Bel_{bad}(b_\alpha)$ represents the probability that the current state is bad (represents a failure), i.e., $Bel_{bad}(b_\alpha) = \int_{S_{bad}} b_\alpha(s)\,\mathrm{d}s = \int_{S_{bad}} Pr(s|\alpha)\,\mathrm{d}s$ (the integral needs to be understood abstractly in the sense of probability functions as given in Section 2.1.6). Similarly, $Bel_{good}(b_\alpha)$ represents the probability that the current state is good, and $Bel_{bad}(b_\alpha) = 1 - Bel_{good}(b_\alpha)$. $Pr(o|a, b_\alpha)$ is the probability that given the current belief state b_α and action a, the next observation is o (it can be computed by "integrating" over the current belief state and appropriately using the transition and observation functions), and $b_{\alpha o}$ is the belief state in the next time step given we have taken action *Continue* and that the next observed symbol is o (it can be computed using Bayes belief propagation). Note that according to the definition of the action *Alarm*, a value of the expected future reward is equal to 0 so $Q(b_\alpha, \mathit{Alarm})$ misses a

term compared to $Q(b_\alpha, \mathit{Continue})$. Therefore, for the given belief state b_α, the optimal policy π^* will return the action *Alarm* iff the following condition holds:

$$Q(b_\alpha, \mathit{Alarm}) \quad \geq \quad Q(b_\alpha, \mathit{Continue}) \tag{4.5}$$

The value of the discount factor γ has to be selected to obtain the desired property of the policy. Infinite horizon solutions require a value of $\gamma < 1$ to guarantee the convergence of the infinite sum. However, if the value function is calculated for the finite horizon it is common to assume that $\gamma = 1$.

Our focus is mostly concentrated on POMDP-based monitors with finite horizon. Mainly, this is because of our interest in the online policy calculation during the system execution. Due to the computational and space complexity that might be required in order to obtain reasonably good convergence rate of the POMDPs with an infinite horizon we omit this class of decision

rules. Therefore, we present a POMDP-based decision rule for the case of finite horizon $h \geq 1$ with discount factor $\gamma = 1$ as follows:

$$\begin{aligned}
Q^h(b_\alpha, Alarm) &\geq Q^h(b_\alpha, Continue) \\
Q^h(b_\alpha, Alarm) &= G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\
Q^h(b_\alpha, Continue) &= -L^c + (G^c + L^c)Bel_{good}(b_\alpha) + \\
&+ \sum_{o \in O} Pr(o|Continue, b_\alpha)V^{h-1}(b_{\alpha o}) \\
V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned} \tag{4.6}$$

4.3 POMDP-Monitor Policy

While many POMDP solvers assume that POMDP models are fully defined, i.e., the transition and observation functions are given, it is not obvious how to define them for systems modeled as PHS. Instead, defining a black-box simulator is straightforward. For a PHS it can be constructed by implementing the difference equations for each mode and transitions for the hybrid mode switching. Such black-box simulator is able to produce a sample of the next state and the observation, given a current state.

POMDPs with continuous state space are even more complex. In order to handle continuous spaces some algorithms have been developed that employ Monte-Carlo simulations [48, 49]. Considering the lack of completely defined POMDP model but the existence of the black-box simulator, we focused on Partially Observable Monte-Carlo Planning (POMCP) to implement

the POMDP-monitor. The traditional POMCP is defined for discrete state spaces, however the algorithm can be easily extended and applied to the continuous case, although the observation space has to be kept discrete. Here we define a Monitoring-POMCP, which is an adaptation of the POMCP for the POMDP-monitors.

Similarly to the traditional POMCP [49], the Monitoring-POMCP is an online POMDP planner. Rather than requiring analytically defined probability distributions it is designed to work with the black-box instantiation of the model. The generative model that is hidden within the black-box is able to produce a sample of the future state s_{t+1}, observation o_{t+1} and reward r_{t+1}, given the pair (s_t, a_t) of the current state and an action.

POMCP overcomes both curses that make exact solutions so hard to apply for the large problems. The value function is estimated by applying Monte-Carlo simulations with the black-box model. Therefore, POMCP has a sample complexity that is determined by the complexity of the underlying POMDP model, and not by the size of the state and observation spaces.

POMCP may be described similarly to any POMDP planner and consists of the following basic steps: (1) update the belief state b_t to obtain b_{t+1} considering the new observation o_{t+1} and the most recent action a_t, (2) for the new belief state b_{t+1} find an action $a_{t+1} \in A$ that should be executed. In POMCP both steps share the same Monte-Carlo simulation to propagate the belief state from b_t to b_{t+1}. This belief propagation step may be performed efficiently by a particle filter and Monte-Carlo simulations even for continuous state space. Provided that there are sufficiently many particles, the approximation of the belief state will be close to the true distribution.

The decision step of the POMCP is based on the *Monte-Carlo Tree Search* adapted for the belief state search space [49]. The root tree node corresponds to the current belief state of the POMDP, and every other node represents a history h of actions and observations. Each tree node has an associated pair $\big(N(h), V(h)\big)$, where $N(h)$ is the number of times the node has been visited during the search, and $V(h)$ is the mean return of all simulations started from the node.

In order to limit the size of the tree and compute a good approximation of the optimal policy, it is necessary to require a finite number of actions and observations. The POMDP-monitor only has two actions, and we have assumed that the continuous observation space is quantized and represented with a finite set. Note, that in practice sensors typically use an analog-to-digital converter to produce the output, which means that the observation measurements are in fact already quantized.

The search tree of the Monitoring-POMCP is constructed sequentially with a number of Monte-Carlo simulations starting from a state sampled from the belief state in the root node. Every simulation represents a sequence of actions and observations. While the observations are produced by a black-box simulation, the action at each simulation step is selected either by a *tree policy* or by a *rollout policy* (see Figure 2).

For a history node that already has at least one action leaf node the *tree policy* is used. For the Monitoring-POMCP, we use the UCB1 (Upper Confidence Bounds) [50] algorithm. UCB1 selects an action that maximizes the value of the node augmented by the exploration bonus: $V^{\oplus}(ha) = V(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$. The scalar value c determines the relative ratio between the

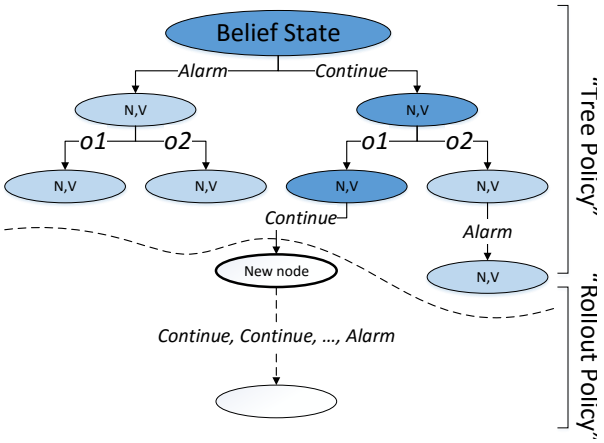

Figure 2. POMCP search tree.

exploration and exploitation. The value of unexplored actions is always set to ∞ so that each action is selected for exploration at least once. The Monitoring-POMCP uses the exploration constant $c = R_{hi} - R_{lo}$ [49], where R_{hi} is the largest value achieved during sample runs of the POMDP with the constant $c = 0$, and R_{lo} is the smallest value returned during sample rollouts. In the context of the Monitoring-POMDP, $R_{hi} = \max(G^c, G^a)$, and $R_{lo} = \min(-L^c, -L^a)$.

For the case of the history node that has no action leaf nodes yet the *rollout policy* is used. In rollout, the execution proceeds up to the end of the fixed horizon. The simplest form of the *rollout policy* is a uniform random policy. However, it is not suitable for the POMDP-based monitor. To see that, consider the following scenario. Assume that at some point during the construction of the search tree, the simulation is at a node with the history h that does not have any leaf nodes so that the *rollout policy* is used. Let's also assume that when this node was reached during the simulation, the system state corresponded to a failure. The random *rollout policy* will generate a finite randomly sampled sequence of actions. The sequence will be

stopped either when the maximum horizon depth is reached, or a terminal action is executed. Let's assume that in the generated sequence of actions the first action is *Continue*, followed again by some number of *Continue* actions and eventually executing an *Alarm*. Such an action sequence would accumulate a significant penalty for a missed alarm, and this penalty would be associated with the new leaf node added to the tree. Now, let's assume that when this search tree node is encountered again in the search, the system state is good, i.e. there is no failure. According to the *tree policy*, at first the *Alarm* branch will be explored, but at further times it will be unlikely that *Continue* branch will be explored again. This might have a significant effect on the value of the expected reward for the action *Continue* at the root node of the tree.

Instead of selecting the action randomly during the *rollout policy* we propose to select the action to maximize the reward at every step. This can be achieved by raising the *Alarm* only at the failure state, and continuing the execution if the system state is good. In this way it is guaranteed that the value assigned to the newly added node will promote further exploration when the *tree policy* is used.

The outcome of the search is an action that produces the largest augmented reward from the root node after the predefined number of simulations have been performed.

4.4 Monitoring Autonomous Systems

In the discussion of threshold-based monitors in Section 2.1.12, and POMDP-based monitors in Section 4.1 we have never yet mentioned whether the monitored system is assumed to be completely autonomous or its dynamics is affected by the external control input. In fact, this

is a very important question, since some monitors might be not applicable or rather complex for the class of externally manipulated systems.

The approximation to threshold-based monitors, as implemented in [8], only considers current belief state when decision is made. Therefore, all the inputs that were seen till now are sufficient to make a decision. However, if there is a need to implement a better approximation by considering the expected effect of future state evolutions then external control input either needs to be given or modeled.

Similarly, in POMDPs decision is made based on the calculation of the expected value function for each action. By observing the decision rule given in Equation 4.6 we may see that both $Pr(o|Continue, b_\alpha)$ and $b_{\alpha o}$ require knowledge of control input to be appropriately computed. In our work we assume that external control input is either deterministically known or is estimated, and is represented as a function of time.

CHAPTER 5

PARAMETRIZATION AND PERFORMANCE OF POMDP-BASED MONITORS

(Partially previously published as Yavolovsky A., Žefran M., Sistla A.P. (2016) Decision-Theoretic Monitoring of Cyber-Physical Systems. In: Falcone Y., Sánchez C. (eds) Runtime Verification. RV 2016. Lecture Notes in Computer Science, vol 10012. Springer, Cham)

5.1 Parametrization of POMDP-Based Monitor

We have identified in Section 4.1 the formal structure of the class of POMDP-based monitors. Most of the components of the POMDP are defined by a monitored system and a property, and, therefore, should not be considered as parameters of the monitor. Obviously, a set of states and observations, transition and observation functions would be shared across different monitoring techniques. Among all, we outline the POMDP-based monitor reward function R as a key control, which may result in a different monitoring performance and needs to be tuned. Recall, that reward function R is based on the assignments of non-negative values to every component of the rewards tuple $(G^c, G^a, L^c, L^a, T^{a,c})$, where $T^{a,c} = 0$.

Dependence of the decision on the reward function may be easily seen by observing the decision rule of the POMDP-based monitors initially given in Equation 4.6, which we present again for convenience:

$$\begin{aligned}
Q^h(b_\alpha, Alarm) &\geq Q^h(b_\alpha, Continue) \\
Q^h(b_\alpha, Alarm) &= G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\
Q^h(b_\alpha, Continue) &= -L^c + (G^c + L^c)Bel_{good}(b_\alpha) + \\
&+ \sum_{o \in O} Pr(o|Continue, b_\alpha)V^{h-1}(b_{\alpha o}) \\
V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned} \tag{5.1}$$

Another parameter that affects the decision rule and is not related to the system is a depth of horizon h. Unlike the rewards, the value of horizon only affects the expected reward for the action *Continue*. In our work, we have studied the effect of the reward function on the monitoring performance only for fixed values of horizon $h = 1$ and $h = 2$. The case of horizon $h = 1$ is special since POMDP value functions become fully defined by the immediate reward values. We study the case of horizon $h = 2$ to investigate the effect of the non-linear operator *max* involved in the value function computations. We expect that similar analysis may be further used for any arbitrary finite horizon values.

Every possible assignment of values in rewards tuple identifies the monitoring decision rule. Further, every decision rule results in different accuracy measures and monitoring time as defined in Sections 2.1.9 and 2.1.10. Given that every reward value may be tuned independently

we may immediately claim that a class of POMDP-based monitors has four degrees of freedom. This statement, although correct, is not a strong statement as it does not consider those combinations of rewards that define monitors with identical performance.

5.2 Equivalent POMDP-Based Monitors

Some classes of equivalent POMDP-based monitors may be identified by considering properties of the value function of a general POMDP. It may be shown that an optimal policy of any POMDP is invariant with respect to the addition of a constant real-valued number to a reward function (see Appendix A). It may be also shown that an optimal policy of any POMDP is invariant with respect to the multiplication of a positive real-valued number with a reward function (see Appendix B).

In POMDP-based monitors, the reward function is completely defined by the assignment of values to the tuple $(G^c, G^a, L^c, L^a, T^{a,c})$. According to Appendix A, if the rewards tuple is modified to $(G^c + k, G^a + k, L^c + k, L^a + k, T^{a,c} + k)$, where $k \in \mathbb{R}$, then the policy of the POMDP-based monitor that implements a new reward function will be unchanged. However, this operation does not reduce the complexity of the reward function. In the best case, only one component of the tuple will be equal to 0. And, according to POMDP-based monitor design, the value of $T^{a,c}$ is already fixed to be 0.

According to Appendix B, if we update the rewards tuple as $(kG^c, kG^a, kL^c, kL^a, kT^{a,c})$, where $k \in \mathbb{R}_{>0}$, then the policy of the POMDP-based monitor that implements a new reward function will be also unchanged. Since $T^{a,c} = 0$, the value of $kT^{a,c}$ is also zero. Let

$\pi^*_{(G^c,G^a,L^c,L^a,T^{a,c})}$ be the optimal policy of a POMDP-based monitor implemented with rewards tuple $(G^c, G^a, L^c, L^a, T^{a,c})$. Then by appropriate selection of a constant k we obtain:

$$\pi^*_{(G^c,G^a,L^c,L^a,T^{a,c}=0)} = \begin{cases} \pi^*_{(1,\frac{G^a}{G^c},\frac{L^c}{G^c},\frac{L^a}{G^c},\frac{T^{a,c}}{G^c}=0)} & k=\frac{1}{G^c}\ ,\ G^c\neq 0 \\ \pi^*_{(\frac{G^c}{G^a},1,\frac{L^c}{G^a},\frac{L^a}{G^a},\frac{T^{a,c}}{G^a}=0)} & k=\frac{1}{G^a}\ ,\ G^a\neq 0 \\ \pi^*_{(\frac{G^c}{L^c},\frac{G^a}{L^c},1,\frac{L^a}{L^c},\frac{T^{a,c}}{L^c}=0)} & k=\frac{1}{L^c}\ ,\ L^c\neq 0 \\ \pi^*_{(\frac{G^c}{L^a},\frac{G^a}{L^a},\frac{L^c}{L^a},1,\frac{T^{a,c}}{L^a}=0)} & k=\frac{1}{L^a}\ ,\ L^a\neq 0 \end{cases} \tag{5.2}$$

Therefore, under the assumption that one of the components of a rewards tuple is non-zero we may safely assume without loss of generality that it is equal to 1 and other rewards are adjusted accordingly. Given a POMDP-based monitor with a fixed rewards tuple we may convert it to an equivalent POMDP-based monitor with at least one positive component equal to 1. Therefore, we may claim that in POMDP-based monitors there are exactly 3 free parameters, i.e. such class of monitors has 3 degrees of freedom.

In our work, we mostly assume that if $G^c \neq 0$, then $G^c = 1$. This assumption does not make our further study less general, while it simplifies the decision rule and aids in certain conclusions.

5.3 Simplest Reward Functions and Monitor Performance

The complexity of POMDP value function computations and, hence, its effect on the monitoring performance measures is determined by parameters in the rewards tuple $(G^c, G^a, L^c, L^a, T^{a,c} = 0)$. We have shown in Section 5.2 that at least one reward value can be safely assumed to be

equal to 1. In this section, we assign some of the rewards to 0 and analyze the performance of resulting monitors.

5.3.1 Reward Functions with Single Non-Zero Reward Parameter

5.3.1.1 Case 1: G^c is the Only Non-Zero Parameter

For this case, we consider $G_c > 0$, while $G^a = L^c = L^a = 0$. We rewrite POMDP value function from Equation 4.6 as follows:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= 0 \\ Q^h(b_\alpha, Continue) &= G^c Bel_{good}(b_\alpha) + \\ &+ \sum_{o \in O} Pr(o|Continue, b_\alpha) V^{h-1}(b_{\alpha o}) \\ V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \end{aligned} \tag{5.3}$$

Given that $G^c > 0$, the value of $V^h(b_\alpha) \geq 0$ at every horizon $h > 1$, and thus, the expected utility of the action *Continue* is $Q^h(b_\alpha, Continue) \geq 0$. Note, that $Q^h(b_\alpha, Continue) = 0$ only when $Bel_{good}(b_\alpha) = 0$. Therefore, $Q^h(b_\alpha, Continue) \geq Q^h(b_\alpha, Alarm)$, which results in no raised *Alarms*, and, therefore, $AA = 1$, $RA = 0$ and $MTIME = \infty$. This performance outcomes are identical to the threshold-based monitor with rejection threshold equal to 1.

