Modeling and Design of Assured and Adaptive Cyber-Physical Systems

by

Mahmoud Elfar

Department of Electrical and Computer Engineering
Duke University

Date: ______________________
Approved:

__________________________
Miroslav Pajic, Advisor

__________________________
Krishnendu Chakrabarty

__________________________
Benjamin C. Lee

__________________________
Michael Zavlanos

__________________________
Mary L. Cummings

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University

2022
ABSTRACT

Modeling and Design of Assured and Adaptive Cyber-Physical Systems

by

Mahmoud Elfar

Department of Electrical and Computer Engineering
Duke University

Date: ______________________________

Approved:

_________________________________
Miroslav Pajic, Advisor

_________________________________
Krishnendu Chakrabarty

_________________________________
Benjamin C. Lee

_________________________________
Michael Zavlanos

_________________________________
Mary L. Cummings

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University

2022
Abstract

Cyber-Physical Systems (CPS) feature synergetic integration of multiple subsystems to control physical environments through cycles of sensing and actuation. The correct-by-design paradigm aims to provide guarantees on the performance of CPS by utilizing formally-proven algorithms for synthesis and validation of various system components. This paradigm postulates the ability to both derive adequate abstractions of the system, and mathematically formalize design requirements. Therefore, developing mathematical tools that allow system designers to easily model CPS, and to capture design requirements, is imperative to such paradigm.

This dissertation provides theoretical and experimental contributions towards modeling and development of assured and adaptive CPS. In particular, we propose Delayed Action Game (DAG) to aid with modeling CPS where part of the state is hidden from the controller. The formalism deploys the concept of delaying actions as means to hide them from other players without the usage of private variables, allowing the use of off-the-shelf model checkers for analysis. Based on a DAG model, we design an algorithm that utilizes model checkers to synthesize optimal strategies. In addition, we propose Context-Aware Probabilistic Temporal Logic (CAPTL) to aid with formalizing temporal requirements that can naturally described as a set of objectives that are prioritized based on some probabilistic conditions. Furthermore, we develop the algorithm that allow for synthesizing optimal strategies for a Markov Decision Process (MDP) that satisfy a given CAPTL-based requirement.

We deploy the theoretical frameworks in two application domains: human-robot interaction and digital microfluidics, with the goal of designing systems to be more adaptive to their environments. First, we develop protocols for supervisory systems where a human operator, supervising a number of Unmanned Aerial Vehicles (UAVs),
can intermittently perform geolocation tasks to aid in detection of possible attacks. We model the system as a DAG, and further use it to synthesize security-aware human-UAV protocols that both provide UAV path plans, increasing the chances of attack detection, and specify the time instances at which the operator is advised to perform a geolocation task. Second, we propose a stochastic game-based framework for droplet routing in Micro-Electrode-Dot Array (MEDA) biochips. The framework utilizes the ability to sense microelectrode health to synthesize routing plans that adapt to the microelectrode degradation levels in run-time. Using multiple real-life bioassays for evaluation, we show that the framework increases the probability of successful completion of benchmark bioassays. Finally, we adapt the framework to utilize Deep Reinforcement Learning (DRL) algorithms to achieve the same task.
Acknowledgements

The work presented in this dissertation was supported in part by the National Science Foundation under grants no. ECCS-1914796 and CNS-1652544; the Office of Naval Research under grants no. N00014-17-1-2012, N00014-20-1-2745, and N00014-17-1-2504; and the Air Force Office of Scientific Research under award no. FA9550-19-1-0169.

I am deeply grateful to my advisor: Prof. Miroslav Pajic, for his endless support, guidance, patience and encouragement throughout my doctoral journey, without which this dissertation would not have been possible. You have taught me so much about cyber-physical systems and formal methods. I have reached dead ends many times, especially when my health failed me, but you have been always patient, understanding and supportive.

I am also grateful to Prof. Krishnendu Chakrabarty, Prof. Benjamin C. Lee, Prof. Michael Zavlanos, and Prof. Missy Cummings, for their guidance, support, and feedback. It is such an honor to have you serving in my committee.

I would like to extend my gratitude to my collaborators: Prof. Missy Cummings, Prof. Krishnendu Chakrabarty, Yu Wang, Aditya Zutshi, Zipeng Li, Haibei Zhu, Tung-Che Liang, Zhanwei Zhong, Yi-Chen Chang, and Harrison Hao-Yu Ku; and to my labmates: Amir Khazaraei, Qitong Gao, Muhammad A. Naem, Michael Naumann, Siddhartha Nalluri, and Ilija Jovanov. It was a pleasure working with you.

Thank you, Prof. Missy Cummings, for teaching me about human-robot interactions and experimental design; for always being friendly, welcoming and sincere; for including me in your lab’s social gatherings and the conference trip to DC; and for being such a great and inspiring character to look up to.

Thank you, Prof. Krishnendu Chakrabarty, for your help with my work on mi-
crofluidics, and for your invaluable insights and feedback on the papers we collaborated on.

I am also grateful for all my professors and instructors at Duke, from whom I learned so much. There are more names than I can mention — thank you all, especially Prof. Michael Zavlanos, and Prof. Kishor S. Trivedi.

Thank you, Prof. Islam S.M. Khalil, for believing in me, and for teaching me how to conduct research for the first time.

Thank you, Prof. Yasser Shoukry, for your endless advice and support, for your kind mentorship over the years, for your friendship, and for being such an inspiring character to look up to, both on a personal and professional level.

Thanks to my friends: Lauri Bakooza, Mohamed Ibrahim, Abdelaziz Mohsen, Mahmoud Samir, Megan Hollenbeck, and Yiming Zhao, for always being there for me. Thank you Dr. William Bynum and Dr. Allison Ford for your unconditional support.

And, above all, I thank God for everything, and for blessing me with such great parents: Ahmed and Afaf, and siblings: Mohamed, Marwa and Mostafa, whose prayers, support and unconditional love have been with me all along.
Contents

Abstract iv

Acknowledgements vi

List of Figures xiv

List of Tables xix

1 Introduction 1

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.1 Human-CPS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.1.2 Digital Microfluidics . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.4 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Delayed-Action Games 10

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.2 Stochastic Games . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.2.1 Notation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.2.2 Turn-Based Stochastic Games (SMGs) . . . . . . . . . . . . . . 13

2.2.3 Hidden-Information Games . . . . . . . . . . . . . . . . . . . . . 13

2.2.4 HIG Semantics . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.2.5 Problem Formulation . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.3 Delayed-Action Games . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.3.1 Delayed Actions . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.3.2 DAG Semantics .............................................. 19
2.3.3 Simulation Relation ........................................ 22
2.4 Properties of DAG and DAG-based Synthesis ................. 24
  2.4.1 Transitions ................................................. 24
  2.4.2 Decomposition ............................................. 25
  2.4.3 Preservation of safety properties ............................ 26
  2.4.4 Synthesis Framework ...................................... 27
2.5 Case Study ..................................................... 28
  2.5.1 Modeling ................................................... 29
  2.5.2 Experimental Results ...................................... 31
2.6 Discussion and Conclusion .................................. 33

3 Context-Aware Probabilistic Temporal Logic .................. 35
  3.1 Introduction .................................................. 35
    3.1.1 Related Work ........................................... 37
  3.2 Problem Setting .............................................. 39
    3.2.1 Preliminaries ........................................... 39
    3.2.2 Motivating Example .................................... 41
  3.3 Context-Aware Temporal Logic ............................... 42
    3.3.1 CAPTL Syntax .......................................... 42
    3.3.2 CAPTL Semantics for MDPs ............................... 44
    3.3.3 CAPTL Fragments ...................................... 47
  3.4 CAPTL-Based Synthesis .................................... 48
    3.4.1 PCTL-Based Approach .................................. 49
    3.4.2 Synthesis for P-CAPTL ................................. 49
3.5 Experimental Evaluation ............................................ 55
  3.5.1 Robotic Task Planner ........................................ 55
  3.5.2 MEDA-Biochip Scheduler .................................... 56
3.6 Conclusion ................................................................. 58

4 Security-Aware Synthesis of Human-UAV Protocols ............ 59
  4.1 Introduction ............................................................ 59
  4.2 Background and Problem Statement ............................. 61
    4.2.1 Stochastic Games ............................................. 61
    4.2.2 Human-UAV Supervisory Systems ............................ 63
  4.3 System Modeling ....................................................... 64
    4.3.1 SMG-Based Model ............................................. 64
    4.3.2 DAG-Based Model ............................................ 65
  4.4 Human Geolocation Model .......................................... 67
    4.4.1 Experimental Platform and Design ....................... 68
    4.4.2 Human Geolocation Strategies .............................. 69
    4.4.3 Human Geolocation Task (GT) Predictor .................. 71
  4.5 Protocol Synthesis .................................................. 73
    4.5.1 Synthesis Objectives ....................................... 73
    4.5.2 Synthesis Procedure ....................................... 74
  4.6 Experimental Results ............................................. 74
  4.7 Discussion and Conclusion ....................................... 78

5 Error Recovery Protocols for MEDA Biochips ................... 80
  5.1 Introduction .......................................................... 80
  5.2 Preliminaries ........................................................ 84
5.2.1 Digital Microfluidics and MEDA ........................................... 84
5.2.2 Formal Modeling of Stochastic Processes ............................... 86
5.3 Problem Formulation .............................................................. 89
5.3.1 Strategy Overview .............................................................. 91
5.4 Synthesis of Local Error-Recovery Protocols ................................ 92
5.4.1 Error-Recovery Model ......................................................... 93
5.4.2 Model Composition ............................................................ 98
5.4.3 Strategy Synthesis ............................................................. 100
5.5 Global Error Recovery ............................................................ 101
5.5.1 Inputs to Local Error Recovery ............................................. 101
5.5.2 Online Synthesis ............................................................... 105
5.6 Experimental Results ............................................................. 106
5.6.1 Model Analysis ................................................................. 107
5.6.2 Benefits of the Dynamic Protocol ....................................... 108
5.6.3 Results for Bioassays ......................................................... 110
5.7 Discussion and Conclusion ...................................................... 113

6 Stochastic Game-based Modeling for MEDA Biochips .................... 115
6.1 Introduction ........................................................................ 115
6.2 Background ....................................................................... 118
6.2.1 Notation ...................................................................... 118
6.2.2 Bioprotocol Synthesis on DMFBs ..................................... 118
6.2.3 Error Recovery Techniques .............................................. 119
6.3 Microelectrode Cell Design .................................................... 119
6.3.1 MEDA Biochips and Microelectrode Cell ......................... 119
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>MEDA Modeling for DRL</td>
<td>159</td>
</tr>
<tr>
<td>7.4</td>
<td>DRL Agent Design and Training</td>
<td>163</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Training Configurations</td>
<td>163</td>
</tr>
<tr>
<td>7.4.2</td>
<td>DNN Design and Training</td>
<td>165</td>
</tr>
<tr>
<td>7.5</td>
<td>Experimental Evaluation</td>
<td>167</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Estimation of Degradation Parameters</td>
<td>167</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Evaluation</td>
<td>169</td>
</tr>
<tr>
<td>7.6</td>
<td>Conclusion</td>
<td>172</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>174</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary and Contributions</td>
<td>174</td>
</tr>
<tr>
<td>8.2</td>
<td>Limitations and Future Avenues</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td><strong>Bibliography</strong></td>
<td>181</td>
</tr>
<tr>
<td></td>
<td><strong>Biography</strong></td>
<td>197</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The UAV belief (solid square) vs. the true value (solid diamond) of its location. 15

2.2 Semantic rules for an HIG. 16

2.3 An example of the UAV motion in a 2D-grid map, modeled as an HIG. Solid squares represent the UAV belief, while solid diamonds represent the ground truth. The UAV action GT denotes performing a geolocation task. 17

2.4 An example of an HIG-based system model comprised of the UAV ($\mathcal{M}_{\text{uav}}$), the adversary ($\mathcal{M}_{\text{adv}}$), and the AS ($\mathcal{M}_{\text{as}}$). Framed information is hidden from the UAV-AS. 17

2.5 The same scenario as in Fig. 2.3 modeled as a DAG. Solid squares represent UAV belief, while solid diamonds represent the ground truth. The UAV action GT denotes performing a geolocation task. 19

2.6 Semantic rules for DAGs. 20

2.7 Semantic rules for HIG-to-DAG transformation. 22

2.8 Synthesis and analysis framework based on the use of DAGs. 27

2.9 Primary DAG components: UAV ($\mathcal{M}_{\text{uav}}$), adversary ($\mathcal{M}_{\text{adv}}$), and AS ($\mathcal{M}_{\text{as}}$). Auxiliary DAG components: memory write ($\mathcal{M}_{\text{mwr}}$) and memory read ($\mathcal{M}_{\text{mrd}}$) models, capturing the DAG representation. At stage $i$, the next memory location to write/read is $m_i$. 30

2.10 (a) The environment setup used for the case study; (b) the induced supergame MDP, where the subgames form its states; and (c) the synthesized protocols. 31

2.11 Analysis results for (a) subgame $\hat{G}_{51}$ and (b) supergame $G_D$. 32

3.1 A motivating example of a robot (left) and part of its model (right). 41

3.2 The CAPTL requirement for the running example. 44
3.3 The persistence CAPTL requirement for the running example.

3.4 Partitioning the state-space of the running example using \( q_0, q_1, q_2, \) and \( q_3 \). For example, \((q_1, g_0, h_5, 2, 1) = \triangleright\) indicates that \( s : (g, h, x, y) = (0, 5, 2, 1) \in R_{q_3}^{q_1} \)

3.5 The protocol synthesized based on the CAPTL requirement in Fig. 3.3, where \( R_V = \text{reach}(\mathcal{M}_A^a, v_0) \).

3.6 The MEDA biochip scheduler model (left) and the droplet model (right).

3.7 P-CAPTL requirement for a MEDA-biochip segment scheduler.

4.1 SMG-based model components: (a) UAV (\( \mathcal{M}_{uav} \)) and adversary (\( \mathcal{M}_{adv} \)), and (b) human geolocation (\( \mathcal{M}_{hgl} \)).

4.2 DAG-based model with the UAV (\( \mathcal{M}_{uav} \)); adversary (\( \mathcal{M}_{adv} \)); human geolocation (\( \mathcal{M}_{hgl} \)); and DAG memory read (\( \mathcal{M}_{mrd} \)), write (\( \mathcal{M}_{mwr} \)).

4.3 RESCHU-SA operator interface (OI) elements. A UAV placement on the map is based on its potentially compromised GPS reported location.

4.4 Human geolocation strategies observed during the experiments. A true positive (TP) refers to a geolocation task confirming discrepancies while a GPS spoofing attack is active.

4.5 An example of map segments and their associated GIS labels.

4.6 Survey interface where image similarity is rated on a 5-point scale.

4.7 The mission setup used for experimental evaluation and the corresponding protocols. For clarity, the map colors are altered and the protocols are partially displayed.

4.8 Analysis results for the synthesized protocol, where \( d_{rem} \) is the remaining distance to target, and \( d_{inc} \) is the increase in distance to target. Times and distances are relative to those of the shortest path.

4.9 (left) An example of the image triplets used for subjective evaluation, and (right) the evaluation results.
5.1 Illustration of the MEDA architecture and microelectrode cell (µ-electrode cell) [LLY+16b]. .................................................. 81
5.2 Sensing circuit in MEDA biochips [HWL+16]. ......................... 85
5.3 Experimental results for LOC distributions; P is the outcome probability for a ⟨operation, category⟩ pair. ................................. 86
5.4 Relationship between local and global error recovery. ............... 91
5.5 Overview of the proposed error-recovery strategy. ...................... 92
5.6 Data flow structure of the error-recovery model. ....................... 93
5.7 Error-recovery model for mixing (ℳmix). ............................... 94
5.8 Error-recovery model for splitting (ℳsplit). ............................ 95
5.9 Model of resources. .......................................................... 96
5.10 Platform abstraction model (ℳplat). .................................... 98
5.11 Synchronization model (ℳsync). ......................................... 99
5.12 Pseudo-code for determining the number of available fluidic modules. ................................. 104
5.13 Online synthesis for global error recovery. ............................ 105
5.14 Effect of resource availability for a mixing operation on its success probability under the optimal error-recovery protocol. ............. 108
5.15 The sequencing graph of an example bioassay. ....................... 109
5.16 The schedule when error (a) never occurs, (b) occurs in ℳ4, (c) occurs in ℳ7 using static protocol, (d) occurs in ℳ7 using dynamic protocol. ................................. 110
5.17 (a) CEP benchmark. (b) Master-mix benchmark. (c) Serial-dilution benchmark. .......................................................... 111
5.18 The average completion time using the proposed dynamic error-recovery protocol for (a) CEP, (b) master-mix, and (c) serial-dilution; the probability of success for (d) CEP, (e) master-mix, and (f) serial-dilution. 112

5.19 A comparison between the average completion time of the static protocol from [LLY+16a] and the proposed dynamic protocol for (a) CEP, (b) master-mix, (c) serial-dilution; and their POS for (d) CEP, (e) master-mix, and (f) serial-dilution. 112

6.1 Schematics of (a) the original and (b) the new proposed microelectrode circuit models for MEDA biochips. 120

6.2 Simulation results for the new microelectrode design. 122

6.3 Simulation results for the correlation coefficient versus the Manhattan distance between two microelectrodes. 123

6.4 (a) The fabricated DMFB and (b) the experimental setup. 125

6.5 Experimental results for electrode degradation from (a) charge trapping and (b) residual charge. 126

6.6 Measured (markers) and fitted (lines) relative EWOD force $F_{ij}$ versus the number of actuations $n$. 128

6.7 Impact of the no. of actuations $n$ on the actual degradation $D_{ij}$ and observed MC health $H_{ij}$ under different configurations. 129

6.8 MEDA biochip segment with droplet $\delta$ at time $k$. 130

6.9 Effect of microfluidic actions on droplets. Blue (solid) markers show initial locations. Red (dashed) markers show locations after successful execution. 131

6.10 MEDA biochip segment with droplet $\delta$ actuated under microfluidic action $a_{NE}$ at time $k$. 133

6.11 Model for droplet $\delta = (x_a, y_a, x_b, y_b)$ showing two microfluidic actions $a_E$ and $a_{NE}$ and their transitions, where $q(E \mid \delta, a) = 1 - p(E \mid \delta, a)$. 134

6.12 Sequence graph example (left) and the corresponding MO list (right). 138
List of Tables

2.1 Results for strategy synthesis using queries $\phi_{syn}$ and $\phi_{ana}$. .................. 33

3.1 Protocol synthesis performance results for the robotic task planner (C1) and the MEDA-biochip scheduler (C2). (St.: states, Tr.: transitions, Ch.: choices). ...................... 57

4.1 List of the GIS layers and their semantics. ...................... 70

4.2 Performance results for using queries $\phi_{syn}$ and $\phi_{ana}$. ...................... 78

5.1 List of Corrective Actions and Associated Costs ...................... 96

5.2 Probability distributions over the possible outcomes of mixing and splitting process. ...................... 97

6.1 Notation used for the simulation. ...................... 122

6.2 Microfluidic actions and their frontier sets, where $\delta = (x_a, y_a, x_b, y_b)$, $x^+ = x + 1$, $x^- = x - 1$, $d \in \{N, S\}$, and $d' \in \{E, W\}$. ...................... 132

6.3 List of microfluidic operations and the corresponding number of input and output droplets. ...................... 138

6.4 Example for converting MOs to RJs for a MEDA biochip of size $W \times H = 60 \times 30$. ...................... 141

6.5 Performance results for various droplet and biochip sizes. ...................... 153

7.1 CNN layers and their configurations. ...................... 165
Chapter 1

Introduction

1.1 Motivation

A Cyber-Physical System (CPS) is a synergetic integration of multiple subsystems that interact with and control a physical environment through sensing and actuation to achieve a number of goals that are often described as a set of requirements. The correct-by-design paradigm in CPS is a central concept during the design phase of various system components. This paradigm postulates the ability to derive adequate abstractions of system and environment behaviors, as well as the ability to mathematically formalize design requirements. The first ability is typically realized by the use and development of well-defined modeling languages, which we refer to in this dissertation as *formal models* or *formalisms*, while the second ability can be realized by using Temporal Logic (TL) to facilitate capturing design requirements in a usable manner.

Formalisms deployed in the CPS design process are mathematical frameworks for modeling system evolution in time for the purpose of describing, explaining, predicting, and controlling its behavior. Typically, a formalism provides means to describe the system states and how they transition. State transitions falls into three categories: deterministic, non-deterministic and stochastic. A transition from one state is deterministic if the next state is the only possible way for the current state to evolve. Conversely, a transition from is stochastic if the next state is sampled from a nonempty set of states based on some known probabilistic distribution. Such transi-

\[^1\]While the term *formalism* encompasses a wider definition, it is used in this dissertation to strictly refer to mathematical frameworks used in CPS.
tion is nondeterministic if the distribution over the possible next states is unknown or undetermined. Nondeterministic transitions can be used to model run-time decisions made by the system (e.g., control actions), or unknown behaviors of the environment (e.g., actions taken by an adversary).

On the other hand, TLs are mathematical frameworks that allow for describing system behavior using combinations of time-based propositions, called properties. The properties captured by a TL can be categorized based on the nature of the behavior being described. For instance, safety properties describe a system behavior where undesirable (or unsafe) states are never reached, while reachability properties describe a system behavior where desirable (or goal) states are eventually reached. Some TLs (e.g., probabilistic computation tree logic) may provide the ability to quantify those properties in a probabilistic manner. Hence, choosing the right TL largely depends on the ability to use such TL to easily and correctly capture the given design requirements — that is, its expressiveness — as well as the availability of model checking tools that support the logic. Consequently, developing TLs that are tailored towards specialized domains in CPS can facilitate the design process.

In this dissertation, the problem of modeling and design of assured and adaptive CPS is explored in two application domains, namely, Human-Cyber-Physical System (HCPS) and Micro-Electrode-Dot Array (MEDA).

1.1.1 Human-CPS

Unsurprisingly, many autonomous systems involve human interactions, ranging from active engagement to supervisory roles where human input is needed to set or update system goals, and to intervene whenever necessary [CBMM07]. One example is the human-unmanned aerial vehicle (HUAV) command and control system, where a human operator is responsible for supervising multiple Unmanned Aerial Vehicles
(UAVs) during a mission. The operator performs various supervisory tasks including, for example, updating mission goals, monitoring agent status, and adjusting flight plans [NCC07]. The operator can also be assigned primary tasks that are mission relevant, such as imagery tasks.

In this dissertation, we consider HUAV where UAVs are prone to malicious attacks on their GPS sensors that aim to introduce incremental deviations to the sensor readings to deviate the UAVs from their original flight plan [HLP+08, KSBH14, KYH15]. In literature, such attacks can be mitigated by relying on sensor redundancy in order to detect and isolate the compromised sensors [PDB13, PLP17, PIW+17, SNP+17]. However, such attacks were shown to evade detection by relying on the introduction of small, yet incremental, deviations from the original readings to remain stealthy [MS10, KHL14]. Moreover, the detection techniques proposed in literature does not consider possible roles of the human supervisor in aiding with the detection of such attacks.

### 1.1.2 Digital Microfluidics

Digital microfluidics is concerned with the precise manipulation of fluid droplets on the micro scale using electrowetting-on-dielectric (EWOD) forces. The process of manipulation occurs within a digital microfluidic biochip (DMFB), a lab-on-chip technology that has found applications in many areas, including medical diagnostics, DNA sequencing, and cell analysis. Micro-electrode-dot-array (MEDA) is an architecture for DMFBs that features a matrix of identical basic microfluidic unit, called microelectrode cells, each of which is capable of both excreting EWOD force on and detecting the presence of a droplet. In contrast to DMFBs where a droplet movement is controlled by a single electrode at a time, microelectrodes in MEDA biochips are actuated in groups, known as actuation patterns, for droplet manip-
ulation. This renders MEDA biochips capable of performing complex and precise microfluidic operations such as droplet mixing, splitting and dilution.

Similar to DMFBs, microfluidic operations in MEDA biochips are prone to faults, mainly due to manufacturing defects and microelectrode degradation. In particular, the repetitive actuation of a microelectrode can lead to charge trapping, as well as the degradation of the insulation layer, impacting the EWOD force that the microelectrode can exert on a droplet. When enough microelectrodes are degraded during a bioassay execution, the associated microfluidic operations may fail, resulting in the failure of the bioassay execution itself. Such failures both increase the time required for successful executions and decrease the useful lifespan of the biochip. Consequently, developing adaptive droplet control schemes is paramount to the successful commercial deployment of MEDA technology.

The methods proposed in literature to mitigate failures encountered during bioassay execution can be characterized as reactive. Reactive methods are concerned with how to recover from errors occurring on the microfluidic operation level. For example, an error occurring during a mixing operation may be corrected by repeating the mixing operation, while an error occurring during a splitting operation may be corrected by remixing and resplitting \([\text{LLY}+16a, \text{LLM}+17, \text{ZLC}18a]\). In other cases, the failed operations may be rescheduled again using different onboard resources to avoid the microelectrodes where the errors previously occurred \([\text{LZ}20]\). Despite their effectiveness, reactive methods for error recovery require that the error occurs in run-time before deploying the recovery strategy. This can be problematic with bioprotocols that involve either time-sensitive operations (e.g., flash chemistry \([\text{YN}08]\)), a limited number of backup droplets due to the limited supply of samples or the high cost of the reagents used (e.g., prenatal testing \([\text{RRL}+20]\)).
1.2 Research Questions

The focus of this dissertation is threefold. First, we investigate the problem of modeling cyber-physical systems with hidden information. Specifically, we are interested in answering the question of how to model cyber-physical systems that feature multiple agents, where the system state is partially hidden. Second, we address the problem of designing cyber-physical systems based on multiple objectives. In particular, we attempt to answer the question of how to formalize system requirements that are naturally specified as a set of objectives with an underlying priority structure. Finally, we address the problem of how to utilize the developed models and requirements to design assured and adaptive cyber-physical systems.

1.3 Contributions

To address the aforementioned questions, the main contributions of this thesis are as follows. First, we develop the delayed-action game (DAG) formalism that represents a new class of formal games that can be used to model systems where the global state is partially hidden from some players. The formalism utilizes the concept of delaying actions as means to simulate the hidden information, omitting the need for state private variables. The omission enables the use of readily-available model checkers that supports stochastic multiplayer games to implement and analyze DAG-based models. We further develop a DAG-based model for HUAV systems that are prone to stealthy attacks on their GPS sensors. We use this model to synthesize human-UAV protocols, where the human operator can aid in the detection of possible attacks by performing a number of geolocation tasks to verify the location of a UAV by comparing its camera feed with the reported location in real-time.

Second, we develop the context-aware probabilistic temporal logic (CAPTL). For
cyber-physical systems, the logic provides an intuitive means to formalize design requirements that consist of multiple objectives prioritized by sets of probabilistic conditional rules. We further develop and implement an algorithm for system synthesis based on CAPTL specifications.

Third, we develop a droplet routing framework for MEDA biochip to address the problem of microelectrode degradation. As highlighted earlier in Sec. 1.1.2, the methods proposed in literature for error recovery are reactive in nature and come with their drawbacks. In contrast, this dissertation proposes a preventive method that aim to minimize the likelihood of such errors in the first place. Specifically, the framework features a novel stochastic game-based model that incorporates droplet movements, microelectrode degradation, and a new microelectrode design that enables the sensing of microelectrode health levels in real time. We further utilize this framework to design droplet routing strategies under various biochip configurations and degradation profiles, such that errors in microfluidic operations are avoided in the first place. To this end, we investigate two approaches. On one hand, we use model checkers to synthesize routing strategies with probabilistic guarantees. On the other hand, we deploy deep reinforcement learning to train deep neural networks to achieve the same task. The deep neural networks are designed to ensure the scalability of the framework where we deploy transfer learning techniques to accelerate the training process. We further run experiments to derive a comparison between both approaches.

1.4 Organization

The remainder of this thesis is organized as follows. Note that specific notations, definitions and literature reviews are provided within each chapter.

- Chapter 2 [Delayed-Action Games] presents the proposed formalism (i.e., DAG)
for modeling multi-agent systems with hidden information, motivated by assured systems in which adversarial actions are potential hidden. The chapter starts by defining the syntax and semantics of DAGs, followed by a proof that DAGs can simulate hidden information games is provided. Next, an algorithm that utilizes off-the-shelf model checkers to synthesize control strategies using DAGs is presented, along with experimental evaluation.

- Chapter 3, **Context-Aware Probabilistic Temporal Logic** addresses the problem of synthesizing system components with context-based probabilistic requirements by introducing the new temporal logic, CAPTL. After providing a motivating example, the CAPTL syntax is defined, followed by defining its satisfaction semantics for MDPs. Given a CAPTL-based requirement for a system modeled as an MDP, an algorithm is provided for strategy synthesis and the necessary proofs. For evaluation, two case studies on HCPS and MEDA.

- Chapter 4, **Security-Aware Synthesis of Human-UAV Protocols** builds on Chapter 2 by demonstrating an application of the DAG synthesis framework in Human-Cyber-Physical System (HCPS). Specifically, we consider systems where multiple unmanned aerial vehicles (UAVs), supervised by a human operator, are vulnerable to stealthy false-data injection attacks. This chapter shows how to use DAG to model the system and, subsequently, how to use off-the-shelf model checkers to synthesize system protocols that provides probabilistic security guarantees.

- Chapter 5, **Error Recovery Protocols for MEDA Biochips** showcases the use of stochastic multiplayer games (SMGs) in Micro-Electrode-Dot Array (MEDA) systems to ensure adaptability against bioassay-level errors. In the first part, we show how to formalize error-recovery objectives for bioassay executions,
followed by the development of an SMG model of the system. In the second part, we demonstrate how to use model checkers to synthesize optimal error-recovery protocols based on the given objectives. Finally, the technique is evaluated using three real-life bioassays.

- Chapter 6, *Stochastic Game-based Modeling for MEDA Biochips*, examines the problem of developing MEDA systems that are resilient to biochip degradation. In particular, the chapter proposes the formal synthesis of droplet routing strategies that take into account the health status of the microelectrodes. It starts by developing an SMG model for droplet movements on a MEDA biochip, followed by proposing a method for using this model to synthesize droplet routing strategies that dynamically adapts to the current degradation levels. The chapter also includes comprehensive simulation results for real-life bioassays to evaluate the proposed method.

- Chapter 7, *Adaptive Droplet Routing for MEDA Biochips via Deep Reinforcement Learning*, presents a deep reinforcement learning approach to the problem discussed in the previous chapter. Specifically, it proposes a framework that utilizes the SMG model for droplet movement to train convolutional neural networks for the adaptive routing task. The chapter describes the DRL process used, including the reward function, network structure, and training algorithm. It also proposes transfer learning to accelerate the training process across various environment configurations. Finally, experimental results are provided, where the framework is evaluated. The chapter ends with a comparison between the proposed approach and the one described in Chapter 6.

- Chapter 8, *Conclusions*, summarizes the topics covered in this thesis, and follows with a discussion of current results, limitations, and potential avenues for
future work.
Chapter 2

Delayed-Action Games

This chapter is an adapted reproduction of “Mahmoud Elfar, Yu Wang, and Miroslav Pajic. Security-aware synthesis using delayed-action games. In International Conference on Computer Aided Verification, pages 180–199. Springer, 2019” [EWP19a]; and hence is not available under a Creative Commons license.

2.1 Introduction

Stochastic multiplayer games (SMGs) are used to model reactive systems where non-deterministic decisions are made by multiple players [BCKT18, FS18, NT16]. SMGs extend probabilistic automata by assigning a player to each choice to be made in the game. This extension enables modeling of complex systems where the behavior of players is unknown at design time. The strategy synthesis problem aims to find a winning strategy, i.e., a strategy that guarantees that a set of objectives (or winning conditions) is satisfied [CFK+13a, LSSS14]. Algorithms for synthesis include, for instance, value iteration and strategy iteration techniques, where multiple reward-based objectives are satisfied [BKTW15, CKSW13, KKKW18]. To tackle the state-space explosion problem, [Wil15] presents an assume-guarantee synthesis framework that relies on synthesizing strategies on the component level first, before composing them into a global winning strategy. Mean-payoffs and ratio rewards are further investigated in [BKW17] to synthesize $\varepsilon$-optimal strategies. Formal tools that support strategy synthesis via SMGs include PRISM [CFK+13b, KPW18] and Uppaal Stratego [DJL+15].

SMGs are classified based on the number of players that can make choices at
each state. In *concurrent* games, more than one player is allowed to concurrently make choices at a given state. Conversely, *turn-based* games assign one player at most to each state. Another classification considers the information available to different players across the game [RB94]. *Complete-information* games (also known as *perfect-information* games [CH05]) grant all players complete access to the information within the game. In *symmetric* games, some information is equally hidden from all players. On the contrary, *asymmetric* games allow some players to have access to more information than the others [RB94].

This work is motivated by security-aware systems in which stealthy adversarial actions are potentially hidden from the system, where the latter can probabilistically and intermittently gain full knowledge about the current state. While hidden-information games (HIGs) can be used to model such systems by using private variables to capture hidden information [CH05], standard model checkers can only synthesize strategies for (full-information) SMGs; thus, demanding for alternative representations. The equivalence between turn-based semi-perfect information games and concurrent perfect-information games was shown [CH05]. Since a player’s strategy mainly rely on full knowledge of the game state [CKSW13], using SMGs for synthesis produces strategies that may violate synthesis specifications in cases where required information is hidden from the player. *Partially-observable* stochastic games (POSGs) allow agents to have different belief states by incorporating uncertainty about both the current state and adversarial plans [HBZ04]. Techniques such as active sensing for online replanning [FT15] and grid-based abstractions of belief spaces [NPZ15] were proposed to mitigate synthesis complexity arising from partial observability. The notion of *delaying actions* has been studied as means for gaining information about a game to improve future strategies [KZ15, Zim16], but was not deployed as means for hiding information.
To this end, we introduce delayed-action games (DAGs) — a new class of games that simulate HIGs, where information is hidden from one player by delaying the actions of the others. The omission of private variables enables the use of off-the-shelf tools to implement and analyze DAG-based models. We show how DAGs (under some mild and practical assumptions) can be decomposed into subgames that can be independently explored, reducing the time required for synthesis by employing parallel computation. Moreover, we propose a DAG-based framework for strategy synthesis and analysis of security-aware systems. Finally, we demonstrate the framework’s applicability through a case study of security-aware planning for an unmanned-aerial vehicle (UAV) system prone to stealthy cyber-attacks, where we develop a DAG-based system model and further synthesize strategies with strong probabilistic security guarantees.

The chapter is organized as follows. Sec. 2.2 presents SMGs, HIGs, and problem formulation. In Sec. 2.3, we introduce DAGs and show that they can simulate HIGs. Sec. 2.4 proposes a DAG-based synthesis framework, which we use for security-aware planning for UAVs in Sec. 2.5, before concluding the chapter in Sec. 2.6.

### 2.2 Stochastic Games

In this section, we present turn-based stochastic games, which assume that all players have full information about the game state. We then introduce hidden-information games and their private-variable semantics.

#### 2.2.1 Notation

We use $\mathbb{N}_0$ to denote the set of non-negative integers. $\mathcal{P}(A)$ denotes the powerset of $A$ (i.e., $2^A$). A variable $v$ has a set of valuations $Ev(v)$, where $\eta(v) \in Ev(v)$ denotes one. We use $\Sigma^*$ to denote the set of all finite words over alphabet $\Sigma$, including the
empty word $\epsilon$. The mapping $Eff : \Sigma^* \times Ev(v) \rightarrow Ev(v)$ indicates the effect of a finite word on $\eta(v)$. Finally, for general indexing, we use $s_i$ or $s^{(i)}$, for $i \in \mathbb{N}_0$, while $PL_{\gamma}$ denotes $Player \gamma$.

2.2.2 Turn-Based Stochastic Games (SMGs)

SMGs can be used to model reactive systems that undergo both stochastic and non-deterministic transitions from one state to another. In a turn-based game\(^1\) actions can be taken at any state by at most one player. Formally, an SMG can be defined as follows [BBGK07, SK16, Wil15].

**Definition 1** (Turn-Based Stochastic Game). A turn-based game (SMG) with players $\Gamma = \{I, II, \bigodot\}$ is a tuple $G = \langle S, (S_I, S_{II}, S_{\bigodot}), A, s_0, \delta \rangle$, where

- $S$ is a finite set of states, partitioned into $S_I$, $S_{II}$ and $S_{\bigodot}$;
- $A = A_I \cup A_{II} \cup \{\tau\}$ is a finite set of actions where $\tau$ is an empty action;
- $s_0 \in S_{II}$ is the initial state; and
- $\delta : S \times A \times S \rightarrow [0, 1]$ is a transition function, such that $\delta(s, a, s') \in \{1, 0\}$, $\forall s \in S_I \cup S_{II}, a \in A$ and $s' \in S$, and $\delta(s, \tau, s') \in [0, 1]$, $\forall s \in S_{\bigodot}$ and $s' \in S_I \cup S_{II}$, where $\sum_{s' \in S_I \cup S_{II}} \delta(s, \tau, s') = 1$ holds.

For all $s \in S_I \cup S_{II}$ and $a \in A_I \cup A_{II}$, we write $s \xrightarrow{a} s'$ if $\delta(s, a, s') = 1$. Similarly, for all $s \in S_{\bigodot}$ we write $s \xrightarrow{\bigodot} s'$ if $s'$ is randomly sampled with probability $p = \delta(s, \tau, s')$.

2.2.3 Hidden-Information Games

SMGs assume that all players have full knowledge of the current state, and hence provide perfect-information models [CH05]. In many applications, however, this as-\(^1\)The term turn-based indicates that at any state only one player can play an action. It does not necessarily imply that players take fair turns.
sumption may not hold. A great example are security-aware models where stealthy adversarial actions can be hidden from the system; e.g., the system may not even be aware that it is under attack. On the other hand, hidden-information games (HIGs) refer to games where one player does not have complete access to (or knowledge of) the current state. The notion of hidden information can be formalized with the use of private variables (PVs) [CH05]. Specifically, a game state can be encoded using variables $v_T$ and $v_B$, representing the true information, which is only known to PL_I, and PL_{II} belief, respectively.

**Definition 2** (Hidden-Information Game). A hidden-information stochastic game (HIG) with players $\Gamma = \{I, II, \Box\}$ over a set of variables $V = \{v_T, v_B\}$ is a tuple $G_H = \langle S, (S_I, S_{II}, S_\Box), A, s_0, \beta, \delta \rangle$, where

- set of states $S \subseteq Ev(v_T) \times Ev(v_B) \times \mathcal{P}(Ev(v_T)) \times \Gamma$, partitioned in $S_I, S_{II}, S_\Box$;
- $A = A_I \cup A_{II} \cup \{\tau, \theta\}$ is a finite set of actions, where $\tau$ denotes an empty action, and $\theta$ is the action capturing PL_{II} attempt to reveal the true value $v_T$;
- $s_0 \in S_{II}$ is the initial state;
- $\beta : A_{II} \rightarrow \mathcal{P}(A_I)$ is a function that defines the set of available PL_I actions, based on PL_{II} action; and
- $\delta : S \times A \times S \rightarrow [0, 1]$ is a transition function such that $\delta(s_I, a, s_\Box) = \delta(s_\Box, a, s_I) = 0$, and $\delta(s_{II}, \theta, s_\Box), \delta(s_{II}, a, s_I), \delta(s_I, a, s_{II}) \in \{0, 1\}$ for all $s_I \in S_I$, $s_{II} \in S_{II}$, $s_\Box \in S_\Box$ and $a \in A$, where $\sum_{s' \in S_{II}} \delta(s_\Box, \tau, s') = 1$.