5.3.1.2 Case 2: G^a is the Only Non-Zero Parameter

For this case, we consider $G_a > 0$, while $G^c = L^c = L^a = 0$. Therefore, we may rewrite the value function from Equation 4.6 for this case as follows:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= G^a - G^a Bel_{good}(b_\alpha) = G^a Bel_{bad}(b_\alpha) \\ Q^h(b_\alpha, Continue) &= \sum_{o \in O} Pr(o|Continue, b_\alpha) V^{h-1}(b_{\alpha o}) \\ V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \end{aligned} \tag{5.4}$$

To analyze the relationship between $Q^h(b_\alpha, Alarm)$ and $Q^h(b_\alpha, Continue)$ we expand the function $V^h(b_\alpha)$ for different values of horizon h. As a result, the following expressions are obtained:

$$
\begin{aligned}
V^1(b_\alpha) &= \max \begin{cases} G^a Bel_{bad}(b_\alpha) \\ \underbrace{\sum_{o \in O} Pr(o|Continue, b_\alpha) V^0(b_{\alpha o})}_{=0} \end{cases} = \\
&= G^a Bel_{bad}(b_\alpha) \\
V^2(b_\alpha) &= \max \begin{cases} G^a Bel_{bad}(b_\alpha) \\ \sum_{o \in O} Pr(o|Continue, b_\alpha) V^1(b_{\alpha o}) \end{cases} = \\
&= \max \begin{cases} G^a Bel_{bad}(b_\alpha) \\ G^a \sum_{o \in O} Pr(o|Continue, b_\alpha) Bel_{bad}(b_{\alpha o}) \end{cases} = \\
&= G^a \sum_{o \in O} Pr(o|Continue, b_\alpha) Bel_{bad}(b_{\alpha o}) \\
V^h(b_\alpha) &= \max \begin{cases} G^a Bel_{bad}(b_\alpha) \\ G^a \sum_{o_1 \in O} Pr(o_1|Continue, b_\alpha) \cdots \sum_{o_h \in O} Pr(o_h|Continue, b_{\alpha o_1 \ldots o_h}) Bel_{bad}(b_{\alpha o_1 \ldots o_h}) \end{cases} = \\
&= G^a \sum_{o_1 \in O} Pr(o_1|Continue, b_\alpha) \cdots \sum_{o_h \in O} Pr(o_h|Continue, b_{\alpha o_1 \ldots o_h}) Bel_{bad}(b_{\alpha o_1 \ldots o_h})
\end{aligned}
\tag{5.5}
$$

Thus, we observe that for every belief state b_α the POMDP-based monitor will make a choice to *Continue*. Therefore, no *Alarms* will be raised, and performance measures are $AA = 1$,

$RA = 0$ and $MTIME = \infty$. This makes that POMDP-based monitor to be equivalent to the threshold-based monitor with a threshold 1.

5.3.1.3 Case 3: L^c is the Only Non-Zero Parameter

For this case, we consider $L_c > 0$, while $G^c = G^a = L^a = 0$. Therefore, we may rewrite the value function from Equation 4.6 for this case as follows:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= 0 \\ Q^h(b_\alpha, Continue) &= -L^c Bel_{bad}(b_\alpha) + \\ &+ \sum_{o \in O} Pr(o|Continue, b_\alpha) V^{h-1}(b_{\alpha o}) \\ V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \end{aligned} \tag{5.6}$$

Given that $L_c > 0$, the optimal value function $V_h(b_\alpha) = 0$, since the utility of the action *Continue* is $Q^h(b_\alpha, Continue) \leq 0$ for every belief state b_α. Note, that $Q^h(b_\alpha, Continue) = 0$ only when $Bel_{bad}(b_\alpha) = 0$. This makes the POMDP-based monitor equivalent to the threshold-based monitor with rejection threshold 0, and thus, performance measures are $AA = 0$, $RA = 1$ and $MTIME = -\infty$.

5.3.1.4 Case 4: L^a is the Only Non-Zero Parameter

For this case, we consider $L_a > 0$, while $G^c = G^a = L^c = 0$. Therefore, we may rewrite the value function from Equation 4.6 for this case as follows:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= -L^a Bel_{good}(b_\alpha) \\ Q^h(b_\alpha, Continue) &= \sum_{o \in O} Pr(o|Continue, b_\alpha) V^{h-1}(b_{\alpha o}) \\ V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \end{aligned} \tag{5.7}$$

Given that $L_a > 0$, the optimal value function $V^h(b_\alpha) = 0$, since the utility of the action *Continue* is $Q^h(b_\alpha, Continue) = 0$ and $Q^h(b_\alpha, Alarm) \leq 0$ for every belief state b_α. Note, that $Q^h(b_\alpha, Alarm) = 0$ only when $Bel_{good}(b_\alpha) = 0$. This makes the POMDP-based monitor equivalent to the threshold-based monitor with rejection threshold 0, i.e. every execution is immediately rejected. Thus, performance measures are $AA = 0$, $RA = 1$ and $MTIME = -\infty$.

5.3.2 Reward Functions with Multiple Non-Zero Reward Parameters

In this section, we consider those assignments of rewards which have two non-zero parameters. Among all of these assignments we have identified that only one produces a trivial POMDP-based monitor. In that assignment we consider the case of $G^c > 0$ and $L^a > 0$, while

$G^a = L^c = 0$. By rewriting the value function from Equation 4.6 for the actions *Alarm* and *Continue* we obtain:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= -L^a Bel_{good}(b_\alpha) \\ Q^h(b_\alpha, Continue) &= G^c Bel_{good}(b_\alpha) + \sum_{o \in O} Pr(o|Continue, b_\alpha) V^{h-1}(b_{\alpha o}) \\ V^h(b_\alpha) &= \begin{cases} \max_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \end{aligned} \tag{5.8}$$

Observe, that both $V^h(b_\alpha)$ and $Q^h(b_\alpha, Continue)$ have non-negative values. This is true, since the value of $Q^h(b_\alpha, Alarm) \leq 0$, and $Q^h(b_\alpha, Continue)$ is represented as a summation of non-negative terms. Thefore, decision to raise an *Alarm* may be only made when both $Q^h(b_\alpha, Continue)$ and $Q^h(b_\alpha, Alarm)$ are equal to 0. This happens only when the value of $Bel_{good}(b_\alpha) = 0$. This makes considered POMDP-based monitor equivalent to the threshold based monitor with rejection threshold 1. Thus, performance measures of such monitor is $AA = 1$, $RA = 0$ and $MTIME = \infty$.

Every other assignment of values to rewards tuple, such that exactly two parameters are non-zero does not make a POMDP-based monitor trivial. We make this conclusion based on the observation that the sign of value function for either action depends on the value of $Bel_{good}(b_\alpha)$ for all considered depths of the horizon. We have also observed the similar issue for those assignments of rewards that have only one parameter equal to 0.

5.4 Performance of POMDP-Based Monitors

In our work we are interested in studying performance measures of various POMDP-based monitors, i.e. show how AA, RA, and $MTIME$ are affected by the assignment of values to rewards tuple the depth of horizon.

The class of *conservative monitors* is defined in a way to guarantee that every bad execution is eventually rejected. In other words, every bad execution is rejected with probability 1 by a conservative monitor. Threshold-based monitors with a rejection threshold may be easily shown to be conservative monitors. Similarly, every non-trivial configuration of rewards tuple makes the corresponding POMDP-based monitor a *conservative monitor*, which we prove in the following lemma.

Lemma 1. *Every non-trivial POMDP-based monitor is a conservative monitor and, therefore, has* $RA = 1$.

Proof. In order to provide the proof, we start, by considering the fact that threshold-based monitors are *conservative monitors*. Let α be the infinite output sequence that is perceived from the system, and assume that α is generated by a bad execution. The value of $RejProb(\alpha) = 1$ and $AccProb(\alpha) = 0$. Let b_α represent the belief state that corresponds to the perceived output sequence α. Then it is easy to see that $Bel_{good}(b_\alpha) = 0$.

Let's proceed by rewriting the expressions for the value functions of the POMDP-based monitor for the belief state b_α:

$$\begin{aligned} Q^h(b_\alpha, Alarm) &= G^a \\ Q^h(b_\alpha, Continue) &= -L^c + \sum_{o \in O} Pr\big(o|Continue, b_\alpha\big) V^{h-1}(b_{\alpha o}) \end{aligned} \tag{5.9}$$

The optimal value function is calculated recursively as follows:

$$\begin{aligned} V^0(b_\alpha) &= 0 \\ V^1(b_\alpha) &= \max \begin{cases} G^a & \\ -L^c + \sum_{o \in O} Pr(o|Continue, b_\alpha) \underbrace{V^0(b_{\alpha o})}_{=0} & \end{cases} = G^a \\ V^2(b_\alpha) &= \max \begin{cases} G^a & \\ -L^c + \sum_{o \in O} Pr(o|Continue, b_\alpha) \underbrace{V^1(b_{\alpha o})}_{=G^a} & \end{cases} = G^a \\ &\dots \\ V^h(b_\alpha) &= G^a \end{aligned} \tag{5.10}$$

From this we directly see that $Q^h(b_\alpha, Continue) = -L^c + G^a$ for $\forall h > 1$, and is $Q^h(b_\alpha, Continue) = -L^c$ when $h = 1$. Therefore, $\pi^*(b_\alpha) = \arg\max_{a \in A} Q^h\big(b_\alpha, a\big) = Alarm$ for $\forall h \geq 1$. This confirms that the POMDP-based monitor is guaranteed to eventually reject the output sequence generated by the bad execution with probability 1, i.e. is a *conservative monitor*.

□

In all of our future discussion, we will only be interested in obtaining AA and $MTIME$, while we always consider $RA = 1$ for the POMDP-based monitors. However, before we proceed we formalize the criteria that we use to compare a pair of monitors.

Definition 1. *A monitor $M+$ with acceptance accuracy AA_{M+}, rejection accuracy RA_{M+} and monitoring time $MTIME_{M+}$ is considered to be better than a monitor $M-$, with performance measures AA_{M-}, RA_{M-} and $MTIME_{M-}$ iff the following condition holds:*

$$\begin{aligned}
&\Big(AA_{M+} > AA_{M-} \quad \wedge \quad RA_{M+} \geq RA_{M-} \quad \wedge \quad MTIME_{M+} \leq MTIME_{M-}\Big) \vee \\
&\Big(AA_{M+} \geq AA_{M-} \quad \wedge \quad RA_{M+} > RA_{M-} \quad \wedge \quad MTIME_{M+} \leq MTIME_{M-}\Big) \vee \\
&\Big(AA_{M+} \geq AA_{M-} \quad \wedge \quad RA_{M+} \geq RA_{M-} \quad \wedge \quad MTIME_{M+} < MTIME_{M-}\Big)
\end{aligned} \tag{5.11}$$

Essentially, the monitor $M+$ is better than the monitor $M-$ if at least one of the performance measures is improved, while others are at least not changed. Note, that improvement in AA and RA means increased value, while improvement in $MTIME$ means decreased value.

In our study we take a class of threshold-based monitors as a baseline, and we are looking for the configuration of a POMDP-based monitor that may perform better than a baseline. For simplicity, we consider an approximation of the threshold-based monitors introduced in Section 2.1.12, which only considers the probability that current state of the system violates the property. Similarly, we only consider small horizon values for a POMDP-based monitor, and only limit our study to the case of horizon 1 and 2.

5.4.1 POMDP-Based Monitor with Horizon 1

Let $POMDP_1$ represent a POMDP-based monitor as defined in Section 4.1 with horizon $h = 1$. Then the value functions for each action and an optimal value function is defined as follows:

$$\begin{aligned} Q^1(b_\alpha, Alarm) &= G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\ Q^1(b_\alpha, Continue) &= -L^c + (G^c + L^c)Bel_{good}(b_\alpha) \\ V^1(b_\alpha) &= \max \begin{cases} G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\ -L^c + (G^c + L^c)Bel_{good}(b_\alpha) \end{cases} \end{aligned} \tag{5.12}$$

According to Equation 5.12 the optimal policy of the $POMDP_1$ is choosing the action $Alarm$, i.e. defines a rejection condition, if the following condition holds:

$$\begin{aligned} Q^1(b_\alpha, Alarm) &\geq Q^1(b_\alpha, Continue) \\ G^a - (L^a + G^a)Bel_{good}(b_\alpha) &\geq -L^c + (G^c + L^c)Bel_{good}(b_\alpha) \\ Bel_{good}(b_\alpha) &\leq \frac{G^a + L^c}{G^a + L^a + G^c + L^c} \end{aligned} \tag{5.13}$$

Remember, that $Bel_{good}(b_\alpha) = AccProb^1(\alpha)$, and hence, we may claim that for a fixed assignment of rewards tuple, there exists a corresponding value of acceptance threshold a_{tr}, such that monitor $POMDP_1$ is equivalent to the threshold-based monitor with horizon 1

configured to reject based on that acceptance threshold. I.e. the equivalent threshold-based monitor rejects output sequences α according to the condition

$$AccProb^1(\alpha) \quad \leq \quad a_{tr} = \frac{G^a + L^c}{G^a + L^a + G^c + L^c} \tag{5.14}$$

In a similar way it is possible to derive an equation to compute the value of rejection threshold r_{tr} for an equivalent threshold-based monitor that operates based on the value of $RejProb^1(\alpha)$.

Proposition 1. *Every POMDP-based monitor with horizon $h = 1$ is equivalent to a single threshold-based monitor with horizon 1.*

Proof. For every rewards tuple we may compute a value of corresponding threshold as shown in Equation 5.14. □

Note, that the opposite statement is not true, as for a single threshold-based monitor we may identify infinite number of reward tuples that will define an equivalent POMDP-based monitor.

5.4.2 POMDP-Based Monitor with Horizon 2

Let $POMDP_2$ represent a POMDP-based monitor as defined in Section 4.1 with horizon $h = 2$. Then value functions for each action are as follows:

$$\begin{aligned} Q^2(b_\alpha, Alarm) &= G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\ Q^2(b_\alpha, Continue) &= -L^c + (G^c + L^c)Bel_{good}(b_\alpha) + \\ &+ \sum_{o \in O} Pr(o|Continue, b_\alpha)V^1(b_{\alpha o}) \\ V^1(b_\alpha) &= \max \begin{cases} G^a - (L^a + G^a)Bel_{good}(b_\alpha) \\ -L^c + (G^c + L^c)Bel_{good}(b_\alpha) \end{cases} \end{aligned} \tag{5.15}$$

According to Equation 5.15 the optimal policy of the $POMDP_2$ is choosing the action $Alarm$, if the following condition holds:

$$\begin{aligned} Q^2(b_\alpha, Alarm) &\geq Q^2(b_\alpha, Continue) \\ G^a - (L^a + G^a)Bel_{good}(b_\alpha) &\geq -L^c + (G^c + L^c)Bel_{good}(b_\alpha) + \\ &+ \sum_{o \in O} Pr(o|Continue, b_\alpha)V^1(b_{\alpha o}) \end{aligned} \tag{5.16}$$

As before, we are interested to compare the performance of the POMDP-based monitor with a threshold-based monitor, specifically a threshold-based monitor with horizon 1. In Section 5.4.1 we found a mapping from the POMDP-based monitor with horizon 1 to the equivalent threshold-based monitor with horizon 1 . Therefore, rather then comparing $POMDP_2$ with a threshold-based monitor, we may compare it with $POMDP_1$, assuming that rewards tuple has been fixed.

Let $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ represent the expected cost of an optimal action after some output o follows α. Essentially,

$$\mathbb{E}_{o\in O}\, V^1(b_{\alpha o}) = \sum_{o\in O} Pr(o|Continue, b_\alpha) V^1(b_{\alpha o}) \tag{5.17}$$

Obviously, the main difference in the decision rule of $POMDP_2$ and $POMDP_1$ is in the value function for the action *Continue*, which includes the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$. Thus, we need to look thoroughly on the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$, and the effect it might have on the AA and $MTIME$.

5.4.2.1 Lower and Upper bound for the Expected Reward

In this section we define the bounds for the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ as defined in the Equation 5.17. First, let's expand all the parts of expression.

$$\mathbb{E}_{o\in O}\, V^1(b_{\alpha o}) = \sum_{o\in O} Pr(o|Continue, b_\alpha) \max \begin{cases} G^a - (L^a + G^a) Bel_{good}(b_{\alpha o}) \\ -L^c + (G^c + L^c) Bel_{good}(b_{\alpha o}) \end{cases} \tag{5.18}$$

One way to define the lower bound for the $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ is to push the operator max out of the sum over all the observations. Clearly,

$$\max \sum_{o \in O} Pr(o|Continue, b_\alpha) \begin{cases} G^a - (L^a + G^a) Bel_{good}(b_{\alpha o}) \\ \\ -L^c + (G^c + L^c) Bel_{good}(b_{\alpha o}) \end{cases} \leq \mathbb{E}_{o \in O} V^1(b_{\alpha o}) \tag{5.19}$$

By further manipulations with the expression we obtain:

$$\begin{aligned} \max \sum_{o \in O} Pr(o|Continue, b_\alpha) & \begin{cases} G^a - (L^a + G^a) Bel_{good}(b_{\alpha o}) \\ \\ -L^c + (G^c + L^c) Bel_{good}(b_{\alpha o}) \end{cases} = \\ \max & \begin{cases} G^a - (L^a + G^a) \sum_{o \in O} Pr(o|Continue, b_\alpha) Bel_{good}(b_{\alpha o}) \\ \\ -L^c + (G^c + L^c) \sum_{o \in O} Pr(o|Continue, b_\alpha) Bel_{good}(b_{\alpha o}) \end{cases} = \\ \max & \begin{cases} G^a - (L^a + G^a) AccProb^2(b_\alpha) \\ \\ -L^c + (G^c + L^c) AccProb^2(b_\alpha) \end{cases} \leq \mathbb{E}_{o \in O} V^1(b_{\alpha o}) \end{aligned} \tag{5.20}$$

In order to obtain the upper bound we may need to start from the general expression for the expected reward function $\mathbb{E}_{o \in O} V^1(b_{\alpha o})$. First, remember that the belief update state is performed according to the following equation:

$$Pr(s'|\alpha o) = \frac{Z(s', Continue, o) \left[\sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) \right]}{Pr(o|Continue, b_\alpha)} \tag{5.21}$$

Considering this, we rewrite the equation for the expected reward function $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ in terms of functions given directly by the POMDP framework:

$$\begin{aligned}\mathbb{E}_{o\in O}\, V^1(b_{\alpha o}) &= \sum_{o\in O} Pr(o|Continue, b_\alpha) \max_{a\in A}\left[\sum_{s\in S} Pr(s|\alpha o)R(s,a)\right] \\ &= \sum_{o\in O} \max_{a\in A}\left[\sum_{s'\in S} Z(s', Continue, o) \sum_{s\in S} Pr(s|s', Continue)Pr(s|\alpha)R(s',a)\right]\end{aligned} \tag{5.22}$$

Considering the equation Equation 5.22 a lower bound may be defined by pushing operator *max* out of the $\sum_{o\in O}$. In fact, we will obtain exactly the same lower bound as we have already

defined in Equation 5.20. For the upper bound we may push the operator *max* in the $\sum_{s' \in S}$. Thus,

$$
\begin{aligned}
\mathbb{E}_{o \in O} V^1(b_{\alpha o}) &\leq \sum_{o \in O} \sum_{s' \in S} \max_{a \in A} \left[Z(s', Continue, o) \sum_{s \in S} Pr(s|s', Continue) Pr(s|alpha) R(s', a) \right] = \\
&= \sum_{s' \in S} \max_{a \in A} \left[\sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) R(s', a) \right] \underbrace{\sum_{o \in O} Z(s', Continue, o)}_{=1} = \\
&= \sum_{s' \in S_{good}} \max \begin{cases} -\sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) L^a \\ \sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) G^c \end{cases} + \\
&+ \sum_{s' \in S_{bad}} \max \begin{cases} \sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) G^a \\ -\sum_{s \in S} Pr(s|s', Continue) Pr(s|\alpha) L^c \end{cases} = \\
&= G^c \sum_{s \in S} Pr(s|\alpha) \sum_{s' \in S_{good}} Pr(s|s', Continue) + \\
&+ G^a \underbrace{\sum_{s \in S} Pr(s|\alpha) \sum_{s' \in S_{bad}} Pr(s|s', Continue)}_{1 - \sum_{s \in S} Pr(s|\alpha) \sum_{s' \in S_{good}} Pr(s|s', Continue)} = \\
&= G^a + (G^c - G^a) \sum_{s \in S} Pr(s|\alpha) \sum_{s' \in S_{good}} Pr(s|s', Continue) = \\
&= G^a + (G^c - G^a) AccProb^2(\alpha)
\end{aligned}
\tag{5.23}
$$

Therefore, the following bounds for the expected reward value hold:

$$\max \begin{cases} G^a - (L^a + G^a)AccProb^2(\alpha) \\ -L^c + (G^c + L^c)AccProb^2(\alpha) \end{cases} \leq \mathbb{E}_{o \in O}\, V^1(b_{\alpha o}) \leq G^a + (G^c - G^a)AccProb^2(\alpha) \tag{5.24}$$

5.4.2.2 Reward Configurations Deteriorating Monitoring Performance.

Given the bounds for the expected reward $\mathbb{E}_{o \in O}\, V^1(b_{\alpha o})$ we may identify certain cases when the monitor $POMDP_2$ is provably not better than $POMDP_1$.