In the above definition, $\delta$ only allows transitions $s_I$ to $s_{II}$, $s_{II}$ to $s_I$ or $s_\Box$, with $s_{II}$ to $s_\Box$ conditioned by action $\theta$, and probabilistic transitions $s_\Box$ to $s_{II}$. A game state can be written as $s = (t, u, \Omega, \gamma)$, but to simplify notation we use $s_\gamma(t, u, \Omega)$ instead, where $t \in Ev(v_T)$ is the true value of the game, $u \in Ev(v_B)$ is PL_{II} current belief,
Ω ∈ ℙ(Ev(vₜ)) \ {∅} is PL_{II} belief space, and γ ∈ Γ is the current player’s index. When the truth is hidden from PL_{II}, the belief space Ω is the information set [RB94], capturing PL_{II} knowledge about the possible true values.

**Example 1 (Belief vs. True Value).** Our motivating example is a system that consists of a UAV and a human operator. For localization, the UAV mainly relies on a GPS sensor that can be compromised to effectively steer the UAV away from its original path. While aggressive attacks can be detected, some may remain stealthy by introducing only bounded errors at each step [LJP17, PWB+17, MST16, JPT19]. For example, Fig. 2.1 shows a UAV (PL_{II}) occupying zone A and flying north (N). An adversary (PL_{I}) can launch a stealthy attack targeting its GPS, introducing a bounded error (NE, NW) to remain stealthy. The set of stealthy actions available to the attacker depends on the preceding UAV action, which is captured by the function β, where β(N) = {NE, N, NW}. Being unaware of the attack, the UAV believes that it is entering zone C, while the true new location is D due to the attack (NE). Initially, \( η(vₜ) = η(vₜ) = zₐ \), and \( Ω = \{zₐ\} \) as the UAV is certain it is in zone \( zₐ \). In \( s₂ \), \( η(vₜ) = zₐ \), yet \( η(vₜ) = zₜ \). Although \( vₜ \) is hidden, PL_{II} is aware that \( η(vₜ) \) is in \( Ω = \{zₜ, zₐ, zₜ\} \).

### 2.2.4 HIG Semantics

\( \mathcal{G}_H \) semantics is described using the rules shown in Fig. 2.2, where H2 and H3 capture PL_{II} and PL_{I} moves, respectively. The rule H4 specifies that a PL_{II} attempt \( θ \) to
\[ H1: \ s_0 = s_{II}(t_0, u_0, \Omega_0) \quad \text{if} \quad t_0 = u_0, \ \Omega_0 = \{t_0\} \]
\[ H2: \ s_{II}(t, u, \Omega) \xrightarrow{a_i} s_1(t', u', \Omega') \quad \text{if} \quad a_i \in A_{II}, \ t' = t, \ u' = Eff(a_i, u), \ \Omega' = \{t' | t' = Eff(b_i, t) \ \forall b_i \in \beta(a_i), t \in \Omega\} \]
\[ \xrightarrow{0} s_\circ(t', u', \Omega') \quad \text{if} \quad t' = t, \ u' = u, \ \Omega' = \Omega \]
\[ H3: \ s_1(t, u, \Omega) \xrightarrow{b_i} s_{II}(t', u', \Omega') \quad \text{if} \quad b_i \in \beta(a_i), \ t' = Eff(b_i, t), \ u' = u, \ \Omega' = \Omega \]
\[ H4: \ s_\circ(t, u, \Omega) \xrightarrow{p_i} s_{II}(t', u', \Omega') \quad \text{if} \quad t' = t, \ u' = u, \ \Omega' = \Omega, \ 1 - p_i = \delta(s_\circ, \tau, s_{II}) \]

Figure 2.2: Semantic rules for an HIG.

reveal the true value can succeed with probability \( p_i \) where \( PL_{II} \) belief is updated (i.e., \( u' = t \)), and remains unchanged otherwise.

Example 2 (HIG Semantics). Continuing Example 7, let us assume that the set of actions \( A_I = A_{II} = \{N, S, E, W, NE, NW, SE, SW\} \), and that \( \theta = GT \) is a geolocation task that attempts to reveal the true value of the game. Now, consider the scenario illustrated in Fig. 2.3. At the initial state \( s_0 \), the UAV attempts to move north (N), progressing the game to the state \( s_1 \), where the adversary takes her turn by selecting an action from the set \( \beta(N) = \{NE, N, NW\} \). The players take turns until the UAV performs a geolocation task \( GT \), moving from the state \( s_4 \) to \( s_5 \). With probability \( p = \delta(s_5, \tau, s_6) \), the UAV detects its true location and updates its belief accordingly (i.e., to \( s_6 \)). Otherwise, the belief remains the same (i.e., equal to \( s_4 \)).

2.2.5 Problem Formulation

Following the system described in Example 2, we now consider the composed HIG \( G_H = M_{adv} \| M_{uav} \| M_{as} \) shown in Fig. 2.4; the HIG-based model incorporates standard models of a UAV (\( M_{uav} \)), an adversary (\( M_{adv} \)), and a geolocation-task advisory system (\( M_{as} \)) (e.g., as introduced in [EZCP19a, FWHT16]). Here, the probability of a successful detection \( p(v_T, v_B) \) is a function of both the location the UAV believes

---

\( ^2 \)A geolocation task is an attempt to localize the UAV by examining its camera feed.
to be its current location ($v_B$) as well as the ground truth location that the UAV actually occupies ($v_T$). Reasoning about the flight plan using such model becomes problematic since the ground truth $v_T$ is inherently unknown to the UAV (i.e., $PL_{II}$), and thus, so is $p(v_T, v_B)$. Furthermore, such representation, where some information is hidden, is not supported by off-the-shelf SMG model checkers. Consequently, for such HIGs, our goal is to find an alternative representation that is suitable for strategy synthesis using off-the-shelf SMG model-checkers.

### 2.3 Delayed-Action Games

In this section, we provide an alternative representation of HIGs that eliminates the use of private variables — we introduce Delayed-Action Games (DAGs) that exploit the notion of delayed actions. Furthermore, we show that for any HIG, a DAG that
simulates the former can be constructed.

### 2.3.1 Delayed Actions

Informally, a DAG reconstructs an HIG such that actions of PL\textsubscript{I} (the player with access to perfect information) follow the actions of PL\textsubscript{II}, i.e., PL\textsubscript{I} actions are delayed. This rearrangement of the players’ actions provides a means to hide information from PL\textsubscript{II} without the use of private variables, since in this case, at PL\textsubscript{II} states, PL\textsubscript{I} actions have not occurred yet. In this way, PL\textsubscript{II} can act as though she has complete information at the moment she makes her decision, as the future state has not yet happened and so cannot be known. In essence, the formalism can be seen as a partial ordering of the players’ actions, exploiting the (partial) superposition property that a wide class of physical systems exhibit. To demonstrate this notion, let us consider DAG modeling on our running example.

**Example 3** (Delaying Actions). Fig. 2.5 depicts the (HIG-based) scenario from Fig. 2.3, but in the corresponding DAG, where the UAV actions are performed first (in $\hat{s}_0, \hat{s}_1, \hat{s}_2$), followed by the adversary delayed actions (in $\hat{s}_3, \hat{s}_4$). Note that, in the DAG model, at the time the UAV executed its actions ($\hat{s}_0, \hat{s}_1, \hat{s}_2$) the adversary actions had not occurred (yet). Moreover, $\hat{s}_0$ and $\hat{s}_6$ (Fig. 2.5) share the same belief and true values as $s_0$ and $s_6$ (Fig. 2.3), respectively, though the transient states do not exactly match. This will be used to show the relationship between the games.

The advantage of this approach is twofold. First, the elimination of private variables enables simulation of an HIG using a full-information game. Thus, the formulation of the strategy synthesis problem using off-the-shelf SMG-based tools becomes feasible. In particular, a PL\textsubscript{II} synthesized strategy becomes dependent on the knowledge of PL\textsubscript{I} behavior (possible actions), rather than the specific (hidden) actions. We formalize a DAG as follows.
Figure 2.5: The same scenario as in Fig. 2.3 modeled as a DAG. Solid squares represent UAV belief, while solid diamonds represent the ground truth. The UAV action GT denotes performing a geolocation task.

Definition 3 (Delayed-Action Game). A delayed-action game (DAG) of an HIG $G_H = (S, (S_1, S_{II}, S_{O}), A, s_0, \beta, \delta)$ with players $\Gamma = \{I, II, O\}$ over a set of variables $V = \{v_T, v_B\}$ is a tuple $G_D = (\hat{S}, (\hat{S}_I, \hat{S}_{II}, \hat{S}_{O}), A, \hat{s}_0, \hat{\beta}, \hat{\delta})$ where

- $\hat{S} \subseteq \text{Ev}(v_T) \times \text{Ev}(v_B) \times A_{II} \times N_0 \times \Gamma$ is the set of states, partitioned into $\hat{S}_I$, $\hat{S}_{II}$ and $\hat{S}_{O}$;
- $\hat{s}_0 \in \hat{S}_{II}$ is the initial state; and
- $\hat{\delta}: \hat{S} \times A \times \hat{S} \rightarrow [0, 1]$ is a transition function such that $\hat{\delta}(\hat{s}_{II}, a, \hat{s}_{O}) = \hat{\delta}(\hat{s}_I, a, \hat{s}_{II}) = \hat{\delta}(\hat{s}_O, a, \hat{s}_I) = 0$, and $\hat{\delta}(\hat{s}_{II}, a, \hat{s}_I) \in \{0, 1\}$, $\hat{\delta}(\hat{s}_{II}, \theta, \hat{s}_I) \in \{0, 1\}$, $\hat{\delta}(\hat{s}_I, a, \hat{s}_I) \in \{0, 1\}$, $\hat{\delta}(\hat{s}_I, a, \hat{s}_O) \in \{0, 1\}$, for all $\hat{s}_I \in \hat{S}_I$, $\hat{s}_{II} \in \hat{S}_{II}$, $\hat{s}_O \in \hat{S}_{O}$ and $a \in A$, where $\sum_{s' \in \hat{S}_{II}} \hat{\delta}(\hat{s}_O, a, s') = 1$.

Note that, in contrast to transition function $\delta$ in HIG $G_H$, $\hat{\delta}$ in DAG $G_D$ only allows transitions $\hat{s}_{II}$ to $\hat{s}_{II}$ or $\hat{s}_I$, as well as $\hat{s}_I$ to $\hat{s}_I$ or $\hat{s}_O$, and probabilistic transitions $\hat{s}_O$ to $\hat{s}_{II}$; also note that $\hat{s}_{II}$ to $\hat{s}_I$ is conditioned by the action $\theta$.

2.3.2 DAG Semantics

A DAG state is a tuple $\hat{s} = (\hat{t}, \hat{u}, w, j, \gamma)$, which for simplicity we shorthand as $\hat{s}_\gamma(\hat{t}, \hat{u}, w, j)$, where $\hat{t} \in \text{Ev}(v_T)$ is the last known true value, $\hat{u} \in \text{Ev}(v_B)$ is PL_{II}
D1: \( \hat{s}_0 = s_{II}(\hat{t}_0, \hat{u}_0, w_0, 0) \) if \( \hat{t}_0 = \hat{u}_0, w_0 = \epsilon \)

D2: \( \hat{s}_{II} (\hat{t}, \hat{u}, w, 0) \xrightarrow{a} \hat{s}_{II} (\hat{t}', \hat{u}', w', 0) \) if \( a_i \in A_{II}, \hat{t}' = \hat{t}, \hat{u}' = Eff(a_i, \hat{u}), w' = wa_i \)

D3: \( \hat{s}_I (\hat{t}, \hat{u}, w, j) \xrightarrow{b} \hat{s}_I (\hat{t}', \hat{u}', w', j+1) \) if \( b_i \in \beta(w_j), \hat{t}' = Eff(b_i, \hat{t}), \hat{u}' = \hat{u}, w' = w, j < |w| - 1 \)

D4: \( \hat{s}_0 (\hat{t}, \hat{u}, w, j) \xrightarrow{p} \hat{s}_{II} (\hat{t}', \hat{u}', w', 0) \) if \( \hat{t}' = \hat{t}, \hat{u}' = \hat{u}, w' = w, r_i = \hat{b}(\hat{s}_0, \hat{s}_{II}) \)

\[ \frac{1-p}{1-p} \hat{s}_{II} (\hat{t}', \hat{u}', w', 0) \] if \( \hat{t}' = \hat{t}_0, \hat{u}' = \hat{u}, w' = w, q_i = \hat{b}(\hat{s}_0, \hat{s}_{II}) \)

**Figure 2.6:** Semantic rules for DAGs.

belief, \( w \in A_{II}^* \) captures PL\(_{II}\) actions taken since the last known true value, \( j \in \mathbb{N}_0 \) is an index on \( w \), and \( \gamma \in \Gamma \) is the current player index. The game transitions are defined using the semantic rules from Fig. 2.6. Note that PL\(_{II}\) can execute multiple moves (i.e., actions) before executing \( \theta \) to attempt to reveal the true value (D2), moving to a PL\(_{I}\) state where PL\(_{I}\) executes all her delayed actions before reaching a ‘revealing’ state \( \hat{s}_\circ \) (D3). Finally, the revealing attempt can succeed with probability \( p_i \) when PL\(_{II}\) belief is updated (i.e., \( \hat{u}' = \hat{t} \)), or otherwise remains unchanged (D4).

In both \( G_H \) and \( G_D \), we label states where all players have full knowledge of the current state as **proper**. We also say that two states are similar if they agree on the belief, and equivalent if they agree on both the belief and ground truth.

**Definition 4 (States).** Let \( s_{\gamma} (t, u, \Omega) \in S \) and \( \hat{s}_{\gamma} (\hat{t}, \hat{u}, w, j) \in \hat{S} \). We say:

- \( s_{\gamma} \) is proper iff \( \Omega = \{t\} \), denoted by \( s_{\gamma} \in \text{Prop}(G_H) \).

- \( \hat{s}_{\gamma} \) is proper iff \( w = \epsilon \), denoted by \( \hat{s}_{\gamma} \in \text{Prop}(G_D) \).

- \( s_{\gamma} \) and \( \hat{s}_{\gamma} \) are similar iff \( \hat{u} = u, \hat{t} \in \Omega, \) and \( \gamma = \hat{\gamma} \), denoted by \( s_{\gamma} \sim \hat{s}_{\gamma} \).

- \( s_{\gamma}, \hat{s}_{\gamma} \) are equivalent iff \( t = \hat{t}, u = \hat{u}, w = \epsilon, \) and \( \gamma = \hat{\gamma} \), denoted by \( s_{\gamma} \simeq \hat{s}_{\gamma} \).

From the above definition, we have that \( s \simeq \hat{s} \implies s \in \text{Prop}(G_H), \hat{s} \in \text{Prop}(G_D) \).

We now define **execution fragments**, possible progressions from a state to another.
Definition 5 (Execution Fragment). An execution fragment (of either an SMG, DAG or HIG) is a finite sequence of states, actions and probabilities \( \varrho = s_0 a_1 p_1 s_1 a_2 p_2 s_2 \ldots a_n p_n s_n \) such that \((s_i \xrightarrow{a_{i+1}} s_{i+1}) \lor (s_i \xrightarrow{(p_{i+1})} s_{i+1}), \forall i \geq 0\).

We use \( \text{first}(\varrho) \) and \( \text{last}(\varrho) \) to refer to the first and last states of \( \varrho \), respectively. If both states are proper, we say that \( \varrho \) is proper as well, denoted by \( \varrho \in \text{Prop}(G) \).

Moreover, \( \varrho \) is deterministic if no probabilities appear in the sequence.

Definition 6 (Move). A move \( m_\gamma \) of an execution \( \varrho \) from state \( s \in \varrho \), denoted by \( \text{move}_\gamma(s, \varrho) \), is a sequence of actions \( a_1 a_2 \ldots a_i \in A_\gamma^* \) that player \( \gamma \) performs in \( \varrho \) starting from \( s \).

By omitting the player index, we refer to the moves of all players. To simplify notation, we use \( \text{move}(\varrho) \) as a short notation for \( \text{move}(\text{first}(\varrho), \varrho) \). We write \((m)(\text{first}(\varrho)) = \text{last}(\varrho)\) to denote that the execution of move \( m \) from the \( \text{first}(\varrho) \) leads to the \( \text{last}(\varrho) \). This allows us to now define the delay operator as follows.

Definition 7 (Delay Operator). For an \( G_H \), let \( m = \text{move}(\varrho) = a_1 b_1 \ldots a_n b_n \theta \) be a move for some deterministic \( \varrho \in \text{TS}(G_H) \), where \( a_1 \ldots a_n \in A_\Pi^*, b_1 \ldots b_n \in A_1^* \). The delay operator, denoted by \( m_\tau \), is defined by the rule \( m_\tau = a_1 \ldots a_n \theta b_1 \ldots b_n \).

Intuitively, the delay operator shifts PL\(_I\) actions to the right of PL\(_\Pi\) actions up until the next probabilistic state. For example,

\[
\text{if } \varrho = s_1^{(0)} \xrightarrow{a_1} s_1^{(1)} \xrightarrow{b_2} s_1^{(2)} \xrightarrow{\theta} s_1^{(3)} \xrightarrow{p_3} s_1^{(4)} \xrightarrow{a_4} s_1^{(5)} \xrightarrow{b_5} s_1^{(6)} \xrightarrow{a_6} s_1^{(7)} \xrightarrow{b_7} s_1^{(8)} \text{ then } m = a_1 \xrightarrow{b_2} \theta \xrightarrow{\tau} a_4 \xrightarrow{b_5} a_6 \xrightarrow{b_7} \\
\text{and } \overline{m} = a_1 \xrightarrow{\theta} b_2 \xrightarrow{\tau} a_4 \xrightarrow{b_5} a_6 \xrightarrow{b_7} \text{.}
\]

\(^3\)For deterministic transitions, \( p = 1, \) hence omitted from \( \varrho \) for readability.

\(^4\)An execution fragment lives in the transition system (TS), i.e., \( \varrho \in \text{Prop(TS}(G)) \). We omit TS for readability.
\[ s_0 = s_{\Pi}(t_0, u_0, \Omega_0) \quad \Rightarrow \quad \hat{s}_0 = \hat{s}_{\Pi}(\hat{t}_0, \hat{u}_0, w_0, 0) \text{ s.t. } \hat{t}_0 = t_0, \hat{u}_0 = u_0 \]
\[ s_{\Pi}(t, u, \Omega) \xrightarrow{a_i} s_1(t', u', \Omega') \quad \Rightarrow \quad \hat{s}_{\Pi}(\hat{t}, \hat{u}, w, 0) \xrightarrow{a_i} \hat{s}_1(\hat{t}', \hat{u}', w', 0) \text{ s.t. } \hat{u} = u \]
\[ s_{\Pi}(t, u, \Omega) \xrightarrow{b_j} s_0(t', u', \Omega') \quad \Rightarrow \quad \hat{s}_{\Pi}(\hat{t}, \hat{u}, w, 0) \xrightarrow{b_j} \hat{s}_0(\hat{t}', \hat{u}', w', 0) \text{ s.t. } \hat{u} = u \]
\[ s_1(t, u, \Omega) \xrightarrow{b_j} s_{\Pi}(t', u', \Omega') \quad \Rightarrow \quad \hat{s}_1(\hat{t}, \hat{u}, w, j) \xrightarrow{b_i} s_1(\hat{t}', \hat{u}', w', j+1) \text{ s.t. } \hat{t} = t, j < |w| \]
\[ s_0(t, u, \Omega) \xrightarrow{1-p_j} s_0(t', u', \Omega') \quad \Rightarrow \quad \hat{s}_0(\hat{t}, \hat{u}, w, j) \xrightarrow{1-p_i} \hat{s}_0(\hat{t}', \hat{u}', w', 0) \text{ s.t. } \hat{t} = t, \hat{u} = u \]

**Figure 2.7:** Semantic rules for HIG-to-DAG transformation.

### 2.3.3 Simulation Relation

Given an HIG \( G_H \), we first define the corresponding DAG \( G_D \).

**Definition 8** (Correspondence). Given an HIG \( G_H \), a corresponding DAG \( G_D = \mathcal{D}[G_H] \) is a DAG that follows the semantic rules displayed in Fig. 2.7.

For the rest of this section, we consider \( G_D = \mathcal{D}[G_H] \), and use \( \rho \in \text{TS}(G_H) \) and \( \hat{\rho} \in \text{TS}(G_D) \) to denote two execution fragments of the HIG and DAG, respectively. We say that \( \rho \) and \( \hat{\rho} \) are similar, denoted by \( \rho \sim \hat{\rho} \), iff \( \text{first}(\rho) \simeq \text{first}(\hat{\rho}) \), \( \text{last}(\rho) \sim \text{last}(\hat{\rho}) \), and \( \text{move}(\rho) = \text{move}(\hat{\rho}) \).