First, let's start to analyze the AA of $POMDP_2$ compared with $POMDP_1$. In order for the monitor $POMDP_2$ to increase the value of AA there should be fewer false alarm. This means that at least for some of good executions that generated output sequence α the value of $\mathbb{E}_{o \in O}\, V^1(b_{\alpha o})$ has to be positive although $AccProb^1(\alpha) \leq a_{tr}$ as given in Equation 5.14. Clearly, if the value of $\mathbb{E}_{o \in O}\, V^1(b_{\alpha o})$ is never positive, then AA of the monitor $POMDP_2$ is never better than of monitor $POMDP_1$. In fact, there will be more false alarms, since negative value of additional term in the value function of action Continue makes it easier for the monitor to satisfy a rejection condition.

By applying Equation 5.24 we may state that $\mathbb{E}_{o \in O}\, V^1(b_{\alpha o}) \leq 0$ when the upper bound for it is ≤ 0. Hence,

$$G^a + (G^c - G^a)AccProb^2(\alpha) \leq 0 \tag{5.25}$$

We are interested in those cases when the condition in Equation 5.25 holds for any value of $AccProb^2(\alpha) \in [0, 1]$. It can be shown that this condition holds only if both G^a and G^c

are equal to 0. In fact, by observing the exact expression for $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ in Equation 5.17 we may see that equality to 0 may occur only in case if $Bel_{good}(\alpha) = 1$. Considering, that $Bel_{good}(\alpha) = 1$ is not reachable in practice we may conclude that assigning both G^a and G^c to 0 deteriorates AA of POMDP-based monitor when we increase the horizon.

In order to reason about the improvement of $MTIME$ we should treat the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ differently. The monitoring time MTIME is only affected by bad executions, but in order to obtain an improvement in the case of $POMDP_2$ it should be the case that $AccProb^1(\alpha) > a_{tr}$ as given in Equation 5.14, however the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ is such that $Alarm$ is raised by the $POMDP_2$. This means that the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ should be negative for some α, while $AccProb^1(\alpha) > a_{tr}$. Clearly, if the value of $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o})$ is never negative then $MTIME$ of $POMDP_2$ is never less than of $POMDP_1$.

Given the lower bound from Equation 5.24 we require that $\mathbb{E}_{o\in O}\, V^1(b_{\alpha o}) \geq 0$. Then the following condition holds:

$$\max \begin{cases} G^a - (L^a + G^a)AccProb^2(\alpha) \\ -L^c + (G^c + L^c)AccProb^2(\alpha) \end{cases} \geq 0 \tag{5.26}$$

The solution for the inequality Equation 5.26 may be given by the intersection point of lines represented with equations inside of operator max. The value of each linear function in the intersection point has to be ≥ 0, i.e.

$$\frac{G^a G^c - L^a L^c}{G^a + L^a + G^c + L^c} \geq 0 \tag{5.27}$$

Since, the values in rewards tuple are nonnegative, we may further reduce the condition in Equation 5.27 to:

$$G^a G^c - L^a L^c \geq 0 \tag{5.28}$$

Condition in the Equation 5.28 defines a relation between reward parameters that guaranties $\mathbb{E}_{o \in O} V^1(b_{\alpha o}) \geq 0$, and, therefore, $MTIME_{POMDP_2} \geq MTIME_{POMDP_1}$.

Although we have performed the analysis for the case of horizon $h = 2$ the same statements hold for any arbitrary horizon. This is true due to the recursive nature of the value function and consistent application of maximization at every horizon depth.

CHAPTER 6

MONITORING SYSTEMS WITH TERMINAL STRONGLY CONNECTED COMPONENTS

As it was shown in Section 4.2 POMDP-based monitors implement a complex decision rule. In general, a rejection condition of such monitors accounts for the probability of future observations. Additionally, it involves a computation of updated belief state for every considered horizon and every possible evolution of the current belief state.

In this chapter, we consider systems with terminal strongly connected components. The great simplification that is provided by this class of systems is in the fact that once a system is in certain states, the set of further reachable states becomes deterministically limited. We use this benefit to simplify the POMDP-based monitor's decision rule to better understand how the space of POMDP monitors is related to the rest of the world.

6.1 Systems with Terminal Strongly Connected Components

To give intuitive understanding of systems with terminal strongly connected components let's imagine that certain states of the system have a special property: every execution involving that state is guaranteed to eventually loop back. Obviously, every state that may be reached on that path has the same property, and thus defines a subset of states that we are going to call a *terminal strongly connected component (TSCC)*.

A formal definition is given as in [11]. Let $f^n_{s,t}$ represent a probability of reaching a state $s \in S$ from a state $t \in S$ in $n \geq 0$ steps for the first time. Further, we define a probability value $f_{s,t} = \sum_{n \geq 0} f^n_{s,t}$, which determines the total probability that the state t is eventually reached from a state s. All the states are classified as *persistent* and *transient*. In order for a state s to be called *persistent* every execution that includes that state is guarantied to loop, i.e. simply saying $f_{s,s} = 1$. This condition is not satisfied in the *transient* states.

A *TSCC* is defined as a largest subset X of S, such that for every pair of states $s, t \in X$ the value of $f_{s,t} > 0$, and for every state $s' \in S \setminus X$ the value of $f_{s',t} = 0$. Intuitively, this means that for every state in the *TSCC* there is a path to another state in the same *TSCC*, but no paths to states not included in the *TSCC*.

In every *TSCC* all states are either *persistent* or *transient*, and hence every *TSCC* is called to be *persistent* or *transient*. It has been shown in [11] that the set of states S can be decomposed in disjoint collection of subsets $T, C_1, C_2, \ldots$, where T is a set of *transient* states, and $C = \{C_1, C_2, \ldots\}$ is the set of *persistent TSCCs (PTSCCs)* .

In the perspective of monitoring problem and the fact that states of the product automaton $\mathcal{B}$ are classified as *good* or *bad* we are able to make some observations with respect to the *PTSCCs*. We call C^{good} a *good PTSCC* if $C^{good} \subset S_{good}$, i.e. every state of the *PTSCC* is *good*. Also, we call C^{bad} a *bad PTSCC* if $C^{bad} \setminus S_{bad} \neq \emptyset$, i.e. there exists a state in C^{bad}, which is a *bad* state. We may observe that every infinite execution that includes a state in C^{good} is *good*, while if it includes a state in C^{bad} is *bad*. Provided that it is known whether the finite prefix of

an infinite execution includes a state in C^{good} or C^{bad} it is possible to deterministically claim that the execution is *good* or *bad*.

To take the full advantage for monitoring of the systems with PTSCCs we define and use further the following assumption:

Assumption 1. *Let α be the sequence of perceived outputs from the system with PTSCCs generated by the execution that includes at least one non-transient state.*

Although Assumption 1 makes the monitoring problem simpler it still remains a hard problem due to the stochastic nature of transitions and partial observability of outputs.

The main implication of Assumption 1 is in computation of the probability that any extension of the finite output sequence α is generated by the bad execution, which is equal to 0. This is so due to the fact that if the current state is in S_{good}, then all the further states in the current executions are also guaranteed to be in S_{good}. As a result, we may rewrite the

expressions for the computation of $AccProb(\alpha)$ and $RejProb(\alpha)$ initially given in Equation 3.5 and Equation 3.6 as follows:

$$\begin{aligned}
RejProb^h(\alpha) &= \int\limits_{s_1 \in S_{bad}} Pr(s_1|\alpha)\mathrm{d}s_1 + \underbrace{\int\limits_{s_1 \in S_{good}} \int\limits_{s_2 \in S_{bad}} Pr(s_1|\alpha)Pr(s_2,s_1)\,\mathrm{d}s_1\,\mathrm{d}s_2}_{=0} + \cdots + \\
&+ \underbrace{\int\limits_{s_1 \in S_{good}} \cdots \int\limits_{s_{h-1} \in S_{good}} \int\limits_{s_h \in S_{bad}} Pr(s_1|\alpha)\ldots Pr(s_h,s_{h-1})\,\mathrm{d}s_1 \ldots\, \mathrm{d}s_h}_{=0} = \\
&= \int\limits_{s_1 \in S_{bad}} Pr(s_1|\alpha)\mathrm{d}s_1 = \\
&= RejProb^1(\alpha) \\
AccProb^h(\alpha) &= 1 - RejProb^h(\alpha) = 1 - RejProb^1(\alpha) = \\
&= AccProb^1(\alpha) \\
RejProb(\alpha) &= \lim_{h\to\infty} RejProb^h(\alpha) = \\
&= RejProb^1(\alpha) \\
AccProb(\alpha) &= \lim_{h\to\infty} AccProb^h(\alpha) = \\
&= AccProb^1(\alpha)
\end{aligned} \tag{6.1}$$

Expressions in Equation 6.1 mean that in systems with PTSCCs under the Assumption 1 the threshold-based monitor (with infinite horizon) is equivalent to the threshold-based monitor with horizon 1.

6.2 Monitoring-POMDP in Systems with TSCCs

Let $\mathcal{B}$ be the product automaton, such that the total set of its states can be decomposed as $S = \{T, C_1, C_2, \ldots C_n\}$, where T is a set of *transient* states, and $\{C_1, C_2, \ldots, C_n\}$ is a finite

set of *PTSCCs*. Some of *TSCCs* $C_i \quad i \in [1,n]$ are *good* and some are *bad* according to the definition in Section 6.1. Let I represent the vector containing indexes of those *TSCCs*, which are *good*. And J represent the vector of those indexes of *TSCCs*, which are *bad*. Then we define the sets $C^{good} = \bigcup_{i \in I} C_i$ and $C^{bad} = \bigcup_{j \in J} C_j$. Obviously $C^{good} \subset S_{good}$ and $C^{bad} \subset S_{bad}$. According to the definition of *PTSCC*, no states in C^{good} transition to any state in C^{bad} and vice versa.

Let's assume that states in C^{good} and C^{bad} and the set of outputs have been decomposed in such a way that every state in $C^{good} \cup C^{bad}$ deterministically emits an output. Note, that we can do that in a similar way as it is described in Section 2.1.5. Formally, this means that for every output $o \in O$, there exists a subset $C_o^{good} \subset C^{good}$, and a subset $C_o^{bad} \subset C^{bad}$, such that $\sum_{s \in C_o^{good} \cup C_o^{bad}} Pr(s|\alpha o) = 1$, where αo is a sequence of perceived outputs such that the last output is o.

In all of our further discussions related to the application of POMDP-based monitor for the systems with *TSCCs* we consider the Assumption 1. As we will also see, it might be useful to think of a specific observation o that has been perceived most recently. Hence, we will mostly use a notation αo and $b_{\alpha o}$ to represent the sequence of observed outputs and the belief state that corresponds to it. Additionally, for convenience, we will expand the value of $Bel_{good}(b_{\alpha o})$ and $Bel_{bad}(b_{\alpha o})$ as follows:

$$\begin{aligned} Bel_{good}(b_{\alpha o}) &= \int_{s \in S_{good}} b_{\alpha o}(s)\,\mathrm{d}s = \int_{s \in S_{good}} Pr(s|\alpha o)\,\mathrm{d}s = \int_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s \\ Bel_{bad}(b_{\alpha o}) &= \int_{s \in S_{bad}} b_{\alpha o}(s)\,\mathrm{d}s = \int_{s \in S_{bad}} Pr(s|\alpha o)\,\mathrm{d}s = \int_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s \end{aligned} \tag{6.2}$$

Then we may rewrite equations for the monitoring POMDP value function initially defined in Equation 4.6 as follows:

$$
\begin{aligned}
Q^h(b_{\alpha o}, \mathit{Alarm}) &= G^a \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s - L^a \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s \\
Q^h(b_{\alpha o}, \mathit{Continue}) &= -L^c \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s + G^c \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s + \\
&+ \sum_{o' \in O} Pr(o'|\mathit{Continue}, b_{\alpha o}) V^{h-1}(b_{\alpha o o'}) \qquad (6.3) \\
V^h(b_\alpha) &= \begin{cases} \max\limits_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned}
$$

By considering Assumption 1 and applying the standard techniques [11], we may obtain a set of equations equivalent to Equation 6.3, but fully represented through the belief state probabilities, the transition probabilities and the monitoring POMDP rewards. Equation 6.4 lists the representation of rejection decision rule of the monitoring POMDP for the case of

systems with *TSCCs*. For compactness and convenience of reading the derivation is skipped here and may be found in Appendix C.

$$
\begin{aligned}
Q^h(b_{\alpha o}, \mathit{Alarm}) &= G^a \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s - L^a \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s \\
Q^h(b_{\alpha o}, \mathit{Continue}) &= -L^c \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s + G^c \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s + \\
&+ \sum_{o' \in O} \bar{V}^{h-1}(b_{\alpha o o'}) \qquad (6.4) \\
\bar{V}^h(b_\alpha) &= \begin{cases} \max\limits_{a \in A} \bar{Q}^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&\bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, \mathit{Alarm}) = \\
&= G^a \int\limits_{s \in C_o^{bad}} \int\limits_{s_1 \in C_{o1}^{bad}} \cdots \int\limits_{s_k \in C_{ok}^{bad}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k)\,\mathrm{d}s\,\mathrm{d}s_1 \ldots\,\mathrm{d}s_k - \\
&- L^a \int\limits_{s \in C_o^{good}} \int\limits_{s_1 \in C_{o1}^{good}} \cdots \int\limits_{s_k \in C_{ok}^{good}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k)\,\mathrm{d}s\,\mathrm{d}s_1 \ldots\,\mathrm{d}s_k \\
&\bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, \mathit{Continue}) = \\
&= -L^c \int\limits_{s \in C_o^{bad}} \int\limits_{s_1 \in C_{o1}^{bad}} \cdots \int\limits_{s_k \in C_{ok}^{bad}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k)\,\mathrm{d}s\,\mathrm{d}s_1 \ldots\,\mathrm{d}s_k + \\
&+ G^c \int\limits_{s \in C_o^{good}} \int\limits_{s_1 \in C_{o1}^{good}} \cdots \int\limits_{s_k \in C_{ok}^{good}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k)\,\mathrm{d}s\,\mathrm{d}s_1 \ldots\,\mathrm{d}s_k + \\
&+ \sum_{o_{k+1} \in O} \bar{V}^{h-1}(b_{\alpha o o_1 \ldots o_k o_{k+1}})
\end{aligned}
$$

6.3 Monitoring-POMDP in Simplified Case of Systems with TSCCs

In Equation 6.4 we were able to show that in systems with *PTSCCs* POMDP value functions may be computed without explicitly represented output probabilities. However, the overall

complexity of computations remains high due to the per-state representation of belief probabilities $Pr(s|\alpha o)$ that have to be used while integrating over all the states in C_o^{bad} and C_o^{good}. In order to simplify the computations even further let's consider the following assumption.

Assumption 2. *For every output $o \in O$ there exists a single state $s_o^{good} \in C^{good}$ and $s_o^{bad} \in C^{bad}$ where output o may be perceived.*

Although Assumption 2 is strong, it still keeps the problem partially observable. Then the following formal statements may be done:

$$\begin{aligned}
\int_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s &= Pr(s_o^{bad}|\alpha o) \\
\int_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s &= Pr(s_o^{good}|\alpha o) \\
\int_{s \in C_o^{bad}} \int_{s' \in C_{o'}^{bad}} Pr(s|\alpha o) Pr(s,s')\,\mathrm{d}s\,\mathrm{d}s' &= Pr(s_o^{bad}|\alpha o) \int_{s' \in C_{o'}^{bad}} Pr(s_o^{bad},s')\,\mathrm{d}s' \\
\int_{s \in C_o^{good}} \int_{s' \in C_{o'}^{good}} Pr(s|\alpha o) Pr(s,s')\,\mathrm{d}s\,\mathrm{d}s' &= Pr(s_o^{good}|\alpha o) \int_{s' \in C_{o'}^{good}} Pr(s_o^{good},s')\,\mathrm{d}s'
\end{aligned} \tag{6.5}$$

By substituting outcomes of equations in Equation 6.5 with Equation 6.4 we may obtain the following value functions of POMDP-based monitor:

$$
\begin{aligned}
Q^h(b_{\alpha o}, Alarm) &= G^a Pr(s_o^{bad}|\alpha o) - L^a Pr(s_o^{good}|\alpha o) \\
Q^h(b_{\alpha o}, Continue) &= -L^c Pr(s_o^{bad}|\alpha o) + G^c Pr(s_o^{good}|\alpha o) + \\
&+ \sum_{o' \in O} \bar{V}^{h-1}(\alpha o o') \\
\bar{V}^h(b_{\alpha)} &= \begin{cases} \max_{a \in A} \bar{Q}^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases} \\
\bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, Alarm) &= G^a Pr(s_o^{bad}|\alpha o) Pr(s_o^{bad}, s_{o_1}^{bad}) \ldots Pr(s_{o_{k-1}}^{bad}, s_{o_k}^{bad}) - \\
&- L^a Pr(s_o^{good}|\alpha o) Pr(s_o^{good}, s_{o_1}^{good}) \ldots Pr(s_{o_{k-1}}^{good}, s_{o_k}^{good}) \\
\bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, Continue) &= -L^c Pr(s_o^{bad}|\alpha o) Pr(s_o^{bad}, s_{o_1}^{bad}) \ldots Pr(s_{o_{k-1}}^{bad}, s_{o_k}^{bad}) + \\
&+ G^c Pr(s_o^{good}|\alpha o) Pr(s_o^{good}, s_{o_1}^{good}) \ldots Pr(s_{o_{k-1}}^{good}, s_{o_k}^{good}) + \\
&+ \sum_{o_{k+1} \in O} \bar{V}^{h-1}(b_{\alpha o o_1 \ldots o_k o_{k+1}})
\end{aligned}
\tag{6.6}
$$

Note, that due to the simplification, the state emitting an most recent output o is either in C_o^{bad} or C_o^{good}, and, therefore,

$$
Pr(s_o^{bad}|\alpha o) = 1 - Pr(s_o^{good}|\alpha o) \tag{6.7}
$$

Hence, it is safe to state that decision condition of that POMDP-based monitor has only 1 parameter which is not defined by the monitored system or the monitor itself. Without loss of generality we may represent the value function in Equation 6.6 as a function of $p = Pr(s_o^{bad}|\alpha o)$, which represents the probability that execution has reached a bad state after perceiving sequence

of outputs αo. For simplicity of further analysis we will assume that the value of horizon $h = 2$, however as we will see the same conclusion may be extended to any arbitrary finite horizon.