**Definition 9** (Game Proper Simulation). A game \( G_D \) properly simulates \( G_H \), denoted by \( G_D \sim G_H \), iff \( \forall \rho \in \text{Prop}(G_H), \exists \hat{\rho} \in \text{Prop}(G_D) \) such that \( \rho \sim \hat{\rho} \).

Before proving the existence of the simulation relation, we first show that if a move is executed on two equivalent states, then the terminal states are similar.

**Lemma 1** (Terminal States Similarity). For any \( s_0 \simeq \hat{s}_0 \) and a deterministic \( \rho \in \text{TS}(G_H) \), where \( \text{first}(\rho) = s_0 \), \( \text{last}(\rho) \in S_{\Pi} \), then \( \text{last}(\rho) \sim \text{last}(\hat{\rho}) \).

**Proof.** Let \( \text{last}(\rho_i) = s_{\gamma i}^{(i)}(t_i, u_i, \Omega_i) \) and \( \text{move}(\rho_i) ) \ (s_0) = \hat{s}_{\gamma i}^{(i)}(\hat{t}_i, \hat{u}_i, w_i, j_i) \), where \( \text{move}(\rho_i) = a_1b_1...a_ib_i\theta \). We then write \( \text{move}(\rho) = a_1...a_i\theta b_1...b_i \). We use induction over \( i \) as follows:
• Base \((i=0)\): \(\varrho_0 = s_0 \implies s^{(0)} \simeq \hat{s}^{(0)}\) where \(u_0 = \hat{u}_0\) and \(t_0 = \hat{t}_0\).

• Induction \((i > 0)\): Assume that the claim holds for \(\text{move}(\varrho_{i-1}) = a_1 b_1 \ldots a_{i-1} b_{i-1} \theta\), i.e., \(u_{i-1} = \hat{u}_{i-1}\) and \(\hat{t}_{i-1} \in \Omega_{i-1}\). For \(\varrho_i\) we have that \(u_i = \text{Eff}(a_i, u_{i-1})\) and \(\hat{u}_i = \text{Eff}(a_i, \hat{u}_{i-1})\). Also, \(t_i = \text{Eff}(b_i, t_{i-1}) \in \Omega_i\) and \(\hat{t}_i = \text{Eff}(b_i, \hat{t}_{i-1})\). Hence, \(u_i = \hat{u}_i\), \(\hat{t}_i \in \Omega_i\) and \(\gamma_i = \gamma_i = \bigcirc\). Thus, \(s^{(i)} \sim \hat{s}^{(i)}\) holds. The same can be shown for \(\text{move}(\varrho) = a_1 b_1 \ldots a_i b_i\) where no \(\theta\) occurs.

\[\square\]

**Theorem 1** (Probabilistic Simulation). For any \(s_0 \simeq \hat{s}_0\) and \(\varrho \in \text{Prop}(\mathcal{G}_H)\) where \(\text{first}(\varrho) = s_0\), it holds that

\[\Pr[\text{last}(\varrho) = s'] = \Pr\left[\left(\text{move}(\varrho)\right)(\hat{s}_0) = \hat{s}'\right] \quad \forall s', \hat{s}' \text{ s.t. } s' \simeq \hat{s}'\]  

*Proof.* We can rewrite \(\varrho\) as \(\varrho = \varrho_0 \overset{p_1}{\rightarrow} \varrho_1 \cdots \varrho_{n-1} \overset{p_n}{\rightarrow} s_H^{(n)}\), where \(\varrho_0, \varrho_1, \ldots, \varrho_{n-1}\) are deterministic. Let \(\text{first}(\varrho_i) = s_H^{(i)}(t_i, u_i, \Omega_i)\), \(\text{last}(\varrho_i) = s_H^{(i)}(t_i', u_i', \Omega_i')\), and

\[\left(\text{move}(\varrho)\right)(\hat{s}_0) = \hat{s}^{(n)}(\hat{t}_n, \hat{u}_n, \hat{w}_n, \hat{f}_n)\]. We use induction over \(n\) as follows:

• Base \((n=0)\): for \(\varrho\) to be deterministic and proper, \(\varrho = \varrho_0 = s^{(0)}\) holds.

• Case \((n = 1)\): \(p_1 = p(t_0', u_0')\). From Lemma \[\square\] \(\hat{u}_1 = u_1\) and \(\hat{t}_1 = t_1\). Hence,

\[\Pr[\text{last}(\varrho) = s_H^{(1)}] = \Pr[\left(\text{move}(\varrho)\right)(\hat{s}_0) = \hat{s}_H^{(1)}] = p(t_0', u_0')\) and \(s_H^{(1)} \simeq \hat{s}_H^{(1)}\).

• Induction \((n > 1)\): It is straightforward to infer that \(p_n = p(t_{n-1}', u_{n-1}')\), hence

\[\Pr[\text{last}(\varrho) = s_H^{(n)}] = \Pr[\left(\text{move}(\varrho)\right)(\hat{s}^{(0)}) = \hat{s}^{(n)}] = P\), and \(s_H^{(n)} \simeq \hat{s}_H^{(n)}\).

\[\square\]
Note that in case of multiple $\theta$ attempts, the above probability $P$ satisfies

$$P = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_i \left( t'_{i-1}, u'_{i-1} \right) \left( 1 - p_{i-1} \left( t'_{i-1}, u'_{i-1} \right) \right)^{(j-1)},$$

where $m_i$ is the number of $\theta$ attempts at stage $i$. Finally, since Theorem 1 imposes no constraints on move($\varrho$), a DAG can simulate all proper executions that exist in the corresponding HIG.

**Theorem 2** (DAG-HIG Simulation). For any HIG $G_H$ there exists a DAG $G_D = D[G_H]$ such that $G_D \sim G_H$ (as defined in Def. 3).

### 2.4 Properties of DAG and DAG-based Synthesis

We here discuss DAG features, including how it can be decomposed into subgames by restricting the simulation to finite executions, and the preservation of safety properties, before proposing a DAG-based synthesis framework.

#### 2.4.1 Transitions

In DAGs, nondeterministic actions of different players underline different semantics. Specifically, $PL_I$ nondeterminism captures what is known about the adversarial behavior, rather than exact actions, where $PL_I$ actions are constrained by the earlier $PL_{II}$ action. Conversely, $PL_{II}$ nondeterminism abstracts the player’s decisions. This distinction reflects how DAGs can be used for strategy synthesis under hidden information. To illustrate this, suppose that a strategy $\pi_{II}$ is to be obtained based on a worst-case scenario. In that case, the game is explored for all possible adversarial behaviors. Yet, if a strategy $\pi_I$ is known about $PL_I$, a counter strategy $\pi_{II}$ can be found by constructing $G_D^{\pi_I}$. 

24
Probabilistic behaviors in DAGs are captured by PL, which is characterized by the transition function \( \hat{\delta} : \hat{S}_O \times \hat{S}_{II} \rightarrow [0, 1] \). The specific definition of \( \hat{\delta} \) depends on the modeled system. For instance, if the transition function (i.e., the probability) is state-independent, i.e., \( \hat{\delta}(\hat{s}_O, \hat{s}_{II}) = c, c \in [0, 1] \), the obtained model becomes trivial. Yet, with a state-dependent transition function, i.e., \( \hat{\delta}(\hat{s}_O, \hat{s}_{II}) = p(\hat{t}, \hat{u}) \), the probability that PLII successfully reveals the true value depends on both the belief and the true value, and the transition function can then be realized since \( \hat{s}_O \) holds both \( \hat{t} \) and \( \hat{u} \).

### 2.4.2 Decomposition

Consider an execution \( \hat{\varrho}^* = \hat{s}_0a_1\hat{s}_1a_2\hat{s}_2 \ldots \) that describes a scenario where PLII performs infinitely many actions with no attempt to reveal the true value. To simulate \( \hat{\varrho}^* \), the word \( w \) needs to infinitely grow. Since we are interested in finite executions, we impose stopping criteria on the DAG, such that the game is trapped whenever \( |w| = h_{\text{max}} \) is true, where \( h_{\text{max}} \in \mathbb{N} \) is an upper horizon. We formalize the stopping criteria as a deterministic finite automaton (DFA) that, when composed with the DAG, traps the game whenever the stopping criteria hold. Note that imposing an upper horizon by itself is not a sufficient criterion for a DAG to be considered a stopping game [CFK+13]. Conversely, consider a proper (and hence finite) execution \( \hat{\varrho} = \hat{s}_0a_1 \ldots \hat{s}' \), where \( \hat{s}_0, \hat{s}' \in \text{Prop}(\mathcal{G}_D) \). From Definition 9, it follows that a DAG initial state is strictly proper, i.e., \( \hat{s}_0 \in \text{Prop}(\mathcal{G}_D) \). Hence, when \( \hat{s}' \) is reached, the game can be seen as if it is repeated with a new initial state \( \hat{s}' \). Consequently, a DAG game (complemented with stopping criteria) can be decomposed into a (possibly infinite) countable set of subgames that have the same structure yet different initial states.

**Definition 10 (DAG Subgames).** The subgames of a \( \mathcal{G}_D \) are defined by the set 
\[
\{ \hat{\mathcal{G}}_i : \hat{\mathcal{G}}_i = \langle \hat{S}(i), \hat{S}_O(i), \hat{S}_{II}(i), A, \hat{s}_0(i), \hat{s}(i) \rangle, i \in \mathbb{N}_0 \} ,
\]  
where \( \hat{S} = \bigcup_i \hat{S}(i) \); \( \hat{S}_{\gamma} = \bigcup_i \hat{S}_{\gamma}(i) \) \( \forall \gamma \in \Gamma \); and \( \hat{s}_0(i) = \hat{s}_{II}(i) \) s.t. \( \hat{s}_{II}(i) \in \text{Prop}(\mathcal{G}_{D}(i)) \), \( \hat{s}(i) \neq \hat{s}(j) \) \( \forall i, j \in \mathbb{N}_0 \).
Intuitively, each subgame either reaches a proper state (representing the initial state of another subgame) or terminates by an upper horizon. This decomposition allows for the independent (and parallel) analysis of individual subgames, drastically reducing both the time required for synthesis and the explored state space, and hence improving scalability. An example of this decompositional approach is provided in Sec. 2.5.

2.4.3 Preservation of safety properties

In DAGs, the action $\theta$ denotes a transition from $\text{PL}_{II}$ to $\text{PL}_{I}$ states and thus the execution of any delayed actions. While this action can simply describe a revealing attempt, it can also serve as a \textit{what-if} analysis of how the true value may evolve at stage $i$ of a subgame. We refer to an execution of the second type as a \textit{hypothetical branch}, where $\text{Hyp}(\hat{\varrho}, h)$ denotes the set of hypothetical branches from $\hat{\varrho}$ at stage $h \in \{1, \ldots, n\}$. Let $L_{\text{safe}}(s)$ be a labeling function denoting if a state is safe. The formula $\Phi_{\text{safe}} := [G \text{safe}]$ is satisfied by an execution $\varrho$ in HIG if and only if all $s(t, u, \Omega) \in \varrho$ are safe.

Now, consider $\hat{\varrho}$ of the DAG, with $\hat{\varrho} \sim \varrho$. We identify the following three cases:

1. $L_{\text{safe}}(s)$ depends only on the belief $u$, then $\varrho \models \Phi_{\text{safe}}$ iff all $\hat{s}_{II} \in \hat{\varrho}$ are safe;

2. $L_{\text{safe}}(s)$ depends only on the true value $t$, then $\varrho \models \Phi_{\text{safe}}$ iff all $\hat{s}_{I} \in \text{Hyp}(\hat{\varrho}, n)$ are safe; and

3. $L_{\text{safe}}(s)$ depends on both the true value $t$ and belief $u$, then $\varrho \models \Phi_{\text{safe}}$ iff $\text{last}(\hat{\varrho}_h)$ is safe for all $\hat{\varrho}_h \in \text{Hyp}(\hat{\varrho}, h), h \in \{1, \ldots, n\}$, where $n$ is the number of $\text{PL}_{II}$ actions.

Taking into account such relations, both safety (e.g., never encounter a hazard) and distance-based requirements (e.g., never exceed a subgame horizon) can be specified when using DAGs for synthesis, to ensure their satisfaction in the original model.
This can be generalized to other reward-based synthesis objectives, which will be part of our future efforts that we discuss in Sec. 2.6.

2.4.4 Synthesis Framework

We here propose a framework for strategy synthesis using DAGs, which is summarized in Fig. 2.8. We start by formulating the automata $\mathcal{M}_I$, $\mathcal{M}_{II}$ and $\mathcal{M}_\circ$, representing PL$_I$, PL$_{II}$ and PL$_\circ$ abstract behaviors, respectively. Next, a FIFO memory stack $(m_i)_{i=1}^n \in A_{II}^n$ is implemented using two automata $\mathcal{M}_{mrd}$ and $\mathcal{M}_{mwr}$ to perform reading and writing operations, respectively. The DAG $\mathcal{G}_D$ is constructed by following Algorithm 1. The game starts with PL$_{II}$ moves until she executes a revealing attempt $\theta$, allowing PL$_I$ to play her delayed actions. Once an end criterion is met, the game terminates, resembling conditions such as ‘running out of fuel’ or ‘reaching map boundaries’.

Algorithm 2 describes the procedure for strategy synthesis based on the DAG $\mathcal{G}_D$, and an rPATL [CFK+13a] synthesis query $\phi_{syn}$ that captures, for example, a safety requirement. Starting with the initial location, the procedure checks whether $\phi_{syn}$ is satisfied if action $\theta$ is performed at stage $h$, and updates the set of feasible strategies $\Pi_i$ for subgame $\hat{G}_i$ until $h_{\text{max}}$ is reached or $\phi_{syn}$ is not satisfied. Next, the set $\Pi_i$

\[ ^5 \text{Specific implementation details are described in Sec. 2.5.} \]

\[ ^6 \text{Failing to find a strategy at stage } i \text{ implies the same for all horizons of size } j > i. \]
Algorithm 1: Procedure for DAG construction

**Input:** Components $\mathcal{M}_1, \mathcal{M}_{\Pi}, \mathcal{M}_{\mathbb{O}}, \mathcal{M}_{\mathbb{wr}}, \mathcal{M}_{\mathbb{mrd}}$; initial state $\hat{s}_0$

**Result:** DAG $\mathcal{D}$

1. while $\neg$(end criterion) do
   2. while $a \neq \theta$ do ▷ PL$_{\Pi}$ plays until a revealing attempt
      3. $\mathcal{M}_{\Pi}.v_B \leftarrow \text{Eff}(a, v_B)$, $\mathcal{M}_{\mathbb{wr}}.\text{write}(a, ++wr)$ ▷ PL$_{\mathbb{wr}}$ plays all delayed actions
      4. while $rd \leq wr$ do ▷ PL$_{\mathbb{mrd}}$ plays all delayed actions
         5. $\mathcal{M}_{\mathbb{mrd}}.\text{read}(a, ++rd)$, $\mathcal{M}_{\mathbb{I}}.v_T \leftarrow \text{Eff}(\beta(a), v_T)$ ▷ PL$_{\mathbb{I}}$ plays successful attempt
      6. if $\text{draw } x \sim \text{Brn}(p(v_T, v_B))$ then ▷ PL$_{\mathbb{O}}$ plays successful attempt
         7. $\mathcal{M}_{\Pi}.v_B \leftarrow \mathcal{M}_{\Pi}.v_T$, $wr \leftarrow 0$, $rd \leftarrow 0$ ▷ Unsuccessful attempt, forget PL$_{\mathbb{I}}$ actions
      8. else $rd \leftarrow 0$

Algorithm 2: Procedure for strategy synthesis

**Input:** Initial location $(x_0, y_0)$, synthesis query $\phi_{\text{syn}}$

**Output:** PL$_{\Pi}$ strategies $\Pi_{\Pi}^*$

1. $\ell \leftarrow [(x_0, y_0)]$, $i \leftarrow 0$

2. while $i < |\ell|$ do ▷ Explore all reachable subgames
   3. $\hat{s}_0 \leftarrow (\ell[i], \ell[i], \epsilon, 0, \Pi)$, $h \leftarrow 1$, stop $\leftarrow \bot$ ▷ Construct initial state
   4. while $h \leq h_{\text{max}} \land \neg$stop do ▷ Explore subgame till upper horizon
      5. $(\pi_\Pi, \varphi) \leftarrow \text{Synth}(\mathcal{G}_{\hat{s}_0}^{\pi_\Pi}, \phi_{\text{syn}})$ ▷ Synthesize strategy for horizon $h$
      6. if $\pi_\Pi \neq \emptyset$ then ▷ Save synthesized strategy
         7. $\Pi_i \leftarrow \Pi_i \cup (\pi_\Pi, \pi_h, \varphi)$, $h++$
      8. else $stop \leftarrow \top$
   9. Prune $(\Pi_i)$, $\Pi_{\Pi}^* \leftarrow \Pi_{\Pi}^* \cup \Pi_i$ ▷ Prune subgame strategies
  10. $\ell \leftarrow \ell \cdot (\text{Reachable}(\Pi_i) \setminus \ell)$, $i++$ ▷ update reachability

is used to update the list of reachable end locations $\ell$ with new initial locations of reachable subgames that should be explored. Finally, the composition of both $\mathcal{G}_H$ and $\Pi_{\Pi}^*$ resolves PL$_{\Pi}$ nondeterminism, where the resulting model $\mathcal{G}_H^{\Pi_{\Pi}^*}$ is a Markov Decision Process (MDP) of complete information that can be easily used for further analysis.

### 2.5 Case Study

In this section, we consider a case study where a human operator supervises a UAV prone to stealthy attacks on its GPS sensor. The UAV mission is to visit a number of targets after being airborne from a known base (initial state), while avoiding hazard
zones that are known a priori. Moreover, the presence of adversarial stealthy attacks via GPS spoofing is assumed. We use the DAG framework to synthesize strategies for both the UAV and an operator advisory system (AS) that schedules geolocation tasks for the operator.

### 2.5.1 Modeling

We model the system as a delayed-action game $G$, where PL$_I$ and PL$_II$ represent the adversary and the UAV-AS coalition, respectively. Fig. 2.9 shows the model primary and auxiliary components. In the UAV model $\mathcal{M}_{\text{uav}}$, $x_B = (x_B, y_B)$ encodes the UAV belief, and $A_{\text{uav}} = \{N, S, E, W, NE, NW, SE, SW\}$ is the set of available movements. The AS can trigger the action *activate* to initiate a geolocation task, attempting to confirm the current location. The adversary behavior is abstracted by $\mathcal{M}_{\text{adv}}$ where $x_T = (x_T, y_T)$ encodes the UAV true location. The adversarial actions are limited to one directional increment at most. If, for example, the UAV is heading $N$, then the adversary set of actions is $\beta(N) = \{N, NE, NW\}$. The auxiliary components $\mathcal{M}_{\text{mwr}}$ and $\mathcal{M}_{\text{mrd}}$ manage a FIFO memory stack $(m_i)_{i=0}^{n-1} \in A_{\text{uav}}^n$. The last UAV movement is saved in $m_i$ by synchronizing $\mathcal{M}_{\text{mwr}}$ with $\mathcal{M}_{\text{uav}}$ via write, while $\mathcal{M}_{\text{mrd}}$ synchronizes with $\mathcal{M}_{\text{adv}}$ via read to read the next UAV action from $m_j$. The subgame terminates whenever action write is attempted and $\mathcal{M}_{\text{mwr}}$ is at state $n$ (i.e., out of memory).

The goal is to find strategies for the UAV-AS coalition based on the following:

- **Target reachability.** To overcome cases where targets are unreachable due to hazard zones, the label *reach* is assigned to the set of states with acceptable checkpoint locations (including the target) to render the objective incrementally feasible. The objective for all encountered subgames is then formalized as

$$\Pr_{\text{max}}[F \text{ reach}] \geq p_{\text{min}}$$

for some bound $p_{\text{min}}$.