We restate the rejection decision rule that is obtained by comparing the value functions for the action *Alarm* and *Continue*:

$$Ap \geq B + \sum_{o_i \in O} max \begin{cases} A'(o, o_i)p - B'(o, o_i) \\ -A''(o, o_i)p + B''(o, o_i) \end{cases} \tag{6.8}$$

The parameters A and B in Equation 6.8 are fully determined by the values of rewards, while functional parameters $A'(o, o_i)$, $A''(o, o_i)$, $B'(o, o_i)$ and $B''(o, o_i)$ depend on the rewards and the transition probabilities. Formally, these parameters are defined as follows:

$$\begin{aligned} A &= G^a + L^a + G^c + L^c \\ B &= G^c + L^a \\ A'(o, o_i) &= G^a Pr(s_o^{bad}, s_{o_i}^{bad}) + L^a Pr(s_o^{good}, s_{o_i}^{good}) \\ B'(o, o_i) &= L^a Pr(s_o^{good}, s_{o_i}^{good}) \\ A''(o, o_i) &= L^c Pr(s_o^{bad}, s_{o_i}^{bad}) + G^c Pr(s_o^{good}, s_{o_i}^{good}) \\ B''(o, o_i) &= G^c Pr(s_o^{good}, s_{o_i}^{good}) \end{aligned} \tag{6.9}$$

Every maximization operator appearing in the right-hand side of Equation 6.8 represents a piece-wise linear function containing a pair of pieces. For every output o_i we may define a breaking point for the corresponding piece-wise linear function as

$$p_i^* = \frac{B'(o, o_i) + B''(o, o_i)}{A'(o, o_i) + A''(o, o_i)} \tag{6.10}$$

Without loss of generality we may assume that outputs o_i have been ordered in such a way that sequence of corresponding breaking points is in an ascending order, i.e. $p_i^* \geq p_{i-1}^* \quad \forall i > 1$. Also let's define $p_0^* = 0$ and $p_{n+1}^* = 1$, where $n = ||O||$. Clearly, $p_0^* < p_1^*$ and $p_n^* < p_{n+1}^*$.

To solve Equation 6.8 with respect to p we need to consider every interval $p \in [p_{i-1}^*, p_i^*] \quad \forall i \in [1, n+1]$ and union all $n+1$ resulting conditions on p.

For every chosen interval $p \in [p_{i-1}^*, p_i^*] \quad \forall i \in [1, n+1]$ the solution of the inequality would be always of the form $p \geq X_i$, where $X_i \in \mathbb{R}$. This can be shown by considering the largest possible coefficient of variable p in the right-hand side of Equation 6.8. Indeed, given that $p \in [p_{i-1}^*, p_i^*]$ the Equation 6.8 will transform into a simple linear inequality of the form:

$$Ap \geq B + Cp + D \tag{6.11}$$

Due to the fact that functions $A'(o, o_i)$ and $A''(o, o_i)$ are nonnegative, the largest possible value of C is:

$$\begin{aligned} C_{max} &= \sum_{o_i \in O} A'(o, o_i) = \\ &= G^a \sum_{o_i \in O} Pr(s_o^{bad}, s_{o_i}^{bad}) + L^a \sum_{o_i \in O} Pr(s_o^{good}, s_{o_i}^{good}) = \\ &= G^a + L^a \leq G^a + L^a + G^c + L^c = A \end{aligned} \tag{6.12}$$

Therefore, the solution of Equation 6.11 would be always of the form $p \geq X_i$, where $X_i \in \mathbb{R}$.

At this point, we need to remember that we would obtain the condition $p \geq X_i$ as a result of assumption that $p \in [p^*_{i-1}, p^*_i]$. Thus, there might be no solution in case if $X_i > p^*_i$. However, let's assume that i_{min} is the smallest value of observation index i such that solution of Equation 6.8 exists and is of the form $p \in [X_{i_{min}}, p^*_i]$, where $X_{i_{min}} \in [p^*_{i-1}, p^*_i]$. It is in our interest to see if $\forall i > i_{min}$ the corresponding solution of inequality in Equation 6.8 would be $p \in [p^*_{i-1}, p^*_i]$, i.e. the final solution of Equation 6.8 would be $p \geq X_{i_{min}}$.

In order to show this, we would need to consider the representation of Equation 6.8 for two consecutive intervals $p \in [p^*_{k-1}, p^*_k]$ and $p \in [p^*_k, p^*_{k+1}]$. Let's also assume that $k = i_{min}$, and, hence, there exists a solution for the resulting representation of Equation 6.8.

Assuming that $p \in [p^*_{k-1}, p^*_k]$, we rewrite the Equation 6.8 as follows:

$$\begin{aligned} Ap &\geq B + \sum_{i=1}^{k-1} \big(A'(o,o_i)p - B'(o,o_i)\big) + \\ &+ A'(o,o_k)p - B'(o,o_k) + \\ &+ \sum_{i=k+1}^{n} \big(-A''(o,o_i)p + B''(o,o_i)\big) \end{aligned} \tag{6.13}$$

$$\begin{aligned} Ap - B &- \\ -\sum_{i=1}^{k-1} \big(A'(o,o_i)p - B'(o,o_i)\big) &- \\ -\sum_{i=k+1}^{n} \big(-A''(o,o_i)p + B''(o,o_i)\big) &\geq A'(o,o_k)p - B'(o,o_k) \end{aligned}$$

Similarly, for the interval $p \in [p^*_k, p^*_{k+1}]$ we obtain identical inequality:

$$\begin{aligned} Ap - B &- \\ -\sum_{i=1}^{k-1} \big(A'(o,o_i)p - B'(o,o_i)\big) &- \\ -\sum_{i=k+1}^{n} \big(-A''(o,o_i)p + B''(o,o_i)\big) &\geq -A''(o,o_k)p + B''(o,o_k) \end{aligned} \tag{6.14}$$

Left hand side of both inequalities in Equation 6.13 and Equation 6.14 is identical, and represents some linear function of p. It may be shown that this linear function is increasing with the value of p. Figure 3 represents visually the problem defined by Equation 6.13 and Equation 6.14. Blue and red lines represent linear functions from the right-hand side of Equation 6.13 and Equation 6.14. The dashed black lines represent possible linear function matching to the left-hand side of Equation 6.13 and Equation 6.14. Note, that intersection of the black dashed lines with blue line has to be before or at p^*_{k-1}.

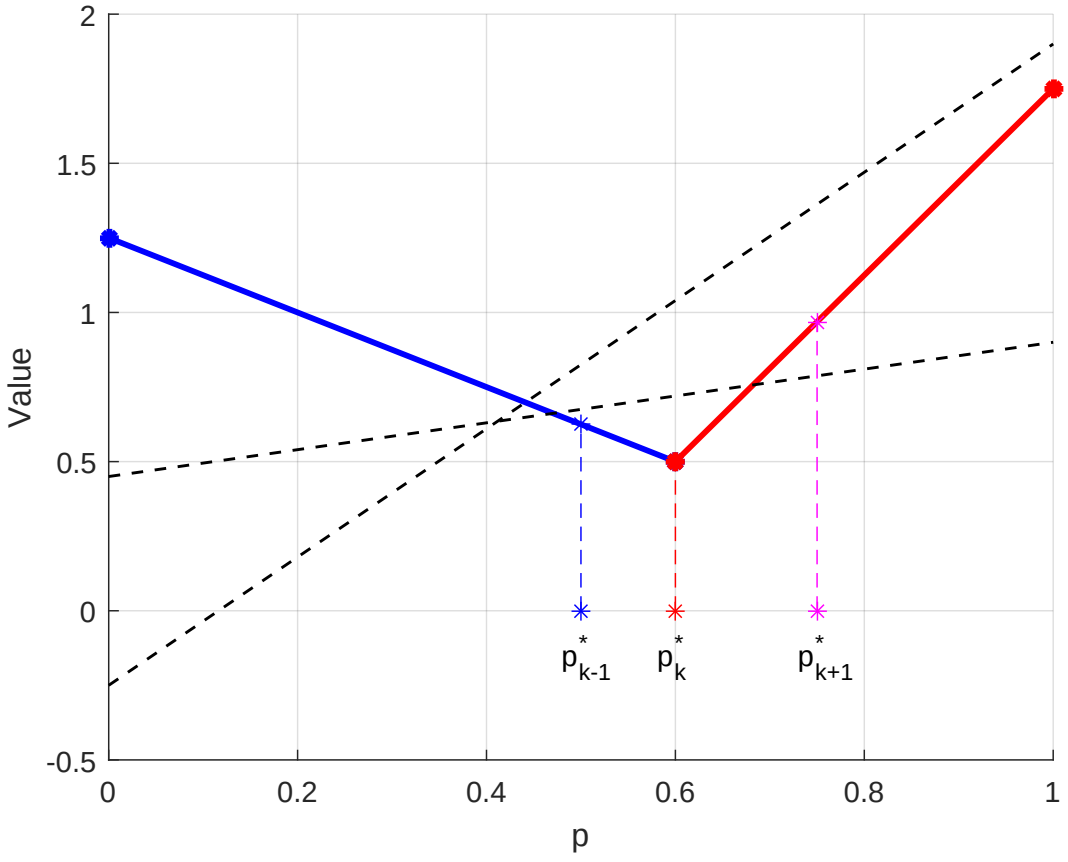

Figure 3. Linear functions in decision rule.

From the Figure 3 we may see that in order for Equation 6.14 to hold for the whole interval $p \in [p_k^*, p_{k+1}^*]$ it is necessary and sufficient that linear functions defined on the both sides of Equation 6.14 don't intersect. We may be able to make this even more strict if we require that those linear functions never intersect in the whole interval of $[p_k^*, 1]$. This may be enforced by comparing the value of each function at $p = 1$. I.e.

$$A - \sum_{i=1}^{k-1} \big(A'(o, o_i) - B'(o, o_i)\big) - \sum_{i=k+1}^{n} \big(- A''(o, o_i) + B''(o, o_i)\big) - B \geq A'(o, o_k) - B'(o, o_k) \tag{6.15}$$

Substituting values in Equation 6.15 in accordance with Equation 6.9 implies a condition

$$L^c + G^c - G^a \geq 0 \tag{6.16}$$

Condition in Equation 6.16 is sufficient, but not necessary for a solution of Equation 6.8 to be in the form of $p \geq X, \quad X \in \mathbb{R}$. The beauty of Equation 6.16 is in independence from the parameters of the monitored system. On the other hand, finding a necessary and sufficient condition is also not hard and only requires a substitution of p with p_{k+1}^* in Equation 6.14, but will depend on the inherent properties of the monitored system.

The condition on the probability parameter p that we have obtained represents the simplified rejection decision rule for the monitoring-POMDP of the class of systems with *TSCCs* satisfying Assumption 2. We were able to derive that under certain conditions the interval of p is continuous starting from some $X \in [0, 1]$ all the way until 1. However, similarly it may be also shown that there exist cases such that the probability the POMDP-based monitor rejects

when p is in the union of discontinued set of probability intervals. According to the definition of p it is easy to see that $RejProb^1(\alpha o) = p$. Therefore, we may claim that rejection decision rule of monitoring POMDP is either equivalent to

$$RejProb^1(\alpha o) \geq r_{th}(o), \quad r_{th}(o) \in [0,1] \tag{6.17}$$

or

$$\begin{aligned} &RejProb^1(\alpha o) \in \left[r_{th_1}(o), r_{th_2}(o)\right] \cup \left[r_{th_3}(o), r_{th_4}(o)\right] \cup \cdots \cup \left[r_{th_n}(o), 1\right] \\ &r_{th_1}(o) \neq r_{th_2}(o) \neq \ldots r_{th_n}(o) \in [0,1] \end{aligned} \tag{6.18}$$

It is easy to see that the above mentioned conclusion holds for arbitrary horizon of monitoring-POMDP. Both decision rules that we obtained in Equation 6.17 and Equation 6.18 are more advanced than the decision rule of traditional threshold-based monitors and may result in improvement of monitoring characteristics. Note, that due to Assumption 1 the value of $RejProb(\alpha o) = RejProb^1(\alpha o)$, and, therefore, we don't need to consider the approximation for threshold-based monitors. In the case of Equation 6.17 the advantage is in more precise definition of the threshold value, which is dependent on the observation. While threshold-based monitors always use the same threshold value, it might not capture difference in good and bad executions. By defining the threshold value as a function of the observation we may utilize the certain properties of the monitored system, that may imply some observations to be indicative of bad or good executions. Rejection criteria in Equation 6.18 is extended further by excluding intervals of potential $RejProb^1(\alpha)$, which may not happen in good, but happen in bad executions.

With the considered example of systems with *TSCCs* we were able to fully solve the monitoring-POMDP, and observe the difference in the decision criteria with respect to the class of threshold-based monitors.

CHAPTER 7

DECISION-THEORETIC MONITORING TOOL

7.1 Purpose and Applications

At every step of study of various monitoring techniques, it is often necessary to implement the monitor itself, the system that needs to be monitored and the property that the monitor should verify. We have solved this problem by designing and implementing a *Decision-Theoretic Monitoring Tool* (DTMT), which is a generic modular software that is suitable for simulation of the system, analysis of various monitoring decision rules by estimating the performance measures and comparison between different monitors.

The key features of DTMT software package are:

- Platform independent implementation: Microsoft Windows and Linux OS are supported.
- Simulation-based measurement of monitoring performance.
- Online monitoring of system execution.
- Fully modular design for an easy extension of monitoring techniques and implementation of arbitrary system and property.
- Fully customized model (system and property) definition.
- Parallelization support for the fastest monitor performance evaluation (only in simulation mode).

- Parallel evaluation of multiple monitoring techniques for comparable results.
- Delivered as a standalone application or library for the integration with other software packages.

At this time, the state estimation implementation is fixed with particle filtering, which may be extended to a broader range of state estimators in the future.

The DTMT is implemented in C++ with use of Boost [51] libraries. The tool may be distributed in the form of a standalone executable or a library for external use. Additional monitoring techniques and systems have to be supplied in the form of library module with a fixed set of external API. For easy extendability, we provide ready to use templates, which define placeholders that should be filled with implementation for both - a monitor decision rule, and a monitored system with a property.

7.2 Architecture Design

Before we proceed to a detailed explanation of decision rules and model representation we introduce the architecture design of the DTMT in a very abstract form. We will use some of the language that might be more clear if the reader is familiar with the basics of *object oriented design* and *software engineering principles*. At the same time, we will, in most cases, avoid discussion of the implementation details, for which we suggest to look directly into the code.

The deliverables of the DTMT are visualized in Figure 4 and are split into following parts:

- `monitoring_experiment` application - a main executable performing experiments and evaluation of various decision rules and models.

- `monitoring_interface` library - a shared library defining interfaces for external use of the monitoring techniques and models.
- `monitoring_common` library - a shared library implementing and defining primary data structures shared across all the other components.
- set of `monitor_[decision]` libraries - implementation of specific decision rules, i.e. monitoring algorithms.
- set of `state_[model]` libraries - implementation of specific monitoring problems.

The listing presented above shows that we support extensions in the form of libraries that define new decision rules and arbitrary monitoring problems. Details about that will follow in the Sections 7.3 and 7.4.

Static library `monitoring_common` carries most of the implementation that is needed to connect all the parts together. Here we present the most important set of classes and data structures defined in `monitoring_common`:

- `BeliefState` - class manages representation of the belief state using a set of particles
- `DecisionCommon` - class defines a set of static functions required for belief propagation and calculation of rejection probability.
- `DecisionMaker` - abstract class defines main set of functions that need to be available in any actual decision maker implementation.
- `DecisionSystem` - class managers several decision makers at a time, maintains simulations by keeping simulation model and belief state.

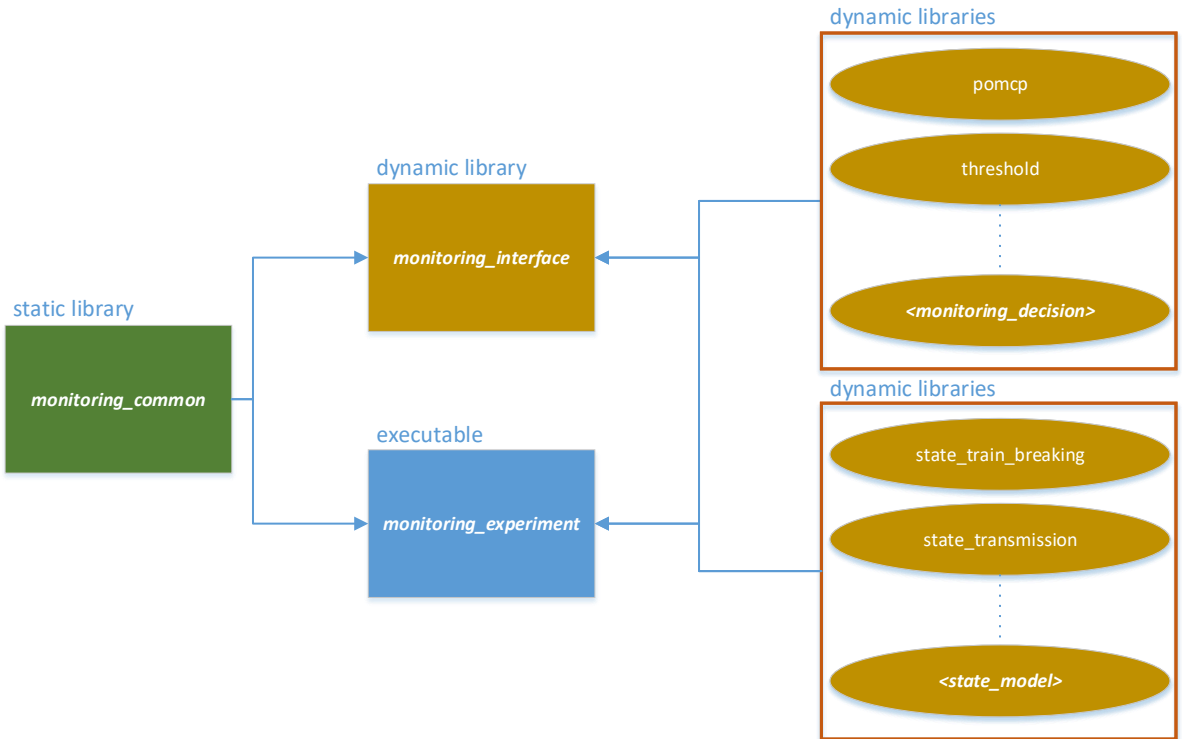

Figure 4. DTMT architecture.