---

7To detect aggressive attacks, techniques from literature (e.g., [PLP17, PWB+17, JP19]) can be used.
Figure 2.9: Primary DAG components: UAV ($\mathcal{M}_{uav}$), adversary ($\mathcal{M}_{adv}$), and AS ($\mathcal{M}_{as}$). Auxiliary DAG components: memory write ($\mathcal{M}_{mwr}$) and memory read ($\mathcal{M}_{mrd}$) models, capturing the DAG representation. At stage $i$, the next memory location to write/read is $m_i$.

- **Hazard Avoidance.** Similar to target reachability, the label hazard is assigned to states corresponding to hazard zones. The objective $\Pr_{\max} [G \neg \text{hazard}] \geq p_{\min}$ is then specified for all encountered subgames.

By refining the aforementioned objectives, synthesis queries are used for both the subgames and the supergame. Specifically, the query

$$\phi_{\text{syn}}(k) := \langle \langle \text{uav} \rangle \rangle \Pr_{\max = ?} [\neg \text{hazard} \mathcal{U}^k (\text{locate} \land \text{reach})] \quad (2.1)$$

is specified for each encountered subgame $\hat{G}_i$, where locate indicates a successful geolocation task. By following Algorithm 2 for a $q$ number of reachable subgames, the supergame is reduced to an MDP $\mathcal{G}_D^\{\pi_i\}_{i=1}^q$ (whose states are the reachable subgames), which is checked against the query

$$\phi_{\text{ana}}(n) := \langle \langle \text{adv} \rangle \rangle \Pr_{\min,\max = ?} [\text{F}^{\leq n} \text{target}] \quad (2.2)$$

to find the bounds on the probability that the target is reached under a maximum number of geolocation tasks $n$. 30
2.5.2 Experimental Results

Fig. 2.10(a) shows the map setting used for implementation. The UAV’s ability to actively detect an attack depends on both its belief and the ground truth. Specifically, the probability of success in a geolocation task mainly relies on the disparity between the belief and true locations, captured by \( f_{\text{dis}}: \text{Ev}(x_B) \times \text{Ev}(x_T) \to [0, 1] \), obtained by assigning probabilities for each pair of locations according to their features (e.g., landmarks) and smoothed using a Gaussian 2D filter. A thorough experimental analysis where probabilities are extracted from experiments with human operators is described in [EZCP19a]. The set of hazard zones include the map boundaries to prevent the UAV from reaching boundary values. Also, the adversary is prohibited from launching attacks for at least the first step, a practical assumption to prevent the UAV model from infinitely bouncing around the target location.

We implemented the model in PRISM [CFK+13, KPW18] and performed the experiments on an Intel Core i7 4.0 GHz CPU, with 10GB RAM dedicated to the tool. Fig. 2.10(b) shows the supergame obtained by following the procedure in Algorithm 2. A vertex \( \hat{G}_{xy} \) represents a subgame (composed with its strategy) that starts at location \((x, y)\), while the outgoing edges point to subgames reachable from the current location.
one. Note that each edge represents a probabilistic transition. Subgames with more than one outgoing transition imply nondeterminism that is resolved by the adversary actions. Hence, the directed graph depicts an MDP.

The synthesized strategy for \((h_{adv}=2, h=4)\) is demonstrated in Fig. 2.10(c). For the initial subgame, Fig. 2.11(a) shows the maximum probability of a successful geolocation task if performed at stage \(h\), and the remaining distance to target. Assuming the adversary can launch attacks after stage \(h_{adv}=2\), the detection probability is maximized by performing the geolocation task at step 4, and hazard areas can still be avoided up till \(h=6\). For \(h_{adv}=1\), however, \(h=3\) has the highest probability of success, which diminishes at \(h=6\) as no possible flight plan exists without encountering a hazard zone. The effect of the maximum number of geolocation tasks \((n)\) on target reachability is studied by analyzing the supergame against \(\phi_{ana}\) as shown in Fig. 2.11(b). The minimum number of geolocation tasks to guarantee a non-zero probability of reaching the target (regardless of the adversary strategy) is 3 with probability bounds of \((33.7\%, 94.4\%)\).

The experimental data obtained for this case study are listed in Table 2.1. For the same grid size, more complex maps require more time for synthesis while the
Table 2.1: Results for strategy synthesis using queries $\phi_{syn}$ and $\phi_{ana}$.

<table>
<thead>
<tr>
<th>Subgame $\hat{G}_1$</th>
<th>Model Size</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Transitions</td>
</tr>
<tr>
<td>Map $t_{adv}$</td>
<td>$k$</td>
<td></td>
</tr>
<tr>
<td>8 x 8</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supergame $\hat{G}_0$</th>
<th>States</th>
<th>Transitions</th>
<th>Choices</th>
<th>Model</th>
<th>$\phi_{syn}$</th>
<th>$\phi_{ana}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6,212</td>
<td>8,306</td>
<td>6,660</td>
<td>2.216</td>
<td>-</td>
<td>2.490</td>
</tr>
</tbody>
</table>

state space size remains unaffected. The state space grows exponentially with the explored horizon size, i.e., $\mathcal{O}(\lvert A_{uav} \rvert \lvert A_{adv} \rvert)^h$, and is typically slowed by, e.g., the presence of hazard areas, since the branches of the game transitions are trimmed upon encountering such areas. Interestingly, for $h=6$ and $h=7$, while the model construction time (size) for $h_{adv}=1$ is almost twice (quadruple) as those for $h_{adv}=2$, the time for checking $\phi_{syn}$ declines in comparison. This reflects the fact that, in case of $h_{adv}=1$ compared to $h_{adv}=2$, the UAV has higher chances to reach a hazard zone for the same $k$, leading to a shorter time for model checking.

2.6 Discussion and Conclusion

In this chapter, we introduced DAGs and showed how they can simulate HIGs by delaying players’ actions. We also derived a DAG-based framework for strategy synthesis and analysis using off-the-shelf SMG model checkers. Under some practical assumptions, we showed that DAGs can be decomposed into independent subgames, utilizing parallel computation to reduce the time needed for model analysis, as well as the size of the state space. We further demonstrated the applicability of the proposed framework on a case study focused on synthesis and analysis of active attack detection strategies for UAVs prone to cyber-attacks.
DAGs come at the cost of increasing the total state space size as $\mathcal{M}_{mrd}$ and $\mathcal{M}_{mwr}$ are introduced. This does not present a significant limitation due to the compositional approach towards strategy synthesis using subgames. However, the synthesis is still limited to model sizes that off-the-shelf tools can handle.

The concept of delaying actions implicitly assumes that the adversary knows the UAV actions a priori. This does not present a concern in the presented case study as an abstract (i.e., nondeterministic) adversary model is analogous to synthesizing against the worst-case attacking scenario. Nevertheless, strategies synthesized using DAGs (and SMGs in general) are inherently conservative. Depending on the considered system, this can easily lead to no feasible solution.

The proposed synthesis framework ensures preservation of safety properties. Yet, general reward-based strategy synthesis is to be approached with care. For example, rewards dependent on the belief can appear in any state, and exploring hypothetical branches is not required. However, rewards dependent on a state’s true value should only appear in proper states, and all hypothetical branches are to be explored. A detailed investigation of how various properties are preserved by DAGs, along with multi-objective synthesis, is a direction for future work.
Chapter 3

Context-Aware Probabilistic Temporal Logic

This chapter is an adapted reproduction of “Mahmoud Elfar, Yu Wang, and Miroslav Pajic. Context-aware temporal logic for probabilistic systems. In International Symposium on Automated Technology for Verification and Analysis, pages 215–232, Springer, 2020” [EWP20]; and hence is not available under a Creative Commons license.

3.1 Introduction

The correct-by-design paradigm in Cyber-Physical Systems (CPS) has been a central concept during the design phase of various system components. This paradigm requires the abstraction of both the system behavior and design requirements [NSKB03, PMS+12]. Typically, the system behavior is modeled as a discrete Kripke structure, with nondeterministic transitions representing various actions or choices that need to be resolved. In systems where probabilistic behavior is prevalent, formalisms such as Markov decision processes (MDPs) are best suited. The applications of correct-by-design synthesis paradigm span CPS fields such as robot path and behavior planning [BWZP20, KGLR18], smart power grids [PSVS14], safety-critical medical devices [JPM+12], and autonomous vehicles [SSS15].

Temporal Logics (TLs) can be utilized to formalize CPS design requirements. For example, Linear Temporal Logic (LTL) [BKL08] is used to capture safety and reachability requirements over Boolean predicates defined over the state space. Similarly, computation tree logic (CTL) [BKL08] allows for expressing requirements over all computations branching from a given state. Probabilistic computation tree logic
(PCTL) can be viewed as a probabilistic variation of CTL to reason about the satisfaction probabilities of temporal requirements.

The choice of which TL to use is both a science and an art. Nevertheless, fundamental factors include expressiveness (i.e., whether the design requirements of interest can be expressed by the logic), and the existence of model checkers that can verify the system model against the design requirement, synthesize winning strategies, or generate counterexamples. Although prevalent TLs can be inherently expressive, two notions are oftentimes overlooked, namely, how easy it is to correctly formalize the design requirements, and whether existing model checkers are optimized for such requirements. The more complex it becomes to formalize a given requirement, the more likely it is that human error is introduced in the process.

In particular, we focus in this chapter on requirements that are naturally specified as a set of various objectives with an underlying priority structure. For instance, the objective of an embedded controller might be focused on achieving a primary task. However, whenever the chances of achieving such task fall below a certain threshold, the controller shall proceed with a fail-safe procedure. Such requirement, while being easy to state and understand, can prove challenging when formalized for two reasons. First, multiple objectives might be involved with a priority structure, i.e., one objective takes priority over another. Second, the context upon which the objectives are switched is of probabilistic nature, i.e., it requires the ability to prioritize objectives based on probabilistic invariants.

To this end, in this work we consider the problem of modeling and synthesis of CPS modeled as MDPs, with context-based probabilistic requirements, where a context is defined over probabilistic conditions. We tackle this problem by introducing the context-aware probabilistic temporal logic (CAPTL). CAPTL provides intuitive means to formalize design requirements as a set of objectives with a priority struc-
ture. For example, a requirement can be defined in terms of primary and secondary objectives, where switching from the former to the latter is based upon a probabilistic condition (i.e., a context). The ability to define context as probabilistic conditions sets CAPTL apart from similar TLs.

In addition to providing the syntax and semantics of CAPTL for MDPs, we investigate the problem of synthesizing winning strategies based on CAPTL requirements. Next, we demonstrate how the synthesis problem can be reduced to a set of PCTL-based synthesis sub-problems. Moreover, for deterministic CAPTL requirements with persistence objectives, we propose an optimized synthesis algorithm. Finally, we implement the algorithm on top of PRISM-games [KPW18], and we show experimental results for two case studies where we synthesize a robotic task planner, and an error-resilient scheduler for microfluidic biochips.

The rest of this section discusses related work. Preliminaries and a motivating example are provided in Sec. 3.2. In Sec. 3.3 we introduce the syntax and semantics of CAPTL. The CAPTL-based synthesis problem is introduced in Sec. 3.4, where we first explore how a CAPTL requirement can be approached using PCTL, followed by our proposed synthesis algorithm. For evaluation, we consider two case studies in Sec. 3.5. Finally, we conclude the chapter in Sec. 3.6.

3.1.1 Related Work

The problem of multi-objective model checking and synthesis has been studied in literature, spanning both MDPs and stochastic games, for various properties, including reachability, safety, probabilistic queries, and reward-based requirements [EKVY07, FKN+11, FKP12]. Our work improves upon the multi-objective synthesis paradigm by enabling priorities over the multiple objectives as we will show in Sec. 3.2. One prevalent workaround is to define multiple reward structures, where states are as-
signed tuples of real numbers depicting how favorable they are with respect to multiple criteria. The synthesis problem is then reduced to an optimization problem over either a normalized version of the rewards (i.e., assigning weights), or one reward with logical constraints on the others [BDK+17, BKN16]. Results are typically presented as Pareto curves, depicting feasible points in the reward space [FKPI2]. Our work differs in two aspects. First, we use probabilities as means to define priorities rather than reward structures. Second, the mechanics needed to define context-based priorities are an integral part of CAPTL.

Perhaps the closest notion to our context-based prioritization scheme are probabilistic invariant sets (PIS) [KDDS12]. Both CAPTL and PIS involve the identification of state-space subsets that maintain a probability measure within specific bounds. While prevalent in the field of probabilistic programs [BEFH16], PIS was not considered in the field of CPS synthesis, despite the fact that (non-probabilistic) invariant sets are used in controller design [Bla99]. The problem of merging strategies for MDPs that correspond to different objectives has been investigated [BBG97, Wil15]. Our approach, however, is primarily focused on formalizing the notion of context-based priorities within the specification logic itself rather than altering the original model. While one can argue that PCTL alone can be used to define priorities by utilizing nested probabilistic operators, the nesting is typically limited to qualitative operators [LAB11]. In contrast, CAPTL relaxes such limitation by allowing quantitative operators as well. Moreover, CAPTL-based synthesis provides an insight into which objective is being pursued at a given state.
3.2 Problem Setting

3.2.1 Preliminaries

For a measurable event $E$, we denote its probability by $\Pr(E)$. The powerset of $A$ is denoted by $\mathcal{P}(A)$. We use $\mathbb{R}$, $\mathbb{N}_0$ and $\mathbb{B}$ for the set of reals, naturals and Booleans, respectively. For a sequence or a vector $\pi$, we write $\pi[i], i \in \mathbb{N}_0$, to denote the $i$-th element of $\pi$.

We formally model the system as an MDP. MDPs feature both probabilistic and nondeterministic transitions, capturing both uncertain behaviors and nondeterministic choices in the modeled system, respectively. We adopt the following definition for a system model as an MDP [BKL08].

**Definition 11** (System Model). A system model is an MDP $\mathcal{M} = (S, \text{Act}, P, s_0, AP, L)$ where $S$ is a finite set of states; Act is a finite set of actions; $P : S \times \text{Act} \times S \to [0, 1]$ is a transition probability function s.t. $\sum_{s' \in S} P(s, a, s') \in \{0, 1\}$ for $a \in \text{Act}; s_0$ is an initial state; $AP$ is a set of atomic propositions; and $L : S \to \mathcal{P}(AP)$ is a labeling function.

Given a system $\mathcal{M}$, a path is a sequence of states $\pi = s_0s_1\ldots$, such that $P(s_i, a_i, s_{i+1}) > 0$ where $a_i \in \text{Act}(s_i)$ for all $i \geq 0$. The trace of $\pi$ is defined as $\text{trace}(\pi) = L(s_0)L(s_1)\ldots$. We use $F\text{Path}_{\mathcal{M},s}$ ($I\text{Path}_{\mathcal{M},s}$) to denote the set of all finite (infinite) paths of $\mathcal{M}$ starting from $s \in S$. We use $\text{Paths}_{\mathcal{M},s}$ to denote the set of all finite and infinite paths starting from $s \in S$. If $P(s, a, s') = p$ and $p > 0$, we write $s \xrightarrow{a,p} s'$ to denote that, with probability $p$, taking action $a$ in state $s$ will yield to state $s'$. We define the cardinality of $\mathcal{M}$ as $|\mathcal{M}| = |S| + |P|$, where $|P|$ is the number of non-zero entries in $P$.

A strategy (also known as a policy or a scheduler) defines the behavior upon which nondeterministic transitions in $\mathcal{M}$ are resolved. A memoryless strategy uses only the
current state to determine what action to take, while a \textit{memory-based} strategy uses
previous states as well. We focus in this work on pure memoryless strategies, which
are shown to suffice for PCTL reachability properties [BKL08].

\textbf{Definition 12 (Strategy).} A \textit{(pure memoryless)} strategy of $\mathcal{M} = (S, Act, P, s_0, AP, L)$ is a function $\sigma : S \rightarrow Act$ that maps states to actions.

By composing $\mathcal{M}$ and $\sigma$, nondeterministic choices in $\mathcal{M}$ are resolved, reducing
the model to a \textit{discrete-time Markov chain} (DTMC), denoted by $\mathcal{M}\sigma$. We use $Pr_{\mathcal{M},s}^\sigma$ to denote the probability measure defined over the set of infinite paths $IPath_{\mathcal{M},s}^\sigma$.

The function $reach(\mathcal{M}, s, \sigma)$ denotes the set of reachable states in $\mathcal{M}$ starting from $s \in S$ under strategy $\sigma$, while $reach(\mathcal{M}, s)$ denotes the set of all reachable states from $s$ under any strategy.

We use \textit{probabilistic computation tree logic} (PCTL) to formalize system objectives
as temporal properties with probabilistic bounds, following the grammar

\[ \Phi ::= \top \mid a \mid \neg \Phi \mid \Phi \land \Phi \mid \mathbb{P}_J[\phi], \quad \phi ::= X\Phi \mid \Phi U \Phi \mid \Phi U \leq k \Phi, \]

where $J \subseteq [0, 1]$, and $X$ and $U$ denote the \textit{next} and \textit{until} temporal modalities, respectively. Other derived modalities include $\Diamond$ (eventually), $\Box$ (always), and $W$ (weak until). Given a system $\mathcal{M}$ and a strategy $\sigma$, the PCTL satisfaction semantics over $s \in S$ and $\pi \in Paths_{\mathcal{M},s}^\sigma$ is defined as follows [BKL08, FKNP11]:

- $s, \sigma \models a \iff a \in L(s)$
- $s, \sigma \models \neg \Phi \iff s \not\models \Phi$
- $s, \sigma \models \Phi_1 \land \Phi_2 \iff s \models \Phi_1 \land s \models \Phi_2$
- $s, \sigma \models \mathbb{P}_J[\phi] \iff Pr\{\pi \mid \pi \models \phi\} \in J$
- $\pi, \sigma \models X\Phi \iff \pi[1] \models \Phi$
- $\pi, \sigma \models \Phi_1 U \Phi_2 \iff \exists j \geq 0. (\pi[j] \models \Phi_2 \land (\forall 0 \leq k < j. \pi[k] \models \Phi_1))$
- $\pi, \sigma \models \Phi_1 U \leq n \Phi_2 \iff \exists 0 \leq j \leq n. (\pi[j] \models \Phi_2 \land (\forall 0 \leq k < j. \pi[k] \models \Phi_1))$

PCTL can be extended with \textit{quantitative queries} of the form $\mathbb{P}_{\min}[^\Phi]$ ($\mathbb{P}_{\max}[\Phi]$) to
compute the minimum (maximum) probability of achieving $\varphi$ [FKNP11, SK16], i.e.,

$$P_{\text{min}}[\varphi] = \inf_{\sigma \in \Sigma} \Pr_{\mathcal{M},s}[\pi | \pi \models \varphi], \quad P_{\text{max}}[\varphi] = \sup_{\sigma \in \Sigma} \Pr_{\mathcal{M},s}[\pi | \pi \models \varphi].$$

We will denote such queries as $P_{\text{opt}}$ (read: optimal), where opt $\in \{\text{max}, \text{min}\}$.

### 3.2.2 Motivating Example

Consider the simple grid-world shown in Fig. 3.1(left). The robot can move between rooms through doorways where obstacles can be probabilistically encountered (e.g., closed doors), requiring the robot to consume more power. The robot state is captured as a tuple $s: (g, h, x, y)$, where $g \in \{\text{on, sleep, error}\}$ is the robot’s status, $h \in \{0, 1, \ldots, 10\}$ is the robot’s battery level, and $x$ and $y$ are its current coordinates.

As shown in Fig. 3.1(right) the system can be modeled as $\mathcal{M} = (S, Act, P, s_0, AP, L)$, where $Act = \{a_N, a_S, a_E, a_W, \text{sleep, error}\}$, and $s_0 = (0, 10, 1, 1)$. Suppose that the main objective for the robot is to reach the goal with a charge $h > 3$ (objective A). However, if the probability of achieving objective A is less than 0.8, the robot should prioritize reaching the charging station and switch to sleep mode (objective B). Moreover, if the probability of achieving objective B falls below 0.7, the robot should stop and switch to err mode, preferably in one of the safe zones (objective C).

Now let us examine how such requirements can be formalized. Let $\varphi_A = \Diamond (\text{goal} \land \ldots$
(h > 3) ∧ on), \( \varphi_B = \Diamond (\text{chrg} \land (h > 3) \land \text{on}) \), and \( \varphi_C = \Diamond (\text{error}) \). One can use PCTL to capture each objective separately as the reachability queries \( \Phi_A = \mathbb{P}_{\max}[\varphi_A] \), \( \Phi_B = \mathbb{P}_{\max}[\varphi_B] \), and \( \Phi_C = \mathbb{P}_{\max}[\varphi_C] \). A multi-objective query \( \Phi_1 = \Phi_A \lor \Phi_B \lor \Phi_C \) does not capture the underlying priority structure in the original requirements. In fact, an optimal strategy for \( \Phi_1 \) always chooses the actions that reflect the objective with the highest probability of success, resulting in a strategy where the robot simply signals an error from the very initial state. Similarly, the use of \( \Phi_2 = \mathbb{P}_{\max}[\varphi_A \mathcal{W} \varphi_B] \) does not provide means to specify the context upon which switching from \( \varphi_A \) to \( \varphi_B \) occurs. Attempts featuring multi-objective queries with nested operators, such as \( \Phi_3 = \mathbb{P}_{\max}[\varphi_A \land \mathbb{P}_{\max}[\geq 0.8[\varphi_A]]] \lor \mathbb{P}_{\max}[\varphi_B \land \mathbb{P}_{\max}[< 0.8[\varphi_A]]] \), have several drawbacks. First, correctly formalizing the requirement is typically cumbersome and hard to troubleshoot. Second, to the best of our knowledge, nested queries in the form of \( \mathbb{P}_{\text{opt} \in J} \) are not supported by model checkers. Third, the semantics of the formalized requirement is potentially different from the original one. For instance, \( \Phi_3 \) allows the system to pursue \( \varphi_A \) even after switching to \( \varphi_B \) if the probability of achieving \( \varphi_A \) rises again above 0.8 — a behavior that was not called for in the original requirement.

Consequently, in this chapter we focus on two problems: the formalization of PCTL objectives with an underlying context-based priority structure, and the synthesis of strategies for such objectives. The first problem is addressed by introducing CAPTL in Sec. 3.3, while the second is addressed in Sec. 3.4. We will use this motivating example as a running one throughout the rest of this chapter.

### 3.3 Context-Aware Temporal Logic

#### 3.3.1 CAPTL Syntax

CAPTL features two pertinent notions, namely, objectives and contexts. Let \( \mathcal{M} \) be our system model, and let \( \Xi \) be the set of all possible PCTL path formulas defined
for $M$. In CAPTL, we define an objective $q$ as a conjunctive optimization query $q = \bigwedge_{i=1}^{m} P_{\text{opt}} [\varphi_i], \varphi_i \in \Xi$, $m > 0$. When $m > 1$, $q$ resembles a multi-objective optimization query in the conjunctive form. Otherwise, in the simplest form where $m = 1$, $q$ is a single-objective query.

A context $w_{(q,q')}$ marks a state where switching from objective $q$ to objective $q'$ is required. Formally, we define a context $w$ over $\Xi$ as a set of satisfaction queries in the disjunctive normal form $w = \bigvee_{j=1}^{n} \bigwedge_{i=1}^{m} P_{\text{opt} \in J_{ij}} [\varphi_{i,j}], \varphi_{ij} \in \Xi, J \subseteq [0,1]$. Intuitively, in a state where $w_{(q,q')}$ is satisfied, the system switches from $q$ to $q'$. Notice that the context definition utilizes the operator $P_{\text{opt} \in J_{ij}}$ with an interval, i.e., a context is evaluated at a given state as a Boolean value in $\mathbb{B}$. In contrast, the objective definition utilizes the operator $P_{\text{opt}}$ without intervals, i.e., a quantitative optimization query that can return a numerical value in $[0,1]$.

A CAPTL requirement defines a set of objectives to be satisfied, in addition to a set of contexts, representing the probabilistic conditions upon which objectives are prioritized. Formally, we define the syntax of a CAPTL requirement as follows.

**Definition 13 (CAPTL Requirement).** Given a set of PCTL path formulas $\Xi$, a CAPTL requirement is a tuple $\mathcal{A} = (Q, W, \Xi, \rightarrow, q_0)$ where

- $Q \subset \{ \bigwedge_{i=1}^{m} P_{\text{opt}} [\varphi_i] \mid \varphi_i \in \Xi \}$ is a finite nonempty set of objectives over $\Xi$,
- $W \subset \left\{ \bigvee_{j=1}^{n} \bigwedge_{i=1}^{m} P_{\text{opt} \in J_{ij}} [\varphi_{i,j}] \mid \varphi_{ij} \in \Xi, J_{ij} \subseteq [0,1] \right\}$ is a set of contexts,
- $\rightarrow \subseteq Q \times W \times Q$ is a conditional transition relation, and
- $q_0 \in Q$ is an initial objective.

In a CAPTL requirement $\mathcal{A}$, each state $q \in Q$ represents an objective, i.e., an optimization query to be satisfied. The conditional transition relation $\rightarrow$ defines how objectives are allowed to change. For instance, if $q \xrightarrow{w} q'$, a shorthand for $(q, w, q') \in \rightarrow$,
then the objectives are switched from \( q \) to \( q' \) if \( w \) is satisfied. Notice that contexts are used as labels for the conditional transition relation. In the rest of this chapter, we will overload the notation and use \( W : Q \rightarrow \mathcal{P}(W) \) to denote the set of contexts emerging from a given objective. We will also use \( Q(q, w) = q' \) to denote that objective \( q \) has a context \( w \) that leads to \( q' \).

**Example 4.** For the running example, Fig. 3.2 shows an example of a CAPTL requirement \( A = (Q, W, \Xi, \rightarrow, q_0) \) where \( Q = \{q_0, q_1, q_2, q_3\} \), \( W = \{w_{01}, w_{02}, w_{13}, w_{23}\} \), and \( \rightarrow = \{(q_0, w_{01}, q_1), (q_0, w_{02}, q_2), (q_1, w_{13}, q_3), (q_0, w_{23}, q_3)\} \). The requirement starts by prioritizing \( q_0 = P_{\text{max}}[\varphi_0] \). If \( P_{\text{max}}[\varphi_0] \in [0.75, 0.85) \), the context \( w_{01} \) becomes true, and by executing \( q_0 \xrightarrow{w_{01}} q_1 = P_{\text{max}}[\varphi_1] \) is prioritized. Similarly, if \( P_{\text{max}}[\varphi_0] \in [0, 0.75) \), \( w_{02} \) becomes true, executing \( q_0 \xrightarrow{w_{02}} q_2 \) where \( q_2 = P_{\text{max}}[\varphi_2] \) is prioritized. Notice that objectives can have a single context, e.g., \( W(q_1) = \{w_{13}\} \); multiple contexts, e.g., \( W(q_0) = \{w_{01}, w_{02}\} \); or none, e.g., \( W(q_3) = \emptyset \).

### 3.3.2 CAPTL Semantics for MDPs

We progressively define CAPTL semantics for MDPs by first defining the satisfaction semantics for objectives and contexts. Let \( q = P_{\text{max}}[\varphi] \) be the objective at state \( s \), and let \( \Sigma \) be the set of all strategies for \( \mathcal{M} \). We say that \( s, \sigma^* \models q \) if \( \sigma^* \in \Sigma \) such
that
\[
\Pr_{\mathcal{M}}^{\sigma^*, s} = \sup_{\sigma \in \Sigma} \Pr_{\mathcal{M}, s}^\sigma \left( \{ \pi \in \text{Paths}_{\mathcal{M}, s}^\sigma \mid \pi \models \varphi \} \right).
\] (3.1)

In that case, we call \(\sigma^*\) a local strategy, i.e., an optimal strategy w.r.t. \(\langle q, s \rangle\).

**Definition 14 (Local Strategy).** Let \(q_i = \mathbb{P}_{\text{opt}} \left[ \varphi_i \right]\) be an objective. A local (optimal) strategy for \(\langle q_i, s_i \rangle\) is a strategy \(\sigma_{\langle q_i, s_i \rangle} \in \Sigma\) such that
\[
\Pr_{\mathcal{M}, s_i}^{\sigma_{\langle q_i, s_i \rangle}} = \sup_{\sigma \in \Sigma} \Pr_{\mathcal{M}, s_i}^\sigma \left( \{ \pi \in \text{Paths}_{\mathcal{M}, s_i}^\sigma \mid \pi \models \varphi_i \} \right)
\]

Next, let \((q, w, q') \in \rightarrow\), where \(w = \mathbb{P}_{\leq c} [\varphi]\). Let \(s_k \in \text{reach}(\mathcal{M}, s, \sigma^*)\), where \(\sigma^*\) is the local strategy for \(\langle q, s \rangle\). We say that \(s_k \models w\) if
\[
\sup_{\sigma \in \Sigma} \Pr_{\mathcal{M}, s_k}^\sigma \left( \{ \pi \in \text{Paths}_{\mathcal{M}, s_k}^\sigma \mid \pi \models \varphi \} \right) \leq c.
\] (3.2)

Note that contrary to (3.1), the set of paths \(\{ \pi \}\) in (3.2) is not limited to those induced by the local strategy \(\sigma^*\). Moreover, if \(\exists \pi = s \ldots s_i \ldots s_k \in FPath_{\mathcal{M}, s}^{\sigma^*}\) s.t. \(s_i \models w\), and \(s_i \not\models w\) for all \(i < k\), then \(s_k\) is called a switching state, i.e., the first state on a path \(\pi\) to satisfy \(w\), triggering a switch from \(q\) to \(q'\).

**Definition 15 (Switching Set).** Let \(q = \mathbb{P}_{\text{opt}} [\varphi]\) and \(\sigma^* \in \Sigma\) such that \(s_0, \sigma^* \models q\). The corresponding switching set \(S_q \subseteq \text{reach}(\mathcal{M}, s_0, \sigma^*)\) is defined as
\[
S_q = \left\{ s_k \mid \exists \pi = s_0 \ldots s_i \ldots s_k \in FPath_{\mathcal{M}, s_0}^{\sigma^*} \text{ s.t. } s_i \not\models w, \forall i < k; s_k \models \bigvee_{w \in W(q)} w \right\}.
\]

We use \(S_q^d\) to denote the set of switching states from \(q\) to \(q'\).

An objective is active in a state \(s\) if it is being pursued at that state.

**Definition 16 (Active Objective).** Let \(\mathcal{A} = (Q, W, \Xi, \rightarrow, q_0)\) and \(\mathcal{M} = (S, \text{Act}, P, s_0, AP, L)\). An activation function \(g : S \rightarrow \mathcal{P}(Q)\) is defined inductively as: (i) \(g(s_0) \ni q_0;\)
and (ii) \( g(s) \ni q' \) if \( g(s) \ni q \) and \( s \in S_{q'} \). We say objective \( q \in Q \) is active at state \( s \in S \) if \( g(s) \ni q \).

As captured in Definition 14, local strategies are tied to their respective objectives. Consequently, a local strategy is switched whenever an objective is switched as well, and the new local strategy substitutes its predecessor. We call the set of local strategies a strategy profile, and the resulting behavior a protocol.

**Definition 17 (Protocol).** Let \( \mathcal{A} = (Q, W, \Xi, \rightarrow, q_0) \) and \( \mathcal{M} = (S, \text{Act}, \mathcal{P}, s_0, \text{AP}, L) \). Given a strategy profile \( \sigma = \{ \sigma_{(q,s)} \ldots \} \), the induced (optimal) protocol is a (partial) function \( \Pi : Q \times S \rightarrow \text{Act} \cup \mathcal{P}(W) \) such that

- \( \Pi(q,s) = \sigma_{(q,s)}(s) \in \text{Act} \) iff \( q \in g(s) \) and \( s \notin S_q \); and
- \( \Pi(q,s) \ni w_{(q,q')} \), where \( w_{(q,q')} \in W \), iff \( q \in g(s) \) and \( s \in S_{q'} \).

Given \( (q,s) \), a protocol assigns either an optimal action based on the local strategy associated with \( q \), or a context to switch the active objective itself. We will use \( \mathcal{P} \) to denote the set of all possible protocols.

**Definition 18 (System-Protocol Composition).** Let \( \mathcal{M} = (S, \text{Act}, \mathcal{P}, s_0, \text{AP}, L) \) and \( \Pi : Q \times S \rightarrow \text{Act} \cup \mathcal{P}(W) \) be a compatible protocol. Their composition is defined as \( \mathcal{M}^{\Pi} = (Q, \text{Act} \cup W, \mathcal{P}, \hat{s}_0, \hat{L}) \) where \( \hat{Q} \subseteq Q \times S \), \( \hat{s}_0 = (q_0, s_0) \), and

\[
\hat{P}((q,s), a, (q', s')) = \begin{cases} 
P(s,a,s') & \text{if } \Pi(q,s) = a, q' = q, \\
1 & \text{if } \Pi(q,s) = w, s' = s, q' = Q(q,w), \\
0 & \text{otherwise}.
\end{cases}
\]

We now define the CAPTL satisfaction semantics as follows.

**Definition 19 (CAPTL Satisfaction Semantics).** Let \( \mathcal{A} = (Q, W, \Xi, \rightarrow, q_0) \), \( \mathcal{M} = (S, \text{Act}, \mathcal{P}, s_0, \text{AP}, L) \), and \( \Pi : Q \times S \rightarrow \text{Act} \cup \mathcal{P}(W) \). The CAPTL satisfaction semantics
is defined inductively as follows:

\[
\begin{align*}
\mathcal{M}, \Pi \models q & \iff \Pr_{\mathcal{M}^n} (\{ \pi \in \text{Paths}_{\mathcal{M}^n} \mid \text{last}(\pi) = (q, s'), s' \models q \}) \geq 1, \\
\mathcal{M}, \Pi \models_c \mathcal{A} & \iff \Pr_{\mathcal{M}^n} (\{ \pi \in \text{Paths}_{\mathcal{M}^n} \mid \text{last}(\pi) = (q, s'), s' \models q, q \in Q \}) = c, \\
\mathcal{M}, \Pi \models \mathcal{A} & \iff \mathcal{M}, \Pi \models_{\geq 1} \mathcal{A}.
\end{align*}
\]

CAPTL semantics dictate that \( \mathcal{M} \) and \( \Pi \) satisfy \( \mathcal{A} \) if every path \( \pi \in \text{Paths}_{\mathcal{M}^n} \) ends with a state \( s \in S \) where \( q \ni g(s) \) and \( s \models q \), i.e., the system reaches some state \( s \) where some objective \( q \) is both active and satisfied.

### 3.3.3 CAPTL Fragments

A CAPTL requirement is nondeterministic if for some \( q \in Q \), \( \exists w_i, w_j \in W(q) \) such that \( S^q_i \cap S^q_j \neq \emptyset \). That is, at least one objective has two or more contexts that can be active at the same state. If that is not the case, then the CAPTL requirement is deterministic. We now identify a fragment of deterministic CAPTL requirements where the following two conditions are met. First, every \( q \in Q \) is a quantitative PCTL persistence objective. Second, every \( w \in W(q) \) is a qualitative PCTL persistence objective over the same persistence set as in \( q \). This is formally captured in the following definition.

**Definition 20** (Persistence CAPTL). A CAPTL requirement \( \mathcal{A} = (Q, W, \Xi, \rightarrow, q_0) \) is persistent if (i) every \( q \in Q \) is of the form \( q = \mathbb{P}_{\text{max}}[\Diamond \Box B] \) for some \( B \subseteq S \) and (ii) if \( W(q) \neq \emptyset \) then for any \( w_{(q,q_j)} \in W(q) \), it holds that \( w_{(q,q_j)} = \mathbb{P}_{\text{max} \in J_j}[\Diamond \Box B] \) where \( (J_j) \) are disjoint intervals satisfying \( \cup_j J_j = [0, c) \) for some \( 0 < c \leq 1 \).

A persistence CAPTL (P-CAPTL) requirement allows for defining persistence objectives, where each objective maximizes the probability of (i.e., prioritizes) reaching a corresponding persistence set. Contexts in this case can be understood as lower bounds of their respective objectives. That is, an objective is pursued as long as, at any transient state, the probability of achieving such objective does not drop below a
certain threshold. The requirement also ensures that at most one context is satisfied at any state, eliminating any nondeterminism in $A$.

**Example 5.** Continuing Example 4, Fig. 3.3 shows the persistence CAPTL requirement for the robot. Notice that all objectives are in the form $P_{\text{max}}[\Diamond \Box B]$. Also, the intervals $[0.75, 0.85)$ and $[0, 0.75)$ of $w_{01}$ and $w_{02}$, respectively, are disjoint, hence at most one context in $W(q_0) = \{w_{01}, w_{02}\}$ can be satisfied at any state.

### 3.4 CAPTL-Based Synthesis

In this section we first define the synthesis problem for CAPTL requirements. Next, we examine a general procedure for deterministic CAPTL where the synthesis problem is reduced to solving a set of PCTL-based strategy synthesis problems. Finally, we utilize the underlying structure of persistence properties to propose a synthesis procedure optimized for P-CAPTL requirements.

In the rest of this section, let $\mathcal{M} = (S, Act, P, s_0, AP, L)$ and $\mathcal{A} = (Q, W, \Xi, \rightarrow, q_0)$. We assume that a probabilistic model checker is given (e.g., PRISM-games $\text{KPW18}$ or UPPAAL STRATEGO $\text{DJL+15}$) that can accept an MDP-based model $\mathcal{M}$ and a PCTL formula $\Phi$ as inputs, and provides the following functions:

- $\text{Reach} :: (\mathcal{M}, s) \mapsto R \subseteq S$ returns the set of reachable states $R = \text{reach}(\mathcal{M}, s)$.
Verify :: \((M, s, \Phi) \mapsto b \in \mathbb{B}\) returns the Boolean value \(\top\) if \(M, s \models \Phi\), and returns \(\bot\) otherwise.

Synth :: \((M, s, \Phi) \mapsto (\sigma, c)\) returns a policy \(\sigma \in \Sigma\) s.t. \(\Pr(M^\sigma_s \models \Phi) = c\) for some \(c \in [0, 1]\).

We also assume that the model checker functions terminate in finite time and return correct answers. We now define the CAPTL synthesis problem as follows.

**Definition 21** (CAPTL Synthesis Problem). Given \(M = (S, \text{Act}, P, s_0, AP, L)\) and \(A = (Q, W, \Xi, \rightarrow, q_0)\), the CAPTL synthesis problem seeks to find a protocol \(\Pi : Q \times S \nrightarrow \text{Act} \cup W\) such that \(M, \Pi \models A\).

### 3.4.1 PCTL-Based Approach

The synthesis problem can be reduced to solving a set of PCTL-based synthesis queries as demonstrated in Algorithm 3. Starting with \(\langle q_0, s_0 \rangle\), the algorithm verifies whether any context \(w \in W(q_0)\) is satisfied, and if true, adds \(w\) to the protocol and switches to the next objective. If no context is satisfied, the algorithm synthesizes a local strategy and adds the corresponding optimal action to the protocol.