- `State` - abstract class defines all the functions that need to be available for every experimental or actual model state that needs to be monitored.
- `Model` - internal interface between State and DecisionSystem
- `ModelInput` - abstract class defining interfaces of a single input
- `ModelInputFunction` - abstract class defining interfaces of an input function by time
- `ModelObservation` - abstract class defining interfaces of a single observation (output)
- `ModelRewardsSystem` - data class maintaining values of rewards
- `ModelSimulationOptions` - data class maintaining a set of options specifically related to simulations
- `SolverOptions` - data class maintaining a set of options of the solver performing experiment

The dynamic library `monitoring_interface` and executable `monitoring_experiment` are build upon the `monitoring_common` static library. The main difference is that executable is able to perform experiments and produce monitoring performance results, while the dynamic library is expected to be used for monitoring of a single run of the system.

7.3 Monitoring Decision Rule Representation

The main data type that is used to represent a monitoring decision rule is `DecisionMaker` and is defined in `monitoring_common` library. This is an abstract data type, which means that it only defines a list of required interfaces and may not have all the implementation.

In order to define a new monitoring rule the following steps are required:

1. Prepare for the definition of a new shared library using a favorite development environment, e.g. Eclipse CDT.
2. Define a new C++ class derived from the abstract class `DecisionMaker` and implement all the abstract methods. The most important of these methods are:
   - `GetAction` - identifies the action (*Alarm* or *Continue*) that has to be executed from the current *BeliefState*.
   - `LoadOptions` - reads from the given file and initializes all the options of the decision rule, that might be important for the experiment or run-time use.
3. Define an interface global function `CreateDecisionMaker` that will be executed every time when the client application needs to create an object to represent the decision rule.

Using a class defined in the `monitoring_common` library - `DecisionSystem`, it is possible to run multiple decision rules on the same belief state simultaneously. This provides consistent and comparable outcomes, which are very useful for the comparison of various methods.

The *DTMT* comes with several decision rules already implemented and ready for use. Here is a list of supported monitoring techniques:

- Threshold-based monitor (`monitor_threshold` library) - implementation of the threshold-based monitor on any arbitrary horizon with the following parameters:
  - `threshold` - real valued rejection probability threshold from the range $[0, 1]$.

  - `accuracy` - real valued precision to be used when a pair of numbers is compared. I.e. a pair of real valued numbers is considered to be equal if their difference is smaller than this value.
  - `horizon` - the depth of horizon to be considered when rejection probability of the belief state is approximated.
  - `simulations` - a number of Monte-Carlo simulations to be used for the approximation of rejection probability for the value of horizon greater than 1.

- POMDP-based monitor (`monitor_pomdp` library) - implementation of the POMDP-based monitor using POMCP technique and the following parameters:
  - `simulations` - number of Monte-Carlo simulations in POMCP computation.
  - `discount` - a value of POMDP discount factor, often assumed to be 1.
  - `explorationconstant` - a value of the POMCP exploration constant.
  - `maxtreedepth` - a maximum depth of the search tree in POMCP, similar to the length of horizon.
  - `good_alarm, good_continue, bad_alarm, bad_continue` - rewards system of the POMDP.

Options for each decision rule are expected in the form of text file, where every line is of the form:

```
[parameterName] = [value]
```

7.4 Model Representation

Support for user defined models is implemented in the *DTMT* in the similar way as we have already shown for the monitoring decision rules in the Section 7.3.

In *DTMT* every model is represented by the definition of the state variables, transition system, observation system, format of the supplied inputs and the input function, which is producing an input for every time instance. From the perspective of the implementation this is done by overriding the abstract data types `Model, State, ModelInput, ModelInputFunction` and `ModelOutput`. We provide more details about every data type and list all the basic steps that need to be performed to define a new experimental model.

In order to define a new model the following steps are required:

1. Prepare for the definition of a new shared library using a favorite development environment, e.g. Eclipse CDT.
2. Define a set of new C++ classes derived from the abstract classes as listed below and implement all the abstract methods:
    - `Model` - identifies data types that are used for the state, observation and input.
    - `ModelObservation` - defines a set of variables that are observed from the state, and the way to convert it to textual representation for logging purposes.
    - `ModelInput` - defines a set of variables that are required by the state to execute a transition towards the next time instance.

- `State` - defines all the state variables and how the transitions affect the state. We only list some of the most important methods that need to be overriden to complete the data type implementation:
  - `Continue` - propagate current state to the next time instance
  - `ResetState` - reset current state to the initial state
  - `IsFailureState` - check if current state represents a failure
  - `GetObservation` - generate an observation according to the observation model
  - `WeightObservation` - calculate the probability weight of the given observation at the current state
  - `ToString` - convert state to the textual representation for logging

3. Define an interface global function `CreateModel` that will be executed every time when the client application needs to create an object to represent the model.

The *DTMT* currently comes with few models that are implemented and ready for use:

- Simplified example of train braking system [8] with only one train car (`state_single_car_train_brakin` library).
- Multi car train braking system [8] (`state_multiple_car_train_braking` library).
- Simplified automatic transmission system [1] (`state_rpm` library).

7.5 Using the Tool

The *DTMT* may be used as a standalone application (`monitoring_experiment`) executable) and externally in the form of shared library (`monitoring_interface`) with any other application.

First we cover the basics of standalone use of the *DTMT*. Parameters of the application may be either supplied directly in the command line of in the file. Here we show some sample command lines, which are more appropriate for Linux OS, however, it is straightforward to make them compatible with Microsoft Windows OS. In order to execute the `monitoring_experiment` by specifying all the parameters in a file, the following command line should be used:

```
monitoring_experiment --config config.ini
```

Configuration file is a textual file where every line is of the format:

```
[parameter] = [value]
```

Here is a list of all the parameters that may be specified in the configuration file:

- `mode = {experiment, generate_trials, report}`
  - `experiment` - execute the experiment using the set of decision rules defined in the directory given by parameter *experiment_setup* on the model given in *experiment_model*.
  - `generate_trials` - generate a number of experiment trials given by parameter *simulation_trials* for the model given in *experiment_model*, and save trials in the directory specified in *experiment_trials*.

  - `report` - generates a report file according to the data stored in experiment directory given in *experiment_output*.

- `experiment_setup` - directory with a list of files, each specifying options for the decision rule. Additional file *setup.ini* specifies a list of decision rules libraries with a corresponding name of the input file, in the order these decision rules are applied for a single belief state.
- `experiment_model` - dynamic library path for the model specifying the model to be monitored.
- `simulation_trials` - number of simulated trials of the model given in *experiment_model*.
- `simulation_trial_maxlength` - maximum number of time steps for each simulation trial.
- `simulation_trial_maxerrortime` - maximum time instance when the failure may happen during the simulation of the experiment model.
- `experiment_trials` - directory path to save simulated model trials
- `experiment_output` - directory path to save experiment output data
- `experiment_threads` - number of CPU threads to be used during the simulation in order to boost the performance.
- `experiment_trial_repeats` - number of times to repeat experiment on a single trial in order to average the effect of the state estimation
- `experiment_belief_size` - number of particles to be used to approximate the belief state using particle filtering

- `experiment_input_function` - path to the file defining the input function to be supplied to the experiment model during simulation and experiment.
- `report` - path to the generated report file.

7.6 Availability

The source code for the DTMT is available at the online repository accessible at the following address:

`https://bitbucket.org/ayavolovsky/dtmt`

We advise to follow the instructions from the readme file, and check out suggested examples for the implementation of new decision rules and custom system models.

Source code may be opened, edited and built using Eclipse CDT IDE. In addition to that all the builds may be done through the command line by executing appropriate make files.

CHAPTER 8

EXPERIMENTAL EVALUATION

(Previously published as Yavolovsky A., Žefran M., Sistla A.P. (2016) Decision-Theoretic Monitoring of Cyber-Physical Systems. In: Falcone Y., Sánchez C. (eds) Runtime Verification. RV 2016. Lecture Notes in Computer Science, vol 10012. Springer, Cham)

8.1 Monitoring of Transmission System

In order to evaluate the efficiency of the decision-theoretic monitor, we use a mobile robot with a software implementation of a 2-gear transmission system [1]. By switching gears, the engine revolutions per minute (RPM) are maintained in a safe range. However, due to failures, RPM might increase and stay beyond the limit over a period of time. This may lead to the engine damage and should be promptly detected. In real vehicles, shutdown procedure for such a monitoring system could be implemented in the form of fuel supply cut off.

We emphasize that while this system is rather simple, it demonstrates that decision-theoretic monitoring techniques can be used in real time and that the POMDP-based monitor works well in practice.

The automatons for the system and monitored property are shown in the Figure 5. The discrete modes of the hybrid system are described by the variable d, c is a timer, a_1 and a_2 correspond to the linear acceleration of the vehicle when in the corresponding gear, and $n_1, \ldots, n_4$ are disturbances. The function $u(t) \in [-1, 1]$ represents the control input from the

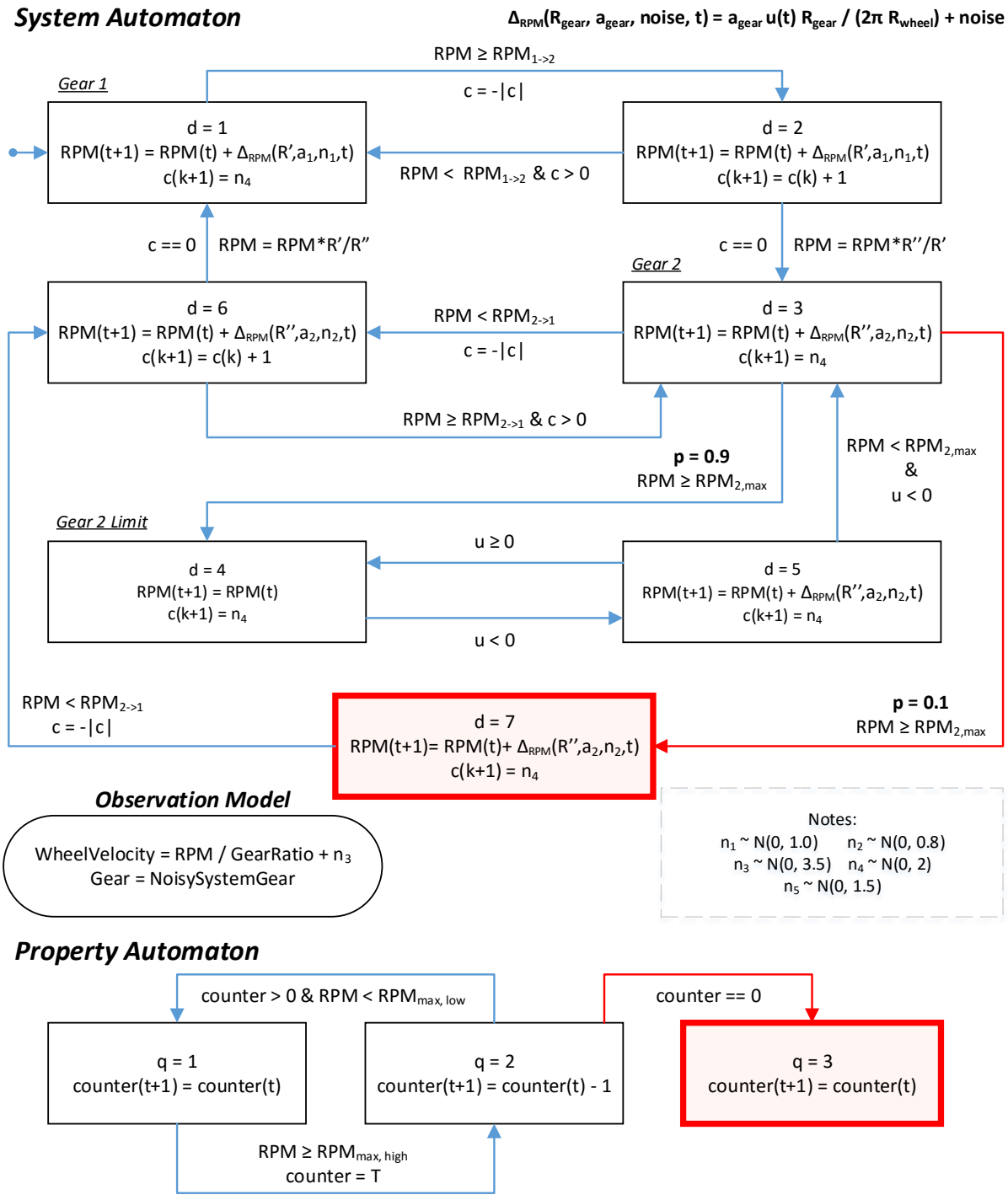

Figure 5. Experimental model.

combined accelerator/brake pedal. The positive values of $u(t)$ correspond to acceleration, while the negative values correspond to braking.

The system starts from the mode $d = 1$ with acceleration dynamics of the first gear. Once the RPM is above the predefined constant $R_{1\to 2}$ a transition to the intermediate mode $d = 2$ occurs and the timer c is set by the random variable n_4. The mode $d = 2$ models the delay due to the shifting between gears. As long as RPM is kept above $R_{1\to 2}$ at least for the time defined by the counter c, the transmission system physically switches to the second gear and the mode $d = 3$ becomes active. In a similar way, the gear may be switched back to the first gear. If the RPM continues to increase and is eventually greater or equal to $R_{2,max}$ the system transitions into mode $d = 4$ that limits the RPM by ignoring any positive, i.e. accelerating, control input $u(t)$. Once the RPM is back to the acceptable range, the system returns to the mode $d = 3$. A nondeterministic transition from the mode $d = 3$ to the mode $d = 7$ is to model a possible failure when the RPM limiting system fails to engage.

The observation model consists of two noisy variables $WheelVelocity$ and $Gear$. $WheelVelocity$ is a vehicle's wheel velocity as a function of RPM distorted by a noise. $Gear$ is a distorted observation of the transmission gear, which matches the actual gear with probability 0.9.

The property automaton is a safety automaton : the engine RPM may not exceed a safe limit for more than time T.

The property automaton is a safety automaton : the engine RPM may not exceed a safe limit for more than time T. The initial state of the property automaton is $q = 1$. Transition to the state $q = 2$ happens once the $RPM \geq RPM_{max,high}$, and the transition back to state $q = 1$

Figure 6. Experimental robot.

occurs when the value of RPM drops below the $RPM_{max,low}$. Value of the $RPM_{max,low}$ and $RPM_{max,high}$ are not necessarily equal to the value of $RPM_{2,max}$ from the system automaton, but we require that $RPM_{max,low} < RPM_{max,high}$ in order to avoid immediate switching that may occur due to the noise in RPM difference equations of the system.

8.2 Experiment Setup

In order to obtain valid results from the experiment, we have designed it in order to average the stochastic effect of state propagation algorithm, which is represented in our implementation in the form of the particle filter.

Stochastic nature of the decision maker, i.e. POMDP implemented with POMCP, that we are using for the experiment requires us to run through the same trajectory of states many times to average the outcome. We conduct the experiment by applying different monitors many times on a set of trajectories that were collected from the physical system. Note that while we operate on recorded data we are able to apply the same monitors online, but we opt out not to do so since experiment will be significantly delayed by the hardware performance and other communication bottlenecks.

For the experimental analysis, we record a set of good and bad executions with corresponding outputs, which will be used in the experiment as corresponding observations. Every bad execution, obviously, includes the episode of transitioning into the bad state but also includes a sequence of states that follow after that. This is an important step to allow enough time for the simulated monitor runs to capture the failure. In the case, if the monitor is unable to detect the failure of the system until the end of recorded bad execution we consider this a missed alarm.

For the purpose of this experiment, we have designed the mobile robot that served as a test platform for running the model of transmission system given in Section 8.1. Although the robot does not physically have a transmission system, we simulate the value of engine RPM from the rotational velocity of the wheel acquired from the rotary motor encoder. This reading adds noise to the observations, which make the whole experiment interesting and relevant to the challenges addressed in our study.

For the purpose of input function $u(t)$, which represents the potential interaction between the user and the engine in the form of acceleration and braking level, we have fixed a sequence of

values, which are aimed to increase, decrease and keep velocity constant within certain intervals of time. This sequence of output is replayed in the loop to make transitions of the system model drive execution through every state.

As we are required to record both good and bad executions we have decided to implement an option to manually control the probability of transition into the mode $d = 7$. Once the system got to run for long enough we manually increase the probability of transition from the mode $d = 3$ to $d = 7$ to the value of $p = 1$, and the execution fails immediately after it returns back to the mode $d = 3$ and transition condition is satisfied.

Experimental data was collected from the robot using *Robot Operating System (ROS)* API [52]. The main building module in ROS is called a node, and for our experiment, we have implemented two ROS nodes. One node is running directly on the *Arduino Mega* board controlling and communicating with motors and motor encoders. The data acquired from the motor encoders is transferred to the workstation PC running the second node in the form of ROS packets. This node is using the DTMT package, described in Chapter 7 to simulate the transmission system model and record the sequence of outputs from the robot sensors in conjunction with actual system state for the further experimental analysis.

8.3 Results

To perform analysis of performance for different monitors we have recorded 14 different state trajectories from the robot. In half of those executions, the mode limiting increase of RPM did not get activated correctly, which led the failure. For every recorded execution we ran a number of POMDP-based monitors configured with different values of rewards, and a

threshold-based monitor with horizon 1. Given the probabilistic nature of each monitor we ran them repeatedly 100 times and averaged the resulting accuracy and monitoring time.

Every POMCP-based monitor was configured to use the discount factor $\gamma = 1$, with a maximum depth of the search tree (search horizon) equal to 20. The total number of simulations executed to construct the search tree and determine the optimal action was 1000. We have used 1000 particles to sample the belief state both in the threshold-based and POMDP-based monitors.