**Proposition 1.** Algorithm terminates; and returns \(\Pi, c\) iff \(M, \Pi \models c\).

### 3.4.2 Synthesis for P-CAPTL

We now propose a synthesis algorithm optimized for persistence CAPTL. To this end, we show that for a given persistence objective, synthesizing a local strategy in the initial state suffices. In a manner similar to switching states (see Definition 15), we devise a partition of reachable states for every objective. We will use those concepts to define a system-CAPTL composition and show that it is bisimilar to \(M^\Pi\).
Algorithm 3: PCTL-Based Synthesis

\begin{algorithm*}
\begin{algorithmic}
\STATE \textbf{Input:} $M = (S, Act, P, s_0, AP, L)$, $A = (Q, W, \Xi, \rightarrow, q_0)\\$
\STATE \textbf{Result:} $\Pi, c$ such that $M, \Pi \models_c A$
\STATE $q \leftarrow Q$ do $S_q \leftarrow \emptyset$, $\hat{S}_q \leftarrow \emptyset$
\STATE $\Pi \leftarrow \emptyset$, $\hat{S}_{q_0} \leftarrow \{s_0\}$, $q \leftarrow q_0$, $C \leftarrow 0_{Q \times S} \in [0, 1]^{Q \times S}$, repeat $\leftarrow \top$
\WHILE{$\hat{S}_q \neq \emptyset$}
\FORALL{$w \in W(q)$}
\IF{$\text{Verify}(M, s, w) = \top$}
\STATE $\Pi \leftarrow \Pi \cup \{(s, q, w)\}$, $q \leftarrow Q(q, w)$, repeat $\leftarrow \top$, break
\ELSE
\STATE $(\sigma, C(q, s)) \leftarrow \text{Synth}(M; s, q)$, $\Pi \leftarrow \Pi \cup \{(s, q, \sigma(s))\}$
\STATE $\hat{S}_q \leftarrow \hat{S}_q \cup (\text{Post}(M, s, \sigma(s)) \setminus \hat{S}_q)$
\ENDIF
\ENDFOR
\STATE $c \leftarrow \text{Verify}(M^\Pi, (q_0, s_0), \mathbb{P}[\bigvee_{q \in Q}((q, s) \land C(q, s) = 1)])$
\STATE $\text{repeat} \leftarrow \bot$
\ENDWHILE
\end{algorithmic}
\end{algorithm*}

Let $R = \text{reach}(M, s_0)$. We first note that given $M$ and $q = \mathbb{P}_{\text{opt}}[\lozenge \Box B]$, existing model checking and synthesis algorithms typically compute a least fixed point (LFP) vector $x_q \in [0, 1]^{|R|}$, where $x_q[s]$ is the optimal probability of satisfying $\lozenge \Box B$ at state $s \in R$ (e.g., see [BKLO8, KKNP10]). That is, when $\text{Synth}(M, s_0, q)$ is called, $x_q$ is computed, but only $c = x_q[s_0]$ is returned (i.e., the value at the initial state). We exploit this fact by implementing a function $\text{ReachP} :: (M, s, q) \mapsto x_q$ that returns the LFP vector $x_q$ associated with $q$.

Lemma 2 (Local Strategy Dominance). Let $M = (S, Act, P, s_0, AP, L)$ and $q = \mathbb{P}_{\text{max}}[\lozenge \Box B]$. For all $s \in \text{reach}(M, s_0)$, $\sigma_{(q, s)} = \sigma_{(q, s_0)}|_{\text{reach}(M, s)}$.

Lemma 2 signifies that a local strategy for $q$ in the initial state (i.e., $\sigma_{(q, s_0)}$) subsumes all local strategies for the same probabilistic reachability objective in every $s \in R$. Next, for every $q \in Q$, let us define the following partition of $R$:

- $R'_q = \{s \in R \mid \forall w = \mathbb{P}_{\text{max}}[\lozenge \Box B] \in W(q), x_q[s] \notin J\}$, i.e., the states in $R$ where, if $q$ is active, keep pursuing $q$.

- $R'_q = \{s \in R \mid \exists w = \mathbb{P}_{\text{max}}[\lozenge \Box B] \in W(q), x_q[s] \in J, Q(q, w) = q'\}$, i.e., the states in $R$ where, if $q$ is active, switch to $q'$.  

50
Lemma 3 (Partitioning). Let $\mathcal{M} = (S, \text{Act}, P, s_0, AP, L)$, $A = (Q, W, \Xi, \rightarrow, q_0)$, and $R = \text{reach}(\mathcal{M}, s_0)$. For every $q \in Q$, $\bigcup_{q' \in Q} R_q^{q'} = R$, and $R_q^{q'} \cap R_q^{q''} = \emptyset$ for every $q' \neq q''$.

Proof Sketch. From Definition 20, the intervals $(J_w)_{w \in W(q)}$ are disjoint; hence $(R_q^{q'})_{q' \neq q}$ are disjoint as well, and that $R_q^i = R/\left(\bigcup_{q' \neq q} R_q^{q'}\right)$.

Example 6. Returning to the P-CAPTL requirement specified in the running example (see Fig. 3.3), Fig. 3.4 depicts the partitioning of the state-space based on $q_0$, $q_1$, $q_2$ and $q_3$. Notice that for any $q \in Q$, the sets $(R_q^{q'})_{q' \in Q}$ are pairwise disjoint, where $\bigcup_{q' \in Q} R_q^{q'} = \text{reach}(\mathcal{M}, s_0)$. For example, $R_{q_0}^{q_0}$, $R_{q_0}^{q_1}$, and $R_{q_0}^{q_2}$ do not intersect, and their union spans $R = \text{reach}(\mathcal{M}, s_0)$. In this case, $R_{q_0}^{q_3} = \emptyset$ since there is no direct context emerging from $q_0$ to $q_3$.

Definition 22 (System-CAPTL Composition). Let $\mathcal{M} = (S, \text{Act}, P, s_0, AP, L)$, $A = (Q, W, \Xi, \rightarrow, q_0)$, and $\sigma = \{\sigma_{(q, s_0)} \mid q \in Q\}$. Their composition is defined as the automaton $\mathcal{M}'_{\mathcal{A}} = (V, \text{Act}, P, v_0, AP, \tilde{L})$ where $V \subseteq S \times Q \times \Gamma$, and $\Gamma = \{1, II\}$; $\text{Act} = \text{Act} \cup W \cup \{\tau\}$, where $\tau$ is a stutter action; $v_0 = \langle s_0, q_0, II \rangle$; $\tilde{L} : V \rightarrow P(AP)$ such that $\tilde{L}(\langle s, q, \gamma \rangle) = L(s)$; and the transition relation $\rightarrow'$ is defined using the fol-
following compositional rules:

\[
\begin{aligned}
&R1: \quad & s & \xrightarrow{a,p} s' \land \sigma_{(q,s_0)}(s) = a & \Rightarrow & \langle s, q, I \rangle & \xrightarrow{a,p'} & \langle s', q, II \rangle \\
&R2: \quad & s & \in R^q & \Rightarrow & \langle s, q, I \rangle & \xrightarrow{\tau'} & \langle s, q, I \rangle \\
&R3: \quad & s & \in R^q' & \Rightarrow & \langle s, q, I \rangle & \xrightarrow{w(s,q')} & \langle s, q', II \rangle.
\end{aligned}
\]

The rules in Definition 22 are interpreted as follows. The state space \( V \) is partitioned into \( V_I \) (where \( M \) actions are allowed) and \( V_{II} \) (where \( A \) actions are allowed), resembling a turn-based 2-player game. [R1] ensures that, if \( q \) is active in \( s \), then only the transitions with the optimal action \( \sigma_{(q,s_0)}(s) \) are allowed. [R2] ensures that, if \( s \in R^q \), the active objective remains unchanged. If \( s \in R^q' \), however, [R3] enforces switching the active objective to \( q' \). The action \( \tau \) is a stutter since \( \forall v \tau \rightarrow v' \), \( L(v) = L(v') \).

**Lemma 4** (Induced DTMC). \( M^\sigma_A \) constructed using Definition 22 is a DTMC.

Lemma 4 dictates that the probability measure \( \Pr_{M^\sigma_A} \) is well-defined. We will now use the notion of stutter equivalence [BKL08] to prove that \( M^\sigma_A \) is bisimilar to \( M^\Pi \). Basically, two paths \( \pi_1 \) and \( \pi_2 \) are stutter-equivalent, denoted by \( \pi_1 \cong \pi_2 \), if there exists a finite sequence \( A_0 \ldots A_n \in (\mathcal{P}(AP))^+ \) such that \( \text{trace}(\pi_1), \text{trace}(\pi) \in A_0^+ A_1^+ \ldots A_n^+ \), where \( A^+ = \{A, AA, \ldots\} \) is the set of finite, non-empty repetitions.

**Theorem 3** (Stutter-Equivalence). Let \( M, A, \) and \( \Pi \in \mathcal{P} \) be such that \( M, \Pi \models A \).

For every \( \pi \in FPath_{M^\Pi} \) there exists \( \hat{\pi} \in FPath_{M^\sigma_A} \) such that \( \pi \cong \hat{\pi} \) and \( \Pr_{M^\Pi}(\pi) = \Pr_{M^\sigma_A}(\hat{\pi}) \). For every \( \hat{\pi} \in FPath_{M^\sigma_A} \), where last(\( \hat{\pi} \)) \( \in V_{II} \), there exists \( \hat{\pi} \in FPath_{M^\Pi} \) such that \( \hat{\pi} \cong \pi \) and \( \Pr_{M^\sigma_A}(\hat{\pi}) = \Pr_{M^\Pi}(\pi) \).

**Proof Sketch.** We show that for every execution fragment \( \varrho_1 = \langle s, q \rangle \xrightarrow{a,p} \langle s, q' \rangle \) there exists \( \hat{\varrho}_1 = \langle s, q, II \rangle \xrightarrow{\tau} \langle s, q, I \rangle \xrightarrow{a,p'} \langle s', q, II \rangle \). Moreover, for every \( \varrho_2 = \langle s, q \rangle \xrightarrow{w} \langle s', q \rangle \) there exists \( \hat{\varrho}_2 = \langle s, q, II \rangle \xrightarrow{w} \langle s, q', II \rangle \). Using induction, we show that for every
arbitrary execution $\varrho$ there exists $\hat{\varrho}$ such that $\varrho \triangleq \hat{\varrho}$, where

$$\text{trace}(\varrho) = (A_0 + A_0 A_0) \ (A_1 + A_1 A_1) \ \ldots \ (A_n + A_n A_n) \in (\mathcal{P}(\mathcal{AP}))^+$$

and Pr($\varrho$) = Pr($\hat{\varrho}$). Similarly, the other direction can be shown for every last($\hat{\varrho}$) that ends with last($\hat{\varrho}$) $\in V_{II}$.

We use Theorem 3 to devise the protocol synthesis procedure summarized in Algorithm 4. In the first part (lines 1–8), the procedure starts by synthesizing a local strategy $\sigma_{(q_0,s_0)}$ and obtaining the associated LFP vector $x_{q_0} \in [0,1]^R$. Next, $R$ is partitioned using $x_{q_0}$ to obtain $(R_{q_0}^q)_{q \in Q}$. If $R_{q_0}^q \neq \emptyset$ for some $q \neq q_0$, the same procedure is repeated for $q$ to obtain $(q,s_0), x_q$ and $(R_{q'}^q)_{q' \in Q}$. In the second part (lines 9–16), three modules are constructed based on Definition 22. The resulting parallel composition constitutes $\mathcal{M}_q^\sigma$, which mimics a stochastic 2-player game between $\hat{\mathcal{M}}$ (player I) and $\hat{\mathcal{A}}$ (player II), where the players’ choices are already resolved by $\hat{\sigma}$. Finally, $\Pi$ is populated by a query that checks for the CAPTL satisfaction condition (line 17): i.e., a state $\langle s, q_i, \gamma \rangle$ is reached, where $q_i = \mathbb{P}_{\max}[\Diamond \Box B^i]$ is active, and $\Box B^i$ holds. Notice that, based on the results from Lemma 2, Algorithm 4 synthesizes a local strategy at most once for every $q \in Q$, compared to Algorithm 3 where synthesis is performed at every reachable state.

**Theorem 4.** Algorithm 4 terminates; and returns $\Pi, c$ iff $\mathcal{M}, \Pi \models c \mathcal{A}$.

**Example 7 (Protocol Synthesis).** For the CAPTL requirement described earlier in Example 5 (see Fig. 3.3), Fig. 3.5 shows a visual representation of the protocol synthesized using Algorithm 4, where blue markers indicate actions in Act, and red markers indicate actions in $W$. While pursuing $q_0$, the robot can achieve the task by moving $a_N(\triangle)$, $a_N(\triangledown)$, $a_E(\triangleright)$ if no obstacles are encountered, or if obstacles are encountered only once while moving $a_E(\triangleright)$. Switching from $q_0$ to $q_1$ via $w_{01}(\triangle)$ occurs in one state
**Algorithm 4:** Synthesis Procedure for P-CAPTL

Input: \( \mathcal{M} = (S, \text{Act}, P, s_0, AP, L) \), \( \mathcal{A} = (Q, W, \Xi, \rightarrow, q_0) \)

Result: \( \Pi, c \) such that \( \mathcal{M}, \Pi \models_c \mathcal{A} \)

1. \textbf{foreach} \( (q, q') \in Q \times Q \) \textbf{do} \( R_q^2 \leftarrow \emptyset \) // Initialize
2. \( \Pi \leftarrow \emptyset, \hat{Q} \leftarrow \{q_0\}, \hat{Q} \leftarrow \emptyset, R \leftarrow \text{REACH}(\mathcal{M}, s_0) \)
3. \textbf{while} \( \hat{Q} \neq \emptyset \) \textbf{do} // Partition \( R \)
4. \hspace{1em} Let \( q \in \hat{Q}, \hat{Q} \leftarrow \hat{Q} \setminus \{q\}, \hat{Q} \leftarrow \hat{Q} \cup \{q\}, R_q^2 \leftarrow R \)
5. \hspace{1em} \( \sigma_{(q, s_0)} \leftarrow \text{SYNTH} (\mathcal{M}; s_0, q), x_q \leftarrow \text{REACHP} (\mathcal{M}, s_0, \sigma_{(q, s_0)}) \)
6. \hspace{1em} \textbf{foreach} \( w \in W(q) \text{ where } q' = Q(q, w) \) \textbf{do}
7. \hspace{2em} \( R_q^2 \leftarrow \{s \mid x_q[s] \in J_w\}, R_q^2 \leftarrow R_q^2 \setminus R_q^2 \)
8. \hspace{2em} \textbf{if} \( R_q^2 \neq \emptyset \land q' \notin \hat{Q} \text{ then } \hat{Q} \leftarrow \hat{Q} \cup \{q'\} \)
9. \hspace{1em} \textbf{construct} \( \hat{\mathcal{M}} \) \textbf{module such that} // Construct \( \hat{\mathcal{M}}^p_A \)
10. \hspace{2em} \textbf{foreach} \( [a] s \rightarrow p_i : (s_i') \) \textbf{do} \add\([a] s \land I \rightarrow p_i : (s_i') \land (I) \)
11. \hspace{1em} \textbf{construct} \( \hat{\mathcal{A}} \) \textbf{module such that}
12. \hspace{2em} \textbf{foreach} \( q \in \hat{Q} \) \textbf{do} \add\([\tau] q_{\text{act}} = q \land \Pi \land L(R_q^2; s) \rightarrow (q_{\text{act}} = q) \land (I) \)
13. \hspace{2em} \textbf{foreach} \( q \xrightarrow{w} q' \) \textbf{do} \add\([w] q_{\text{act}} = q \land \Pi \land L(R_q^2; s) \rightarrow (q_{\text{act}} = q') \land (II) \)
14. \hspace{1em} \textbf{construct} \( \hat{\sigma} \) \textbf{module such that}
15. \hspace{2em} \textbf{foreach} \( \sigma_{(q, s_0)} \neq \emptyset \) and \( s \in R \) \textbf{do} \add\([\sigma_{(q, s_0)}(s)] q_{\text{act}} = q \land s \rightarrow \top \)
16. \hspace{1em} \( \hat{\mathcal{M}}^p_A \leftarrow \hat{\mathcal{M}} \parallel \hat{\mathcal{A}} \parallel \hat{\sigma} \)
17. \( (\Pi, c) \leftarrow \text{SYNTH} (\hat{\mathcal{M}}^p_A, q_0, s_0, \Pi), \mathbb{P} = \bigvee_{q_i \in Q} \Diamond (q_{\text{act}} = q_i) \land B_i \)

**Figure 3.5:** The protocol synthesized based on the CAPTL requirement in Fig. 3.3, where \( R_V = \text{reach}(\mathcal{M}^p_A, v_0) \).
while switching from \( q_0 \) to \( q_2 \) via \( w_{02} \) occurs in four states \((0, 8, 1, 1), (0, 4, 3, 1), (2, 7, 2, 3) \) and \((0, 4, 1, 3)\).

### 3.5 Experimental Evaluation

We demonstrate the use of CAPTL for protocol synthesis and analysis on two case studies. The first extends the robot task planning problem introduced in Sec. 3.2. The second considers the problem of synthesizing an error-resilient scheduler for digital microfluidic biochips. To this end, we implemented Algorithm 4 in MATLAB on top of a modified version of PRISM-games \([\text{KPW18}]\) (v4.4), where \textsc{ReachP} functionality was added. The experiments presented in this section were run on an Intel Core i7 2.6GHz CPU with 16GB RAM.

#### 3.5.1 Robotic Task Planner

Table 3.1 summarizes the performance results for running Algorithm 4 on various sizes of the running example. Notice that the number of choices in \( \mathcal{M}_A^\sigma \) always matches the number of states, which agrees with the results from Lemma 4. In the three models, \( q_0 \) is always active in \( s_0 \), and thus is always verified. As the grid size grows larger, the probability of reaching the goal — and hence satisfying \( q_0 \) — becomes lower, dropping below 0.85 at the initial state in both \((6 \times 6)\) and \((9 \times 9)\). As a result, \( q_1 \) is never active (and hence is never verified) in the second and third models. We also notice that the total time required to run Algorithm 4 does not necessarily grow as the size of the problem grows. In fact, the total time required for \((6 \times 6)\) and \((9 \times 9)\) is lower than the one for \((3 \times 3)\). This is primarily due to the fact that \( q_1 \) is never reached or verified in the second and third models as we described. When comparing the model size for \( \mathcal{M} \) and \( \mathcal{M}_A^\sigma \), we notice that \( |\mathcal{M}_A^\sigma| < |\mathcal{M}| \), with the difference being in orders of magnitude for larger models. However, the time required to construct
$M^*_A$ is longer than the time required to construct $M$.

### 3.5.2 MEDA-Biochip Scheduler

We now consider synthesizing error-resilient scheduler for micro-electrode-dot-array (MEDA) digital microfluidic biochips, where we borrow examples from [EZZ+17a, LLY+16a]. A biochip segment consists of a $W \times H$ matrix of on-chip actuators and sensors to manipulate microfluidic droplets, and is further partitioned into $3 \times 3$ blocks. Two reservoirs are used to dispense droplets $A$ and $B$. Various activation patterns can be applied to manipulate the droplets, including moving (moving droplets individually), flushing (moving both droplets at the same time in the same direction) and mixing (merging two droplets occupying the same block). As the biochip degrades, the actuators become less reliable, and an actuation command may not result in the droplet moving as expected. The probability of an error occurring is proportional to the total number of errors occurred in the same block.

Fig. 3.6 shows part of the segment scheduler (left) and the droplet (right) models. Initially, the scheduler can dispense both droplets through the dispense action, where the droplet location $(x, y)$ can probabilistically deviate from the dispenser location $(x_0, y_0)$ with error $\epsilon$. Subsequently, droplets can be individually manipulated via $\text{mvA}[d]$ and $\text{mvB}[d]$ actions where $d$ is the direction, or together via $\text{flush}$. The probability of successful manipulation $(1 - p(\epsilon))$ depends on both the number of errors within the same block ($\epsilon_{\ell}$) and the activation pattern used. The scheduler executes update to sense droplet locations and register errors.

The primary task of the scheduler is to perform a mixing operation within the given segment ($q_0$). However, if the droplets are dispensed and (due to faulty blocks) the probability of a successful mixing operation is below $0.85$ ($w_{01}$), salvaging the dispensed droplets by moving them to an adjacent segment is prioritized ($q_1$). If
the mixing probability drops below 0.7 \( (w_{02}) \), or if the salvaging probability drops below 0.7 \( (w_{12}) \), the scheduler is to abort the operation \( (q_2) \). The aforementioned requirements are formalized using CAPTL as shown in Fig. 3.7. The set of objectives is \( Q = \{ q_0, q_1, q_2 \} \), and the set of contexts is defined as \( W = \{ w_{01}, w_{02}, w_{12} \} \). The performance results for running Algorithm 4 on three different segment sizes is reported in Table 3.1.

![Figure 3.6: The MEDA biochip scheduler model (left) and the droplet model (right).](image)

![Figure 3.7: P-CAPTL requirement for a MEDA-biochip segment scheduler.](image)

Table 3.1: Protocol synthesis performance results for the robotic task planner (C1) and the MEDA-biochip scheduler (C2). (St.: states, Tr.: transitions, Ch.: choices).