For every monitor run and every system trajectory, we count how many times the alarm was raised or missed. Then, the acceptance and rejection accuracies, and monitoring time, denoted by AA, RA, and $MTIME$ respectively, were computed according to:

$$AA = \frac{g_a}{g_a + g_r} \quad RA = \frac{b_r}{b_a + b_r} \quad MTIME = \frac{\sum_{i=1}^{b_r} T_{b_r}^i}{b_r}, \tag{8.1}$$

where g_a (resp., g_r) is the number of good runs that were accepted (resp., rejected), b_r (resp., b_a) is the number of bad runs that were rejected (resp., accepted), and $T_{b_r}^i$ is the from when the failure occurs to when the monitor raises an alarm. Note that g_r corresponds to the number of false alarms, and b_a to the number of missed alarms; the accuracies approach 1 as these numbers approach 0. An execution was considered good if the state of the property automaton at the end of the run was not representing a failure, and bad otherwise.

The number of raised false alarms as well as the convergence time depends on the monitor configuration. For the case of the threshold-based monitor, it depends on the value of the

threshold. The lower the threshold, the larger will be the number of false alarms and the smaller will be the monitoring time. Threshold monitors do not allow these two quantities to be independently adjusted.

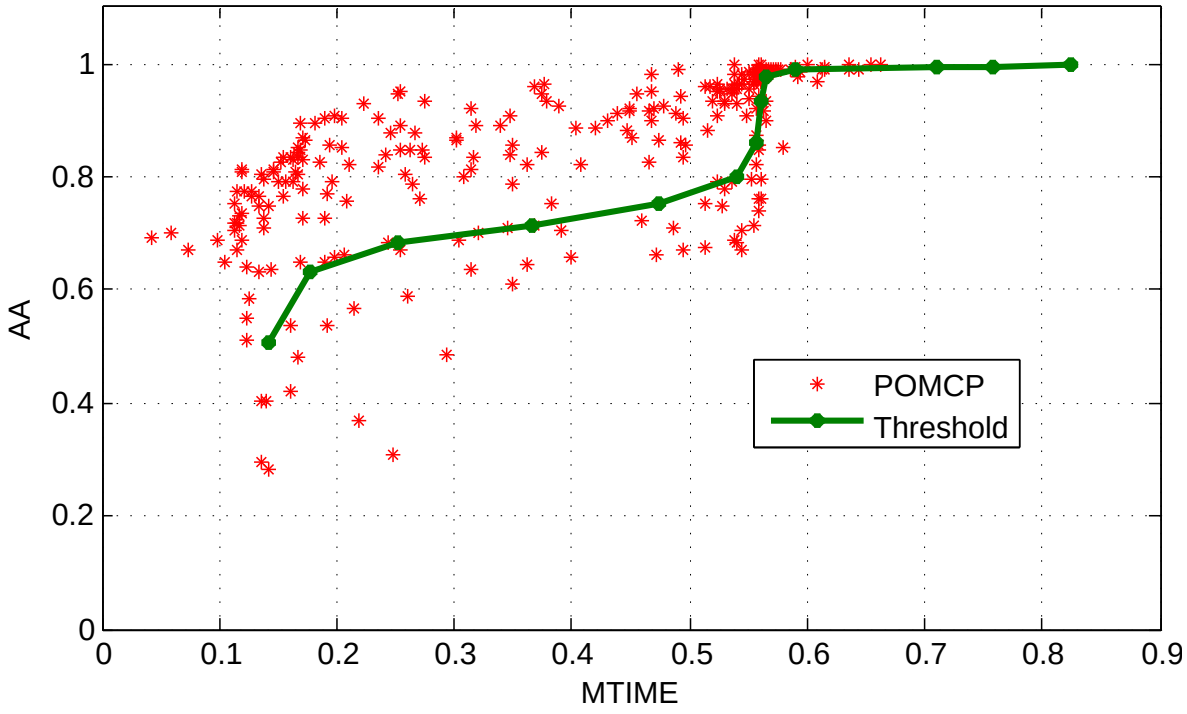

Figure 7. Acceptance accuracy vs. monitoring time.

On the other hand, POMDP-based monitors are configured by the assignment of reward values. For the purpose of the experiment we have used large variety of reward values. Ev-

ery combination of rewards was chosen to match a particular value of rejection threshold in threshold-based monitor with horizon 1 as described in Section 5.4.1. While one of rewards was fixed to the value of 1, we varied other rewards to consider cases with different ratio between them. The goal of this experiment was not to show a specific pattern in the assignment of rewards to improve monitoring performance with respect to threshold-based monitors. Instead, we aimed to show that monitoring POMDP may represent policies that consider special properties of the monitored system and use those properties to gain the advantage. Results shown in Figure 7 show the relation between the $MTIME$ and AA for the set of explored monitors. Every red star represents a single POMDP-based monitor implemented via POMCP. The green curve represents the performance measures for the threshold-based monitor with horizon 1. Our results show that many of configurations that have been chosen for the experiments, indeed, result in a better accuracy for the same monitoring time. While the threshold-based monitor may achieve an arbitrary value of AA, we were able to find those configurations of POMDP-based monitor that are able to achieve the same AA with lower values of $MTIME$.

This experimental evaluation confirms that POMDP-based approach is a promising direction in designing efficient monitoring algorithms.

CHAPTER 9

FUTURE WORK

In this paper, we were primarily focused on studying one particular approach, namely POMDP, when it is applied towards solving the monitoring problem of CPS. At every step of our research, we were either comparing the performance measures of one POMDP-based monitor with the other or with a class of threshold-based monitors. However, obviously, the total space of possible decision theoretic monitors is not limited only by POMDP and threshold-based monitors. In fact, this space of monitors is unbounded. We may ask a lot of research questions that have not yet been addressed in this paper, but for now, we will only describe a few of those, which we think are interesting and promising for the relevant research community.

In further sections, we suggest considering alternative monitoring rules than those studied in this paper. We present a few ideas that we think may work well in practice, and we discuss potential challenging questions that need to be studied further. Additionally, we bring attention to the topic known as Inverse Reinforcement Learning (IRL) and explain how it may be applied towards the problem of monitoring.

9.1 Alternative Monitoring Decision Rules

9.1.1 Adding More Non-Determinism into the Monitoring Decision Rule

Say, we are given a finite set of monitors $S_M = \{M_1, M_2, \dots\ M_n\}$, such that the corresponding performance measures are known. The acceptance accuracy of the monitor M_i is $AA(M_i)$,

the rejection accuracy is $RA(M_i)$ and the monitoring time is $MTIME(M_i)$. For each monitor M_i we assign a probability value $Pr(M_i)$, such that $\sum_{i=1}^{n} Pr(M_i) = 1$. Every time, right before the execution is about to start, the value of $Pr(M_i)$ determines whether the monitor M_i will be used exclusively to monitor the upcoming run.

Let M represent the monitor that combines decision rules of monitors from the set S_M in a non-deterministic fashion. According to the definition of performance measures it is easy to see that

$$\begin{aligned} AA(M) &= \sum_{i=1}^{n} Pr(M_i)AA(M_i) \\ RA(M) &= \sum_{i=1}^{n} Pr(M_i)RA(M_i) \\ MTIME(M) &= \sum_{i=1}^{n} Pr(M_i)MTIME(M_i) \end{aligned} \tag{9.1}$$

Note, that if it is known that every monitor in set S_M is a conservative monitor, then resulting non-deterministic monitor M would also be a conservative monitor, and, therefore, $RA(M) = 1$.

Different assignments of the probability values $Pr(M_i)$ may result in different values of accuracy and monitoring time. However, it is more important to question whether there exists an assignment of $Pr(M_i)$ such that the monitor M is better than some of monitors in S_M, or if it is better than some other monitor, which is not presented in S_M. This answer may be given by solving a set of linear inequalities, but it obviously depends on the monitored system itself and the way every monitor M_i is chosen.

To clarify it further, let's assume that $S_M = \{M_{rtr_1}, M_{rtr_2}\}$, where monitors M_{rtr_1} and M_{rtr_2} are threshold-based monitors with rejection thresholds rtr_1 and rtr_2 respectively. Let's

further assume that $rtr_1 < rtr_2$, and, therefore, according to the definition and properties of threshold-based monitors $AA(M_{rtr_1}) \leq AA(M_{rtr_2})$, while $MTIME(M_{rtr_1}) \geq MTIME(M_{rtr_2})$. An interesting question is whether we may be able to chose the value of $p = Pr_{M_{rtr_1}}$ in such a way that resulting performance of the monitor M is better than in a reference threshold-based monitor M_{ref} with the value of threshold $rtr_{ref} \in [rtr_1, rtr_2]$. Remember, that we improve performance of monitors by increasing the acceptance accuracy and reducing the monitoring time. It is easy to show that $AA_M \geq AA_{M_{ref}}$ and $MTIME_M \leq MTIME_{M_{ref}}$ if the following conditions are satisfied:

$$\begin{aligned} \frac{AA_{rtr_2} - AA_{ref}}{AA_{ref} - AA_{rtr_1}} &\geq \frac{p}{1-p} \\ \frac{MTIME_{rtr_2} - MTIME_{ref}}{MTIME_{ref} - MTIME_{rtr_1}} &\leq \frac{p}{1-p} \end{aligned} \tag{9.2}$$

From Equation 9.2 it may be concluded that in order for the non-deterministic monitor M to perform better than the reference monitor M_{ref} it is important for the AA and $MTIME$ change accordingly with respect to the threshold value. The bigger question is whether this condition might be satisfied for some systems.

9.1.2 Combining Multiple Decision Rules

For the implementation of a threshold-based monitor an important role plays the type of approximation that is used to compute the acceptance probability $AccProb(\alpha)$ for each every possible sequence of perceived outputs α. The value of $AccProb(\alpha)$ may be approximated by the finite horizon computations given in Section 3.5. By definition $AccProb^{h+1}(\alpha) \leq AccProb^{h}(\alpha)$,

i.e. by increasing horizon the value of acceptance probability become smaller. However, the velocity at which the value of $AccProb^h(\alpha)$ will converge depends on the actual value of α and inherent properties of the system model.

Here we propose a pair of monitor classes, which are similar to the finite horizon threshold-based monitors, but instead of working with one condition we combine multiple to boost the monitoring performance.

Consider the monitor M, which is a horizon 1 threshold-based monitor with rejection decision rule defined as $AccProb^1(\alpha) \leq a_{tr}$. Our goal is to define the new class of monitors that improve either both AA and $MTIME$, or keep one constant, while improving the other one.

Let the monitor M_1 declare the following rejection decision rule:

$$\begin{aligned} &AccProb^1(\alpha) \leq a_{tr_1} \vee AccProb^2(\alpha) \leq a_{tr_2} \\ &\text{, where } 0 \leq a_{tr_2} < a_{tr_1} \leq 1 \end{aligned} \tag{9.3}$$

Monitor M_1 attempts to keep the value of acceptance accuracy AA_{M_1} equal or as close as possible to the reference acceptance accuracy AA_M. Total portion of good executions that are rejected by M_1, but are not rejected by M is limited by the condition $AccProb^2(\alpha) \leq a_{tr_2}$, where a_{tr_2} has to be as small as possible. Note that we are talking about good executions here, and it might be reasonable to have an expectation that the model is well defined in a sense that acceptance probability does not drop to a significantly small value.

At the same time we are guarantied that every bad execution that is rejected by M, will be rejected by M_1, and it will either happen simultaneously, or earlier in the case if $AccProb^2(\alpha) \leq a_{tr_2}$ is satisfied before $AccProb^1(\alpha) \leq a_{tr_1}$.

Now, let's consider the other monitor M_2, which declares the following rejection decision rule:

$$\begin{aligned} &\big(AccProb^1(\alpha) \leq a_{tr_1} \wedge AccProb^2(\alpha) \leq a_{tr_1} - \delta\big) \vee AccProb^2(\alpha) \leq a_{tr_2} \\ &\text{, where } 0 \leq \delta \leq a_{tr_1} \text{ and } 0 \leq a_{tr_2} < a_{tr_1} \leq 1 \end{aligned} \tag{9.4}$$

In order to reduce number of false alarms raised by the monitor M_2 we have added a condition $AccProb^2(\alpha) \leq a_{tr_1} - \delta$. Our hypothesis is that good executions generally should not lead to a significant drop in acceptance probability when deeper horizon is used in the computations. On the other hand the same condition affects timing for rejection of bad executions. Although $AccProb^2(\alpha) \leq a_{tr_2}$ should help in rejecting earlier than M, a subset of executions might have a delayed rejection depending on the value of δ. Our initial experiments have confirmed that for the right selection of δ we are able to achieve improvement in monitoring performance as it is shown in Figure 8. Value of δ have been obtained by studying the change in $AccProb^2(\alpha)$ in good and bad executions.

Discussion around the monitors M_1, M_2 and their performance compared to M was given in order to illustrate that by combining multiple decision rules in one we are able to maintain performance measures around reference point, while using benefits of multiple decision rules to improve in certain aspects of monitoring. Outcome in both cases highly depends on what

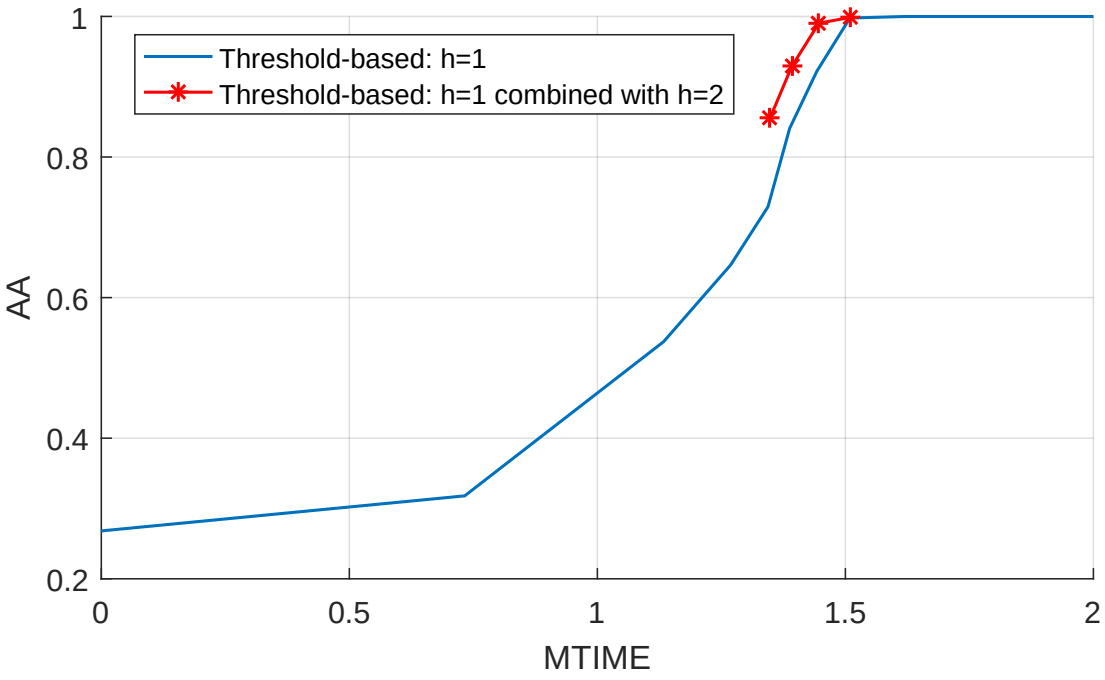

Figure 8. Monitoring performance when combining multiple rejection conditions.

maybe expected from the system executions and the way state probabilities evolve while system is running.

9.2 Inverse Reinforcement Learning for Monitoring-POMDP

The Inverse Reinforcement Learning (IRL) is aimed to recover the reward function based on the behavior of the intelligent agent. It was first introduced in [53] and has been further studied when applied to the MDP [54–56] and POMDP [57, 58] rewards.

Details of the IRL algorithms will be omitted in this paper and reader may find the answers in literature. As the name suggests the IRL is focused on solving an inverse problem for the reinforcement learning. While in the reinforcement learning the goal is to identify an optimal policy, the IRL receives the policy as an input and works to recover the reward function that has to be used to obtain that policy.

In this paper we have studied the effect of reward function of the monitoring POMDP on the policy, and while we have seen POMDP work well in few scenarios, we still have not fully solved the challenging problem of how to set up the rewards to obtain the best performance of the POMDP-based monitor. On the other hand defining the POMDP policy is a relatively easy problem. For the monitoring problem, this is simply the rejection or acceptance decision rule parameterized by the belief state or a sequence of outputs. The hard problem is to define the decision rule that would be able to consider all the aspects of the monitored system to produce the best possible performance.

CHAPTER 10

CONCLUSIONS

In this thesis, we have studied the subject of monitoring of CPS with decision-theoretic perspective. Our work was motivated by the results and ideas of traditional threshold-based monitors studied before in [8, 9, 44]. When a safety property of any particular system is monitored the main goal is to guarantee the expected performance measures, such as acceptance accuracy, rejection accuracy and the monitoring time. Our work was focused on improving these parameters over the traditional approaches.

We used the decision-theoretic formalism in the form of POMDPs to declare and formulate monitors of safety properties in CPS. Although the decision rule in monitoring-POMDPs is well defined, it is a challenging problem to find the optimal POMDP policy for cases when the system is driven by continuous dynamics. We have tailored the POMCP approach for computing the policy of monitoring-POMDPs online. The monitoring-POMCP algorithm given in Section 4.3 does not require explicitly defined transition and observation probability functions and only depends on the black-box of the system model, which is easy to implement using the PHS model of the monitored CPS.

Although we were able to compute an approximation to the optimal policy in monitoring-POMDPs, that policy does not always produce the same set of monitoring performance measures. We have shown that outcome of the decision rule in monitoring-POMDPs depends on the selection of rewards, that define the POMDP reward function. Every assignment of re-

wards tuple produces a different monitoring-POMDP. We have shown that certain assignments of rewards always produce identical monitoring performance, and concluded that monitoring-POMDPs have 3 degrees of freedom. Further, we were able to prove that the rejection accuracy in monitoring-POMDPs is always equal to 1, which is equivalent to saying that every failing execution is always eventually rejection. Thus, POMDP-based monitors represent a class of conservative monitors.

By studying assignments of rewards we identified POMDP-based monitors that are trivial, i.e. either accept or reject all executions. We have shown that for the case when POMDP-based monitor policy is computed with horizon 1 it is always equivalent to the threshold-based monitor also computed with horizon 1. Further, we identified classes of POMDP-based monitors that don't improve the monitoring performance compared to the class of threshold-based monitors computed with horizon 1.

We have seen in Chapter 5 that analyzing the performance of POMDP-based monitors for the general class of systems is a challenging task. To simplify the problem we study the application of POMDP-based monitors for the class of systems containing TSCCs. For these systems, under certain assumptions, we were able to solve the POMDP explicitly and present the decision rule in the form directly comparable to threshold-based monitors. This result has shown that POMDP-based monitors may take into consideration certain properties of the monitored system to better distinguish bad and good executions, and, therefore, improve the set of monitoring performance measures.

We conducted an experimental evaluation using the relatively complex example of a transmission system. With the experiment, we were able to confirm that monitoring-POMCP algorithm that we proposed in Section 4.3 is efficient to be applied for online monitoring. Our results have confirmed that POMDP-based monitors may be configured so that resulting performance measures are improved simultaneously, rather than independently as in traditional threshold-based monitors.

We have developed the DTMT, a software tool for the analysis of various monitoring approaches and for the online monitoring of safety properties in CPS. The tool architecture is fully extendable to support arbitrary monitoring decision rule and system model. The tool has been published in an online source code repository for the public use.

APPENDICES

Appendix A

INVARIANCE OF POMDP POLICY WITH RESPECT TO A CONSTANT APPENDED TO THE REWARD FUNCTION

Consider a general POMDP defined as a tuple $(S, A, T, R, O, Z, \gamma)$ as defined in Section 2.2.3. For simplicity, we will assume that both S and O are finite. The optimal policy π^* of the POMDP is defined as follows:

$$\begin{aligned} \pi^*(b) &= \arg\max_{a \in A} Q(b,a) \\ Q(b,a) &= \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in O} Pr(o|a,b) V^*(b_o^a) \\ V^*(b) &= \max_{a \in A} Q(b,a) \end{aligned} \tag{A.1}$$

Now, consider another POMDP, which is fully identical to the original POMDP with the only difference in the reward function. Let the newly defined reward function R_+ be defined by addition of a constant value $c \in \mathbb{R}$, i.e.

$$R_+(s,a) = R(s,a) + c \tag{A.2}$$

Lemma 2. *The optimal policy of any POMDP is invariant to the operation of addition of a constant $c \in \mathbb{R}$ to the reward function.*

Appendix A (Continued)

Proof. Let $Q_+(b,a)$ represent the value function of the POMDP with the modified reward function R_+.

$$\begin{aligned} Q_+(b,a) &= \sum_{s\in S} b(s)R_+(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a) \\ V_+^*(b) &= \max_{a\in A} Q_+(b,a) \end{aligned} \tag{A.3}$$

According to the definition of $R_+(s,a)$, we may obtain:

$$\begin{aligned} Q_+(b,a) &= \sum_{s\in S} b(s)\big(R(s,a)+c\big) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a) = \\ &= c\sum_{s\in S} b(s) + \sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a) = \\ &= c + \sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a) \\ V_+^*(b) &= \max_{a\in A}\Big(c + \sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a)\Big) = \\ &= c + \max_{a\in A}\Big(\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V_+^*(b_o^a)\Big) \end{aligned} \tag{A.4}$$

It is easy to see that every iteration of $V_+^*(b,a)$ produces additional constant c that factors out, and is multiplied with the discount factor γ on the previous step of recursion. Given the definition of initial $V^*(b,a)$ it is easy to see how value functions are related in compared POMDPs:

$$\begin{aligned} V_+^*(b) &= c + \gamma c + \gamma^2 c + \cdots + \gamma^\infty c + \\ &+ \max_{a\in A}\Big(\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V^*(b_o^a)\Big) = \\ &= c + \gamma c + \gamma^2 c + \cdots + \gamma^\infty c + V^*(b) \end{aligned} \tag{A.5}$$

Appendix A (Continued)

In the case if we assume infinite horizon for the policy calculation, then it is safe to assume that $\gamma < 1$, and therefore

$$V_+^*(b) \quad = \quad \frac{c}{1-\gamma} + V^*(b) \tag{A.6}$$

Otherwise, if the horizon is finite and is equal to h, then we may assume that $\gamma = 1$. Thus we obtain,

$$V_+^h(b) \quad = \quad hc + V^h(b) \tag{A.7}$$

Since in both cases, when the horizon is infinite and finite, the added value in the expression for $V_+^*(b, a)$ is independent of an action, we may directly claim that policies π^* and π_+^* are equivalent.

□

Appendix B

INVARIANCE OF POMDP POLICY WITH RESPECT TO A CONSTANT MULTIPLIED WITH THE REWARD FUNCTION

Consider a general POMDP defined as a tuple $(S, A, T, R, O, Z, \gamma)$ as defined in Section 2.2.3. For simplicity, we will assume that both S and O are finite. The optimal policy π^* of the POMDP is defined as follows:

$$\begin{aligned} \pi^*(b) &= \arg\max_{a \in A} Q(b,a) \\ Q(b,a) &= \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in O} Pr(o|a,b) V^*(b_o^a) \\ V^*(b) &= \max_{a \in A} Q(b,a) \end{aligned} \tag{B.1}$$

Now, consider another POMDP, which is fully identical to the original POMDP with the only difference in the reward function. Let the newly defined reward function $R_\times$ be defined by multiplying the reward function R with a positive constant $c \in \mathbb{R}_{>0}$, i.e.

$$R_\times(s,a) = cR(s,a) \tag{B.2}$$

Lemma 3. *The optimal policy of any POMDP is invariant to the operation of multiplication of a constant value $c \in \mathbb{R}_{>0}$ with the reward function.*

Appendix B (Continued)

Proof. Let $Q_{\times}(b,a)$ represent the value function of the POMDP with the modified reward function $R_{\times}$.

$$\begin{aligned} Q_{\times}(b,a) &= \sum_{s\in S} b(s)R_{\times}(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V^*_{\times}(b^a_o) \\ V^*_{\times}(b) &= \max_{a\in A} Q_{\times}(b,a) \end{aligned} \tag{B.3}$$

According to the definition of $R_{\times}(s,a)$, we may obtain:

$$\begin{aligned} Q_{\times}(b,a) &= c\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V^*_{\times}(b^a_o) = \\ V^*_{\times}(b) &= \max_{a\in A}\Big(c\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V^*_{\times}(b^a_o)\Big) = \\ &= c\max_{a\in A}\Big(\sum_{s\in S} b(s)R(s,a) + \tfrac{1}{c}\gamma \sum_{o\in O} Pr(o|a,b)V^*_{\times}(b^a_o)\Big) \end{aligned} \tag{B.4}$$

The expression for the $V^*_{\times}(b)$ is defined as recursion, and it is easy to see that $\frac{1}{c}$ will cancel out by multiplication with c that will be caused by the further execution of $V^*_{\times}(b)$. Thus we may represent the updated value function with the original value function as follows:

$$\begin{aligned} V^*_{\times}(b) &= c\max_{a\in A}\Big(\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in O} Pr(o|a,b)V^*(b^a_o)\Big) \\ &= cV^*(b) \end{aligned} \tag{B.5}$$

Since the change in the value function is independent of the POMDP action we directly may claim that policies π^* and $\pi^*_{\times}$ are equivalent. □

Appendix C

POMDP VALUE FUNCTIONS FOR SYSTEMS WITH TERMINAL STRONGLY CONNECTED COMPONENTS

To derive the POMDP value function for systems with *TSCCs* we use the notations and definitions introduced in Section 6.2.

The general equations defining value function are:

$$
\begin{aligned}
Q^h(b_{\alpha o}, \mathit{Alarm}) &= G^a \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s - L^a \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s \\
Q^h(b_{\alpha o}, \mathit{Continue}) &= -L^c \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s + G^c \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s + \\
&+ \sum_{o_i \in O} Pr(o_i|\mathit{Continue}, b_{\alpha o}) V^{h-1}(b_{\alpha o o_i}) \qquad \text{(C.1)} \\
V^h(b_\alpha) &= \begin{cases} \max\limits_{a \in A} Q^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned}
$$

Our first step is to define the value of $Pr(o_i|\mathit{Continue}, b_{\alpha o})$ that is a part of right-hand side of the equation for the value function of action *Continue*. In order to do so we will work on more general representation of probability value $Pr(o_n|\mathit{Continue}, b_{\alpha o o_1 \ldots o_{n-1}})$, were we will assume that after an already perceived observation sequence αo we expect a sequence of observations of $o_1 \ldots o_{n-1}$. We are interested in the observation o_n that will potentially follow. We will

Appendix C (Continued)

label probability as *good* or *bad* to identify that it relates to observation being perceived in *good* or *bad PTSCC* respectively. For simplicity we will represent $Pr(o_n|Continue, b_{\alpha o o_1 \dots o_{n-1}}) = Pr(o_n|b_{\alpha o o_1 \dots o_{n-1}})$.

$$\begin{aligned}
Pr(o_n|b_{\alpha o o_1 \dots o_{n-1}}) &= \beta(\alpha o o_1 \dots o_{n-1}) \times \\
&\times \left(Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) + Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}})\right) \\
Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) &= \int\limits_{s \in C_o^{good}} \int\limits_{s_1 \in C_{o_1}^{good}} \cdots \int\limits_{s_n \in C_{o_n}^{good}} Pr(s|\alpha o) \times \\
&\times Pr(s, s_1) \dots Pr(s_{n-1}, s_n)\, \mathrm{d}s\, \mathrm{d}s_1 \dots \mathrm{d}s_n \\
Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) &= \int\limits_{s \in C_o^{bad}} \int\limits_{s_1 \in C_{o_1}^{bad}} \cdots \int\limits_{s_n \in C_{o_n}^{bad}} Pr(s|\alpha o) \times \\
&\times Pr(s, s_1) \dots Pr(s_{n-1}, s_n)\, \mathrm{d}s\, \mathrm{d}s_1 \dots \mathrm{d}s_n
\end{aligned} \tag{C.2}$$

$\beta(\alpha o o_1 \dots o_{n-1})$ is a normalizing coefficient defined as:

$$\begin{aligned}
\beta(\alpha o o_1 \dots o_{n-1}) &= \frac{1}{\sum\limits_{o_n \in O} \left(Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) + Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}})\right)} = \\
&= \frac{1}{\sum\limits_{o_n \in O} Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) + \sum\limits_{o_n \in O} Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}})}
\end{aligned} \tag{C.3}$$

Appendix C (Continued)

Note, that

$$\sum_{o_n \in O} Pr_{good}(o_n | b_{\alpha o o_1 \dots o_{n-1}}) =$$

$$= \sum_{o_n \in O} \int_{s \in C_o^{good}} \int_{s_1 \in C_{o_1}^{good}} \cdots \int_{s_n \in C_{o_n}^{good}} Pr(s|\alpha o) Pr(s, s_1) \dots Pr(s_{n-1}, s_n) \, \mathrm{d}s \, \mathrm{d}s_1 \dots \, \mathrm{d}s_n =$$

$$= \int_{s \in C_o^{good}} \int_{s_1 \in C_{o_1}^{good}} \cdots \int_{s_{n-1} \in C_{o_{n-1}}^{good}} Pr(s|\alpha o) Pr(s, s_1) \dots \underbrace{\sum_{o_n \in O} \int_{s_n \in C_{o_n}^{good}} Pr(s_{n-1}, s_n)}_{= \int_{s_n \in C_{o_n}^{good}} Pr(s_{n-1}, s_n) = 1 \, \mathrm{d}s_n} \, \mathrm{d}s \, \mathrm{d}s_1 \dots \, \mathrm{d}s_n =$$

$$= \int_{s \in C_o^{good}} \int_{s_1 \in C_{o_1}^{good}} \cdots \int_{s_{n-1} \in C_{o_{n-1}}^{good}} Pr(s|\alpha o) Pr(s, s_1) \dots Pr(s_{n-2}, s_{n-1}) \, \mathrm{d}s \, \mathrm{d}s_1 \dots \, \mathrm{d}s_{n-1} =$$

$$= Pr_{good}(o_{n-1} | b_{\alpha o o_1 \dots o_{n-2}}) \tag{C.4}$$

Similarly,

$$\sum_{o_n \in O} Pr_{bad}(o_n | b_{\alpha o o_1 \dots o_{n-1}}) = Pr_{bad}(o_{n-1} | b_{\alpha o o_1 \dots o_{n-2}}) \tag{C.5}$$

Therefore based on Equation C.3, Equation C.4, and Equation C.5,

$$Pr_{good}(o_n | b_{\alpha o o_1 \dots o_{n-1}}) + Pr_{bad}(o_n | b_{\alpha o o_1 \dots o_{n-1}}) = \frac{1}{\beta(\alpha o o_1 \dots o_n)} \tag{C.6}$$

$$\beta(\alpha o o_1 \dots o_n) = \frac{1}{Pr_{good}(o_n | b_{\alpha o o_1 \dots o_{n-1}}) + Pr_{bad}(o_n | b_{\alpha o o_1 \dots o_{n-1}})} \tag{C.7}$$

$$Pr(o_n | b_{\alpha o o_1 \dots o_{n-1}}) \quad = \quad \frac{\beta(\alpha o o_1 \dots o_{n-1})}{\beta(\alpha o o_1 \dots o_n)} \tag{C.8}$$

Appendix C (Continued)

It's worth noting here that $\beta(\alpha o) = 1$ and $Pr(o_1|\alpha o) = \frac{1}{\beta(b_{\alpha o o_1})}$. Indeed, by rewriting Equation C.3 we get,

$$\begin{aligned}
\beta(\alpha o o_1) &= \frac{1}{\int\limits_{s \in C_o^{good}} Pr(s|b_{\alpha o})\,\mathrm{d}s + \int\limits_{s \in C_o^{bad}} Pr(s|b_{\alpha o})\,\mathrm{d}s} = \\
&= \frac{1}{\int\limits_{s \in C^{good}} Pr(s|b_{\alpha o})\,\mathrm{d}s + \int\limits_{s \in C^{bad}} Pr(s|b_{\alpha o)}\,\mathrm{d}s} = \\
&= 1
\end{aligned} \tag{C.9}$$

Now, let's focus on the representation of $\int\limits_{s \in C^{good}} Pr(s|\alpha o o_1 \ldots o_n)\,\mathrm{d}s$ and $\int\limits_{s \in C^{bad}} Pr(s|\alpha o o_1 \ldots o_n)\,\mathrm{d}s$.

$$\begin{aligned}
\int\limits_{s \in C^{good}} Pr(s|\alpha o o_1 \ldots o_n)\,\mathrm{d}s &= \gamma(\alpha o o_1 \ldots o_n) Pr_{good}(o_n|b_{\alpha o o_1 \ldots o_{n-1}}) \\
\int\limits_{s \in C^{bad}} Pr(s|\alpha o o_1 \ldots o_n)\,\mathrm{d}s &= \gamma(\alpha o o_1 \ldots o_n) Pr_{bad}(o_n|b_{\alpha o o_1 \ldots o_{n-1}})
\end{aligned} \tag{C.10}$$

$\gamma(\alpha o o_1 \ldots o_n)$ is a normalizing coefficient defined as follows:

$$\gamma(\alpha o o_1 \ldots o_n) = \frac{1}{Pr_{good}(o_n|b_{\alpha o o_1 \ldots o_{n-1}}) + Pr_{bad}(o_n|b_{\alpha o o_1 \ldots o_{n-1}})} \tag{C.11}$$

However, as we can see

$$\gamma(\alpha o o_1 \ldots o_n) = \beta(\alpha o o_1 \ldots o_n) \tag{C.12}$$

Appendix C (Continued)

Thus we may rewrite Equation C.10 as follows:

$$\begin{aligned} \int\limits_{s\in C^{good}} Pr(s|\alpha o o_1 \dots o_n)\,\mathrm{d}s &= \beta(\alpha o o_1 \dots o_n) Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) \\ \int\limits_{s\in C^{bad}} Pr(s|\alpha o o_1 \dots o_n)\,\mathrm{d}s &= \beta(\alpha o o_1 \dots o_n) Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) \end{aligned} \tag{C.13}$$

Given the representation in Equation C.8, Equation C.13 and Equation C.12 we may rewrite POMDP value functions. To simplify the notations we will assume symbol β_n where we use a function $\beta(\alpha o o_1 \dots o_n)$. First we start from the value function of the action Alarm:

$$\begin{aligned} & Q^h(b_{\alpha o o_1 \dots o_n}, Alarm) = \\ = \; & G^a \int\limits_{s\in S_{bad}} Pr(s|\alpha o o_1 \dots o_n)\,\mathrm{d}s - L^a \int\limits_{s\in S_{bad}} Pr(s|\alpha o o_1 \dots o_n)\,\mathrm{d}s = \\ = \; & G^a \beta_n Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) - L^a \beta_n Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) = \\ = \; & \beta_n \big(G^a Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) - L^a Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}})\big) \end{aligned} \tag{C.14}$$

Similar substitution in the value function of the action *Continue* leads to the following equation:

$$\begin{aligned} & Q^h(b_{\alpha o o_1 \dots o_n}, Continue) = \\ = \; & \beta_n \big(G^a Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) - L^a Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}})\big) + \\ + \; & \sum_{o_{n+1}\in O} \tfrac{\beta_n}{\beta_{n+1}} V^{h-1}(b_{\alpha o o_1 \dots o_{n+1}}) = \\ = \; & \beta_n \big(G^a Pr_{bad}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) - L^a Pr_{good}(o_n|b_{\alpha o o_1 \dots o_{n-1}}) + \\ + \; & \sum_{o_{n+1}\in O} \tfrac{1}{\beta_{n+1}} V^{h-1}(b_{\alpha o o_1 \dots o_{n+1}})\big) \end{aligned} \tag{C.15}$$

Appendix C (Continued)

By definition of function $V^h(\alpha)$ given in Equation C.1 its value is equal to 0 only when $h = 0$. For all other values of h it depends on the value of action *Alarm* and *Continue*. It is obvious from Equation C.14 and Equation C.15 that the value of actions *Alarm* and *Continue* is proportional to β_n. Therefore, by using a simple induction it may be shown that value functions of monitoring POMDP defined for class of systems with *TSCCs* are:

$$
\begin{aligned}
Q^h(b_{\alpha o}, \mathit{Alarm}) &= G^a \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s - L^a \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s \\
Q^h(b_{\alpha o}, \mathit{Continue}) &= -L^c \int\limits_{s \in C_o^{bad}} Pr(s|\alpha o)\,\mathrm{d}s + G^c \int\limits_{s \in C_o^{good}} Pr(s|\alpha o)\,\mathrm{d}s + \\
&+ \sum_{o' \in O} \bar{V}^{h-1}(b_{\alpha o o'}) \qquad \text{(C.16)} \\
\bar{V}^h(b_\alpha) &= \begin{cases} \max\limits_{a \in A} \bar{Q}^h(b_\alpha, a) & h > 0 \\ 0 & h = 0 \end{cases}
\end{aligned}
$$

Appendix C (Continued)

$$
\begin{aligned}
& \bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, \mathit{Alarm}) && = \\
= \; & G^a \int\limits_{s \in C_o^{bad}} \int\limits_{s_1 \in C_{o_1}^{bad}} \cdots \int\limits_{s_k \in C_{o_k}^{bad}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k) \, \mathrm{d}s \, \mathrm{d}s_1 \ldots \, \mathrm{d}s_k && - \\
- \; & L^a \int\limits_{s \in C_o^{good}} \int\limits_{s_1 \in C_{o_1}^{good}} \cdots \int\limits_{s_k \in C_{o_k}^{good}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k) \, \mathrm{d}s \, \mathrm{d}s_1 \ldots \, \mathrm{d}s_k && \\
& \bar{Q}^h(b_{\alpha o o_1 \ldots o_k}, \mathit{Continue}) && = \\
= \; & -L^c \int\limits_{s \in C_o^{bad}} \int\limits_{s_1 \in C_{o_1}^{bad}} \cdots \int\limits_{s_k \in C_{o_k}^{bad}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k) \, \mathrm{d}s \, \mathrm{d}s_1 \ldots \, \mathrm{d}s_k && + \\
+ \; & G^c \int\limits_{s \in C_o^{good}} \int\limits_{s_1 \in C_{o_1}^{good}} \cdots \int\limits_{s_k \in C_{o_k}^{good}} Pr(s|\alpha o) Pr(s, s_1) \ldots Pr(s_{k-1}, s_k) \, \mathrm{d}s \, \mathrm{d}s_1 \ldots \, \mathrm{d}s_k && + \\
+ \; & \sum_{o_{k+1} \in O} \bar{V}^{h-1}(b_{\alpha o o_1 \ldots o_k o_{k+1}}) &&
\end{aligned}
$$

Appendix D

COPYRIGHT PERMISSIONS

This appendix includes copyright permissions for the RV'16 article [1], whose contents were reused in this thesis.

SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Mar 15, 2018

This Agreement between Andrey Yavolovsky ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	4310070429626
License date	Mar 15, 2018
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Decision-Theoretic Monitoring of Cyber-Physical Systems
Licensed Content Author	Andrey Yavolovsky, Miloš Žefran, A. Prasad Sistla
Licensed Content Date	Jan 1, 2016
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	501 to 1000
Author of this Springer Nature content	yes
Title	Decision-Theoretic Monitoring of Cyber-Physical Systems
Instructor name	Andrey Yavolovsky
Institution name	University of Illinois at Chicago
Expected presentation date	Mar 2018
Requestor Location	Andrey Yavolovsky 16608 NE 37th St Apt U2063 REDMOND, WA 98052 United States Attn: Andrey Yavolovsky
Billing Type	Invoice
Billing Address	Andrey Yavolovsky 16608 NE 37th St Apt U2063 REDMOND, WA 98052 United States Attn: Andrey Yavolovsky
Total	0.00 USD
Terms and Conditions	

Springer Nature Terms and Conditions for RightsLink Permissions

Springer Customer Service Centre GmbH (the Licensor) hereby grants you a non-exclusive, world-wide licence to reproduce the material and for the purpose and requirements specified in the attached copy of your order form, and for no other use, subject to the conditions below:

1. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of this material. However, you should ensure that the material you are requesting is original to the Licensor and does not carry the copyright of another entity (as credited in the published version).

   If the credit line on any part of the material you have requested indicates that it was reprinted or adapted with permission from another source, then you should also seek permission from that source to reuse the material.

2. Where **print only** permission has been granted for a fee, separate permission must be obtained for any additional electronic re-use.

3. Permission granted **free of charge** for material in print is also usually granted for any electronic version of that work, provided that the material is incidental to your work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.

4. A licence for 'post on a website' is valid for 12 months from the licence date. This licence does not cover use of full text articles on websites.

5. Where **'reuse in a dissertation/thesis'** has been selected the following terms apply: Print rights for up to 100 copies, electronic rights for use only on a personal website or institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/).

6. Permission granted for books and journals is granted for the lifetime of the first edition and does not apply to second and subsequent editions (except where the first edition permission was granted free of charge or for signatories to the STM Permissions Guidelines http://www.stm-assoc.org/copyright-legal-affairs/permissions/permissions-guidelines/), and does not apply for editions in other languages unless additional translation rights have been granted separately in the licence.

7. Rights for additional components such as custom editions and derivatives require additional permission and may be subject to an additional fee. Please apply to Journalpermissions@springernature.com/bookpermissions@springernature.com for these rights.

8. The Licensor's permission must be acknowledged next to the licensed material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract, and must be hyperlinked to the journal/book's homepage. Our required acknowledgement format is in the Appendix below.

9. Use of the material for incidental promotional use, minor editing privileges (this does not include cropping, adapting, omitting material or any other changes that affect the meaning, intention or moral rights of the author) and copies for the disabled are permitted under this licence.

10. Minor adaptations of single figures (changes of format, colour and style) do not require the Licensor's approval. However, the adaptation should be credited as shown in Appendix below.

Appendix — Acknowledgements:

For Journal Content:
Reprinted by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication)

For **Advance Online Publication papers:**
Reprinted by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].)

For Adaptations/Translations:
Adapted/Translated by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication)

Note: For any republication from the British Journal of Cancer, the following credit line style applies:

Reprinted/adapted/translated by permission from [**the Licensor**]: on behalf of Cancer Research UK: : [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT]** (year of publication)

For **Advance Online Publication** papers:
Reprinted by permission from The [**the Licensor**]: on behalf of Cancer Research UK: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM])

For Book content:
Reprinted/adapted by permission from [**the Licensor**]: [**Book Publisher** (e.g. Palgrave Macmillan, Springer etc) [**Book Title**] by [**Book author**(s)] [**COPYRIGHT**] (year of publication)

Other Conditions:

Version 1.0

CITED LITERATURE

1. Yavolovsky, A., Žefran, M., and Sistla, A. P.: Decision-theoretic monitoring of cyber-physical systems. In International Conference on Runtime Verification, pages 404–419. Springer, 2016.

2. NSF: Cyber-physical systems. `https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286`, 2015. [Online; accessed 10/23/2015].

3. Rajhans, A., Cheng, S.-W., Schmerl, B., Garlan, D., Krogh, B. H., Agbi, C., and Bhave, A.: An architectural approach to the design and analysis of cyber-physical systems. Electronic Communications of the EASST, 21, 2009.

4. ZHANG, Y., XIE, F., DONG, Y., YANG, G., and ZHOU, X.: High fidelity virtualization of cyber-physical systems. International Journal of Modeling, Simulation, and Scientific Computing, 4(02):1340005, 2013.

5. Baheti, R. and Gill, H.: Cyber-physical systems. The impact of control technology, pages 161–166, 2011.

6. Knight, J. C.: Safety critical systems: challenges and directions. In Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, pages 547–550. IEEE, 2002.

7. Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P.: What's decidable about hybrid automata? Journal of computer and system sciences, 57(1):94–124, 1998.

8. Sistla, A. P., Žefran, M., and Feng, Y.: Monitorability of stochastic dynamical systems. In Computer Aided Verification, eds. G. Gopalakrishnan and S. Qadeer, volume 6806 of Lecture Notes in Computer Science, pages 720–736. Springer Berlin / Heidelberg, 2011.

9. Sistla, A. P., Žefran, Miloš, and Feng, Y.: Runtime monitoring of stochastic cyber-physical systems with hybrid state. In 2nd International Conference on Runtime Verification (RV2011), 2011.

10. Sistla, A. P., Žefran, M., Feng, Y., and Ben, Y.: Timely monitoring of partially observable stochastic systems. In Proceedings of the 17th international conference on Hybrid systems: computation and control, pages 61–70. ACM, 2014.

11. Papoulis, A. and Pillai, S. U.: Probability, random variables, and stochastic processes. Tata McGraw-Hill Education, 2002.

12. Vardi, M. Y.: Automatic verification of probabilistic concurrent finite state programs. In Foundations of Computer Science, 1985., 26th Annual Symposium on, pages 327–338. IEEE, 1985.

13. Cappé, O. and Moulines, E.: T., rydén (2005): Inference in hidden markov models.

14. Rudin, W.: Real and complex analysis. Tata McGraw-Hill Education, 1987.

15. Hofbaur, M. and Williams, B.: Mode estimation of probabilistic hybrid systems. In Hybrid Systems: Computation and Control, volume 2289 of Lecture Notes in Computer Science, pages 81–91. Springer, 2002.

16. Kiefer, S. and Sistla, A. P.: Distinguishing hidden markov chains. In Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, pages 66–75. ACM, 2016.

17. Papoulis, A.: Random variables, and stochastic processes, 1990.

18. Russell, S. and Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 2009.

19. Howard, R. A.: Dynamic programming and Markov processes. Wiley for The Massachusetts Institute of Technology, 1964.

20. Puterman, M. L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, 2014.

21. Astrom, K. J.: Optimal control of markov processes with incomplete state information. Journal of Mathematical Analysis and Applications, 10(1):174, 1965.

22. Pineau, J., Gordon, G., and Thrun, S.: Anytime point-based approximations for large pomdps. Journal of Artificial Intelligence Research, pages 335–380, 2006.

23. DeCarlo, R. A., Branicky, M. S., Pettersson, S., and Lennartson, B.: Perspectives and results on the stability and stabilizability of hybrid systems. Proceedings of the IEEE, 88(7):1069–1082, 2000.

24. Alur, R., Henzinger, T. A., Lafferriere, G., and Pappas, G. J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE, 88(7):971–984, 2000.

25. van der Schaft, A. J. and Schumacher, J. M.: An introduction to hybrid dynamical systems, volume 251. Springer London, 2000.

26. Liberzon, D.: Switching in systems and control. Springer, 2003.

27. Pnueli, A.: Verifying liveness properties of reactive systems. In Hybrid and Real-Time Systems, volume 1201. New York, NY, Springer, 1997.

28. Koutsoukos, X., Kurien, J., and Zhao, F.: Estimation of distributed hybrid systems using particle filtering methods. In Hybrid Systems: Computation and Control, volume 2623 of Lecture Notes in Computer Science, pages 298–313. Springer, 2003.

29. Verma, V., Gordon, G., Simmons, R., and Thrun, S.: Real-time fault diagnosis. IEEE Robotics & Automation Magazine, 11(2):56–66, 2004.

30. Blom, H. and Bloem, E.: Particle filtering for stochastic hybrid systems. In 43rd IEEE Conference on Decision and Control, 2004. CDC, volume 3, 2004.

31. Lerner, U., Moses, B., Scott, M., McIlraith, S., and Koller, D.: Monitoring a complex physical system using a hybrid dynamic bayes net. In Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI), pages 301–310, 2002.

32. Clarke, E. M., Grumberg, O., and Peled, D.: Model checking. MIT press, 1999.

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

34. Pantelic, V., Postma, S., and Lawford, M.: Probabilistic supervisory control of probabilistic discrete event systems. IEEE Transactions on Automatic Control, 54(8):2013–2018, 2009.

35. Yoo, T. and Lafortune, S.: Polynomial-time verification of diagnosability of partially observed discrete-event systems. IEEE Transactions on Automatic Control, 47(9):1491–1495, 2002.

36. Stoller, S. D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S. A., and Zadok, E.: Runtime verification with state estimation. In Runtime Verification, eds. S. Khurshid and K. Sen, number 7186 in Lecture Notes in Computer Science, pages 193–207. Springer Berlin Heidelberg, September 2011.

37. Isard, M. and Blake, A.: Condensation—conditional density propagation for visual tracking. International journal of computer vision, 29(1):5–28, 1998.

38. Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J.: Where's waldo? sensor-based temporal logic motion planning. In Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 3116–3121. IEEE, 2007.

39. Lygeros, J., Godbole, D. N., and Sastry, S.: A game-theoretic approach to hybrid system design. In Hybrid systems III, pages 1–12. Springer, 1996.

40. Grunske, L. and Zhang, P.: Monitoring probabilistic properties. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pages 183–192. ACM, 2009.

41. Sammapun, U., Lee, I., and Sokolsky, O.: Rt-mac: runtime monitoring and checking of quantitative and probabilistic properties. In 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), pages 147–153. IEEE, 2005.

42. Pnueli, A., Zaks, A., and Zuck, L.: Monitoring interfaces for faults. Electronic Notes in Theoretical Computer Science, 144(4):73–89, 2006.

43. Margaria, T., Sistla, A. P., Steffen, B., and Zuck, L. D.: Taming interface specifications. In International Conference on Concurrency Theory, pages 548–561. Springer, 2005.

44. Sistla, A. P., Zhou, M., and Zuck, L. D.: Monitoring off-the-shelf components. In International Workshop on Verification, Model Checking, and Abstract Interpretation, pages 222–236. Springer, 2006.

45. Agate, R. and Seward, D.: Autonomous safety decision-making in intelligent robotic systems in the uncertain environments. In Fuzzy Information Processing Society, 2008. NAFIPS 2008. Annual Meeting of the North American, pages 1–6. IEEE, 2008.

46. Hsu, S.-P. and Arapostathis, A.: Safety control of partially observed MDPs with applications to machine maintenance problems. In Systems, Man and Cybernetics, 2004 IEEE International Conference on, volume 1, pages 261–265. IEEE, 2004.

47. Seward, D., Pace, C., and Agate, R.: Safe and effective navigation of autonomous robots in hazardous environments. Autonomous Robots, 22(3):223–242, 2007.

48. Bai, H., Hsu, D., Lee, W. S., and Ngo, V. A.: Monte carlo value iteration for continuous-state pomdps. In Algorithmic foundations of robotics IX, pages 175–191. Springer, 2010.

49. Silver, D. and Veness, J.: Monte-carlo planning in large pomdps. In Advances in neural information processing systems, pages 2164–2172, 2010.

50. Auer, P., Cesa-Bianchi, N., and Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2-3):235–256, 2002.

51. Boost: Boost. `http://www.boost.org/`, 2015. [Online; accessed 07/15/2017].

52. ROS: Ros. `http://www.ros.org/`, 2018. [Online; accessed 02/12/2018].

53. Russell, S.: Learning agents for uncertain environments. In Proceedings of the eleventh annual conference on Computational learning theory, pages 101–103. ACM, 1998.

54. Abbeel, P. and Ng, A. Y.: Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1. ACM, 2004.

55. Ng, A. Y., Russell, S. J., et al.: Algorithms for inverse reinforcement learning. In Icml, pages 663–670, 2000.

56. Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K.: Maximum entropy inverse reinforcement learning. In AAAI, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

57. Choi, J. and Kim, K.-E.: Inverse reinforcement learning in partially observable environments. Journal of Machine Learning Research, 12(Mar):691–730, 2011.

58. Zhifei, S. and Meng Joo, E.: A survey of inverse reinforcement learning techniques. International Journal of Intelligent Computing and Cybernetics, 5(3):293–311, 2012.

59. Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. Springer, 1993.

60. Billingsley, P.: Probability and measure. John Wiley & Sons, 2008.

61. Blom, H. A.: Stochastic hybrid processes with hybrid jumps. Analysis and Design of Hybrid System, pages 319–324, 2003.

62. Cassandra, A. R., Kaelbling, L. P., and Littman, M. L.: Acting optimally in partially observable stochastic domains. In AAAI, volume 94, pages 1023–1028, 1994.

63. Hespanha, J. P.: Stochastic hybrid systems: Application to communication networks. In Hybrid systems: computation and control, pages 387–401. Springer, 2004.

64. Hofbaur, M. and Williams, B.: Mode estimation of probabilistic hybrid systems. In Hybrid Systems: Computation and Control, volume 2289 of Lecture Notes in Computer Science, pages 81–91. Springer, 2002.

65. Lynch, N., Segala, R., Vaandrager, F., and Weinberg, H. B.: Hybrid i/o automata. Springer, 1996.

66. Pola, G., Bujorianu, M., Lygeros, J., and Benedetto, M.: Stochastic hybrid models: An overview. In Proc. IFAC conf. anal. design hybrid syst, pages 45–50, 2003.

VITA

Andrey Yavolovsky

Email: andrey.yavolovsky@gmail.com

Education

- Ph.D. Candidate, *Computer Science* 2010-2018

  University of Illinois at Chicago (UIC)

  GPA: 3.84/4.0

- M.S., *Electrical and Electronics Engineering* 2007-2009

  Ivane Javakhishvili Tbilisi State University (TSU)

  Thesis:

  Automation of measurement system for investigation of ESD influence to cables and devices.

  GPA: 4.0/4.0

- B.S., *Computer Science* 2002-2006

  Ivane Javakhishvili Tbilisi State University (TSU)

Publications

3. **A. Yavolovsky**, M. Žefran, and A. P. Sistla. "Decision-theoretic monitoring of cyber-physical systems." In International Conference on Runtime Verification, pp. 404-419. Springer, Cham, 2016.
2. M. Javaid, M. Žefran, and **A. Yavolovsky**. "Using pressure sensors to identify manipulation actions during human physical interaction." In Robot and Human Interactive Communication (RO-MAN), 2015 24th IEEE International Symposium on, pp. 670-675. IEEE, 2015.
1. A. Demurov, I. Oganezova, **A. Yavolovsky**, Z. Kuchadze, F. Bogdanov, and R. Jobava. "Analysis of EMC/EMI problems in printed circuit boards using Green's function approach." In Direct and Inverse Problems of Electromagnetic and Acoustic Wave Theory (DIPED), 2010 Xvth International Seminar/Workshop on, pp. 91-95. IEEE, 2010.

Awards

- The Graduate Student Presenter Award, UIC 2016
- The Graduate Student Council Travel Award, UIC 2015
- Computer Science Department Conference, 1st Place Award, TSU 2008
- Computer Science Department Conference, 3rd Place Award, TSU 2004

Presentations

- International Conference on Runtime Verification (RV'16) Oct 2016
- The Midwest Verification Day (MVD 2015) Oct 2015
- NSF 5th Annual Cyber-Physical Systems Principal Investigators' Meeting Nov 2014

Work Experience

- Software Engineer 2017 - Present
  Microsoft Corporation, Redmond, WA, United States
- Research Assistant 2010-2017
  Robotics Lab, UIC
- Teaching Assistant 2011-2015
  Computer Architecture I: Logic and Computer Structures (CS 266), UIC
  Computer Architecture II: Hardware-Software Interface (CS 366), UIC
  Advanced Computer Architecture (CS 466), UIC
  Computer Organization (CS 261), UIC
  Computer Design (CS 362), UIC
- Software Engineer Intern Summer 2014
  The MathWorks, Inc., Natick, MA, United States
- Software Enginner Intern Summer 2015
  Microsoft Corporation, Redmond, WA, United States
- Software Engineer 2003 - 2010
  EMCoS (EM Consulting and Software), Tbilisi, Georgia