<table>
<thead>
<tr>
<th>Model</th>
<th>( \mathcal{M} ) Size</th>
<th>( \mathcal{M}^\sigma_A ) Size</th>
<th>Construction/Synthesis Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>St.</td>
<td>Tr.</td>
<td>Ch.</td>
</tr>
<tr>
<td>C1 3×3</td>
<td>233</td>
<td>1,117</td>
<td>745</td>
</tr>
<tr>
<td>6×6</td>
<td>595</td>
<td>2,692</td>
<td>1,874</td>
</tr>
<tr>
<td>9×9</td>
<td>733</td>
<td>3,242</td>
<td>2,278</td>
</tr>
<tr>
<td>C2 8×5</td>
<td>2,851</td>
<td>8,269</td>
<td>5,678</td>
</tr>
<tr>
<td>11×5</td>
<td>8,498</td>
<td>25,502</td>
<td>17,214</td>
</tr>
<tr>
<td>14×8</td>
<td>61,489</td>
<td>201,469</td>
<td>130,718</td>
</tr>
</tbody>
</table>
3.6 Conclusion

In this chapter we have introduced context-aware probabilistic temporal logic (CAPTL). The logic provides intuitive means to formalize requirements that comprises a number of objectives with an underlying priority structure. CAPTL allows for defining context (i.e., probabilistic conditions) as the basis for switching between two different objectives. We have presented CAPTL syntax and semantics for MDPs. We have also investigated the CAPTL synthesis problem, both from PCTL and CAPTL-based approaches, where we have shown that the latter provides significant performance improvements. To demonstrate our work, we have presented two case studies. As this work has primarily considered CAPTL semantics for MDPs, further investigation is required to generalize the results for stochastic multi-player games. Another research direction involves expanding the results to include PCTL fragments beyond persistence objectives, such as safety, bounded reachability and reward-based objectives.
Chapter 4

Security-Aware Synthesis of Human-UAV Protocols

This chapter is an adapted reproduction of “Mahmoud Elfar, Haibei Zhu, Mary L. Cummings, and Miroslav Pajic. Security-aware synthesis of human-UAV protocols. In 2019 International Conference on Robotics and Automation (ICRA), pages 8011–8017, IEEE, 2019” [EZCP19b]; and hence is not available under a Creative Commons license.

4.1 Introduction

Contrary to what the terminology may suggest, autonomous systems mostly involve human presence; from actively engaging with the system to merely monitoring the system status or intervening whenever necessary [CBMM07]. A typical example is the human-unmanned aerial vehicle (H-UAV) command and control system, where various deployed applications depend on having human operators responsible for supervising a fleet of UAVs during a mission. The operator performs various supervisory tasks including, for example, updating mission goals, monitoring agent status, and adjusting flight plans [NCC07]. The operator can also be assigned primary tasks that are mission relevant, such as imagery tasks.

With the human presence, considering human factors becomes an essential part of the modeling and design of those systems, for which several attempts have been introduced in literature. The reliance on experimental data has been proposed to model human-autonomy interactions in various applications such as autonomous cars [SDCP+14, LSSSI14], industrial [PDSI15, ULT+18] and social robotics [ASI14].
Due to their ability to model reactive systems, Markovian formalisms, e.g., Markov Decisions Processes (MDPs) and stochastic games, were exploited for the theoretical exploration of human-robot interactions [FWHT16].

On the other hand, UAV navigational systems have been recently proven to be vulnerable to cyber and physical attacks, such as false-data injection attacks that target GPS receivers [HLP+08, KSBH14], raising security concerns in this domain. A number of studies have focused on attack detection via sensor redundancy (e.g., [PLP17, PWB+17, IPL16, PIW+17, PDB13, SNP+17, MCS14]). Yet a class of these attacks can remain stealthy by introducing non-aggressive and incremental deviations [KLHT14, MST10, MCS14, JPT19]. Although humans are likely to surpass autonomy in such situations of high uncertainty [Cum14], no research has addressed the human role in ensuring the security of H-UAV systems; such as, whether we can improve the overall security guarantees by harnessing the human power of inductive reasoning and the ability to provide context and additional information to the system in real-time.

Hence, in this chapter, we focus on synthesis of protocols for H-UAV systems where the operator can intermittently perform geolocation tasks to aid in detection of possible attacks. First, the system dynamics, operator geolocation task, and the adversarial behavior are modeled using stochastic games. By developing RESCHUSA testbed, experiments were conducted to understand operator strategies during geolocation tasks. Next, we use machine learning to predict correctness of a geolocation task at a given location. Note that in security problems, the system (i.e., UAV) is not aware of the information related to attacker’s actions, which presents a significant synthesis challenge. Thus, we construct the model as a delayed-action game, which allows for the use of off-the-shelf tools (PRISM-games) to synthesize security-aware H-UAV protocols; such protocols provide UAV path plans that increase chances of
attack detection. Moreover, the protocols specify time instances at which the operator is advised to perform a geolocation task, maximizing its correctness. The formal synthesis of the advisory system guarantees a limit to the workload level, in order to avoid performance deterioration without compromising system security. Finally, we present a case study where the synthesized protocols are analyzed and subjectively evaluated by human operators.

The rest of this chapter is organized as follows. Sec. 4.2 provides a background on stochastic games and H-UAV control systems before formulating the problem statement. Sec. 4.3 presents the system modeling using stochastic and delayed-action games, while Sec. 4.4 describes the experiments used to obtain model parameters. The protocol synthesis framework is provided in Sec. 4.5. Sec. 4.6 presents a case study where the synthesized protocols are analyzed and evaluated. Finally, Sec. 4.7 concludes the chapter and provides avenues for future work.

4.2 Background and Problem Statement

We start with the related background on stochastic games and strategy synthesis, followed by the problem statement.

4.2.1 Stochastic Games

Stochastic multiplayer games (SMGs) can model reactive systems with both stochastic and nondeterministic transitions, where the latter are resolved by more than one player. Stochasticity arises when the system evolution cannot be precisely predicted, yet a probabilistic profile can be assumed. Conversely, nondeterminism abstracts players’ decisions, either to incorporate a family of behaviors or to reason about a winning strategy. A turn-based game is played such that, at each state, only one player at most can make decisions. Turn-based SMGs have proven to be useful for
modeling reactive systems [CFK+13a].

Definition 23 (Turn-Based SMGs). A turn-based SMG over players \( \Gamma = \{I, \Pi, \bigcirc\} \) is a tuple \( \mathcal{G} = (S, (S_I, S_{II}, S_{\bigcirc}), A, \varsigma, \delta) \), where \( S \) is a finite set of states, partitioned into players’ and stochastic states \( S_I, S_{II} \) and \( S_{\bigcirc} \); \( A \) is a finite set of actions; \( \varsigma \in \text{Dist}(S_{II}) \) is an initial distribution over \( S_{II} \); \( \delta : S \times S \rightarrow [0, 1] \) is a transition function s.t. \( \delta(s, s') \in \{0, 1\}, \ \forall s \in S_I \cup S_{II} \) and \( s' \in S \), and \( \delta(s, s') \in [0, 1], \ \forall s \in S_{\bigcirc} \) and \( s' \in S_I \cup S_{II} \), where \( \sum_{s'\in S} \delta(s, s') = 1 \) holds.

In contrast to SMGs where the game state is visible to all players, in [EWP19b] we introduced delayed-action games (DAGs) that partially obscure the game state from one player by substituting the hidden information (i.e., the truth) regarding some states with the player’s belief for these hidden states.

Definition 24 (Delayed Action Game). A delayed-action game (DAG) is a tuple \( \hat{\mathcal{G}} = (\mathcal{G}, V, \Gamma) \) where \( \mathcal{G} \) is a turn-based SMG over a set of game variables \( V = V_T \cup V_B \) and players \( \Gamma = \{I, \Pi, \bigcirc\} \), such that \( S \subseteq \text{Ev}(V) \times \mathcal{P}\text{Ev}(V_T) \times \Gamma \). In addition, \( V_T \) is the set of variables holding the true values of the game, known to player I; and \( V_B \) is the set of variables holding player II beliefs about the values of the game.

In Definition 24, for any variable \( \text{var} \) from a set \( V \), \( \text{Ev}(\text{var}) \) denotes the set of evaluations that assign values to \( \text{var} \). Also, for a set \( A \), \( \mathcal{P}A \) refers to the power set \( 2^A \).

Intuitively, a player’s strategy is how she resolves nondeterminism throughout the game, while a protocol is a set of strategies adopted by a coalition of players. The synthesis problem seeks a winning strategy (protocol); that is, it resolves choices for a player (coalition) such that some objectives are satisfied. Synthesis objectives can be specified using temporal logic such as ATL and rPATL [CFK+13b].
4.2.2 Human-UAV Supervisory Systems

This work is motivated by H-UAV supervisory systems, where the UAV is supervised by a human operator [CBMM07]. A typical mission is for the UAVs to reach a number of targets to perform imagery tasks. While an autonomous planner automatically assigns UAVs to target locations, the operator can override these assignments or the path plan if necessary. Once a target is reached, the operator is notified to assist with the imagery task by analyzing the live camera feed [NCC07].

UAVs are prone to adversarial attacks, such as GPS spoofing, that can drive the UAV away from its planned path [HLP+08, KSBH14, KYH15]. Several techniques have been proposed to aid with detecting such attack by relying on redundant sensors (e.g., [PLP17, PWB+17, IPL16, PIW+17, PDB13, SNP+17, MCS14]). Nevertheless, when a sufficient number of sensors is compromised, a smart attacker can remain stealthy by injecting non-aggressive and incremental deviations in sensor measurements, which will still force the UAV into any undesired state through the actions of the controller [KLH14, MS10, JP19].

On the other hand, a geolocation task ultimately aims at localizing the UAV through side channel information. For example, we showed that landmarks and other geo-features from a UAV’s camera feed can be used to estimate the UAV’s location in real time by a human-operator [ZEP+18]. Thus, in this work, we address the problem of synthesizing protocols for the Human-UAV supervisory system that also employs the human operator for attack detection. The synthesized protocols shall provide the UAV path plan, as well as the time instances at which the operator shall be advised to perform a geolocation task, while ensuring that a given set of performance objectives (e.g., time, safety, workload) is satisfied.
4.3 System Modeling

In this section, we describe a system model, before showing how it can form a DAG used for the protocol synthesis.

4.3.1 SMG-Based Model

Due to the stealthy nature of cyber-attacks considered in this study, one should differentiate between the UAV belief about its location, which may not be accurate, and the ground truth, which is assumed to be known to the adversary. Fig. 4.1(a) shows a standard UAV model \( M_{\text{uav}} \) (e.g., as in [TBF05]) where the UAV movement is discretized into action set \( A_{\text{uav}} = \{N, S, E, W, NE, NW, SE, SW\} \). Note that the UAV’s actions affect its belief \( x_B \). Conversely, \( M_{\text{adv}} \) shows how the adversary’s stealthy actions are constrained by the UAV movements, influencing the ground truth \( x_T \) to avoid detection, only slowly increasing deviations from the planned trajectory can be achieved by the attacker; e.g., through GPS spoofing and actions of the low-level controller, small errors are added to the desired commands in each step. If, for example, the UAV is heading N, then the adversary available actions are \( A_{\text{adv}}(N) = \{NE, N, NW\} \). Note that more aggressive attacks can be detected as in e.g., [PLP17, PWB+17].

As in Fig. 4.1(b), the human geolocation model \( M_{\text{hgl}} \) can initiate a geolocation task via the action \( \text{start} \). The outcome of the task, however, can be successful or not with probabilities \( p(x_T, x_B) \) and \( 1 - p(x_T, x_B) \), respectively, depending primarily on both \( x_B \) and \( x_T \) — detailed probability modeling is provided in Section 4.4. While the presented model is a standard SMG, since \( x_T \) is unknown to the UAV — and thus so is \( p(x_T, x_B) \) — the model cannot be used to reason about strategies.

\footnote{Other factors affecting \( p \), such as operator skills and the current workload, are not considered in this study. Also, the model assumes that task repetition has no impact on \( p \). See the discussion in Sec. 4.7.}
Moreover, if SMG semantics are used (i.e., the truth is implicitly known to the UAV), a synthesized strategy becomes a function of the adversarial specific actions — which are unknown — thus, rendering the strategy useless.

### 4.3.2 DAG-Based Model

To overcome these challenges, a DAG variation of the SMG model can be used (Fig. 4.2). The basic idea is that, during a mission, the UAV is unaware of differences between its belief and the ground truth until a geolocation task is correctly done. Thus, although we model missions as the players taking turns (SMG model), the same behavior can be captured if the UAV makes its moves, updating only the belief state, until a geolocation occurs; followed by the adversary’s corresponding actions, updating the ground truth. With such a DAG model, the UAV takes a number of actions ahead of the adversary; that is, the adversary’s actions are hidden from the UAV as they have not occurred yet in the model. Since the UAV decisions are made without knowing the adversary’s specific actions, a UAV synthesized strategy becomes independent of those actions. Hence, the DAG model provides a different representation of the system without altering its behavior, as we show in [EWP19b],

---

**Figure 4.1**: SMG-based model components: (a) UAV ($\mathcal{M}_{uav}$) and adversary ($\mathcal{M}_{adv}$), and (b) human geolocation ($\mathcal{M}_{hgl}$).
and thus can be used to synthesize strategies for the original SMG model. This mechanism is realized by the auxiliary components $M_{mwr}$ and $M_{mrd}$ that model write and read operations, respectively, on a FIFO memory stack $(m_i)_{i=0}^{n-1} \in A_{adv}^n$ as shown in Fig. 4.2(c). By synchronizing with $write$, $M_{mwr}$ saves the last UAV action in memory location $m_i$, executing $s_i \xrightarrow{\text{write}} s_{i+1}, i \in \{0, ..., n-1\}$. Similarly, $M_{mrd}$ synchronizes with the action $read$ to read the saved UAV action from memory location $m_j$, executing $s_j \xrightarrow{\text{read}} s_{j+1}, j \in \{0, ..., n-1\}$.

The DAG is formed by composing both the model and auxiliary components using the procedure summarized in Algorithm 5. The composed DAG allows the UAV ($M_{uav}$) to play a finite number of actions before attempting a geolocation task, updating its belief $x_B$. Based on these actions, the adversary ($M_{adv}$) plays her delayed actions, updating the ground truth $x_T$. Hence, the probability of performing

```

Figure 4.2: DAG-based model with the UAV ($M_{uav}$); adversary ($M_{adv}$); human geolocation ($M_{hgl}$); and DAG memory read ($M_{mrd}$), write ($M_{mwr}$).

(a) \( pl = uav \)
   \( \text{start!} \)
   \( pl := adv \)

\( f(y_{d_B}), d_B \in A_{uav} \)
\( x_B := x_B + \Delta(d_B) \)

\( M_{uav} \)
\( pl = uav \)
\( write! \)

(b) \( pl = as \)
   \( \text{update!} \)
   \( pl := uav \)

\( x_g := x_r \)
\( p(x_T, x_B) \)

\( M_{hgl} \)
\( pl = hgl \)
\( fail \)
\( pl := uav \)

(c) \( pl = adv \)
   \( \text{check!} \)
   \( pl := hgl \)

\( (p_l = adv) \land (s_{mrd} = s_{mwr}) \)
\( s_{mrd} < s_{mwr} \)

\( M_{adv} \)
\( pl = adv \)
\( x_f := x_f + \Delta(d_f), d_f \in A_{adv}(d_B) \)

\( M_{mrd} \)
\( \text{update?} \)
\( m_i := 0 \)

\( M_{mwr} \)
\( \text{update?} \)
\( m_{i-1} := d_B \)

\( m_{i} := d_B \)

Legend
- guard
- channel transmit (!)
- channel receive (?)
- assignment
- state
- transition

```
Algorithm 5: DAG construction procedure

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>while end criterion not met do</td>
</tr>
<tr>
<td>2</td>
<td>while $a_{uav} \neq$ geolocation task do</td>
</tr>
<tr>
<td>3</td>
<td>$M_{uav}.x_B \leftarrow \text{Eff}(a_{uav}, x_B)$ UAV moves, updating belief</td>
</tr>
<tr>
<td>4</td>
<td>$M_{mwr}.write(a_{uav}, ++wr)$ write action to memory</td>
</tr>
<tr>
<td>5</td>
<td>while $wr \leq rd$ do</td>
</tr>
<tr>
<td>6</td>
<td>$M_{adv}.read(a_{uav}, ++rd)$ read UAV action from memory</td>
</tr>
<tr>
<td>7</td>
<td>$M_{adv}.x_T \leftarrow \text{Eff}(\beta(a_{uav}), x_T)$</td>
</tr>
<tr>
<td>8</td>
<td>if draw $x \sim \text{Brn}(f(x_T, x_B))$ then</td>
</tr>
<tr>
<td>9</td>
<td>$M_{uav}.x_B \leftarrow M_{adv}.x_T$ update belief to match truth</td>
</tr>
<tr>
<td>10</td>
<td>reset($wr, rd$) reset memory</td>
</tr>
<tr>
<td>11</td>
<td>else reset(rd) forget (hide) ADV actions</td>
</tr>
</tbody>
</table>

A correct geolocation task is given by some function $f(x_T, x_B)$ — the realization of the function $f(x_T, x_B)$ is discussed in Sec. 4.4. If the geolocation task is unsuccessful, the adversarial actions are discarded, and the UAV can continue to move or attempt another task. Otherwise, the UAV belief is updated to match the ground truth, at which point the game is repeated with a new starting location. In the rest of the chapter, we will refer to each of these repetitions as a subgame $\hat{G}_i$, where $i$ is the subgame index, and to the set of all subgames of interest as the supergame $\hat{G}$. Intuitively, each subgame explores the possible adversarial effect for each UAV sequence of actions, while the supergame examines the collective behavior of those subgames.

### 4.4 Human Geolocation Model

As the presented DAG-based model is characterized by the probabilities $f(x_T, x_B)$, in this section we present the experimental platform and evaluations used to obtain this function.
4.4.1 Experimental Platform and Design

The aim of this experiment was twofold. First, we wanted to validate the hypothesis that UAV operators can successfully perform geolocation tasks. Second, we wanted to understand what strategies the operators adopt to perform the geolocation tasks and their relevant factors.

To test our hypothesis, we developed Security-Aware extension for RESCHU (RESCHU-SA)\(^2\) a virtual platform for studying the security aspects of human-UAV supervisory systems prone to cyber-attacks. Fig. 4.3 shows RESCHU-SA operator interface (OI) with the map area displaying flight plans, target locations, and threat zones. While the map shows a UAV’s reported location, which may be compromised, the camera feed displays the live video stream from the selected UAV, from the UAV’s true location. Through the OI, the operator can supervise a fleet of UAVs on a timed mission where a number of target locations should be visited by the UAVs, and a visual task should be completed by the operator through the UAV camera feed once each location is reached.

Each experiment consisted of two missions with high and low workloads. During each mission, a number of geolocation requests are randomly introduced, where the operator can respond by activating the camera feed to perform a geolocation task and further report whether the reported location is correct.\(^3\) After the two missions are done, the operator is interviewed and a retrospective verbal protocol analysis is performed to elicit more insights on their behavior and decisions made during the experiment.

\(^2\)Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles (RESCHU) was originally developed to evaluate Human-UV supervisory systems [Neh09, DNC10]. The initial platform does not support attack modeling or real-time video feed from any location from a map, based on UAVs’ trajectories that the user can change during experiments.

\(^3\)Video: [https://cpsl.pratt.duke.edu/research/security-aware-human-loop-cps](https://cpsl.pratt.duke.edu/research/security-aware-human-loop-cps)
Figure 4.3: RESCHU-SA operator interface (OI) elements. A UAV placement on the map is based on its potentially compromised GPS reported location.

4.4.2 Human Geolocation Strategies

We performed experiments with 36 participants and a total of 641 geolocation attempts; the results of this experiment are summarized in Fig. 4.4 – due to space limitations, the full results can be found in [ZCE+18]. Participants were found to adopt three main strategies for geolocationing: comparing road patterns (59.3%), observing terrains (23.7%), and examining landmarks (17.0%). Note that the road pattern-based strategy was used more than half of the time, with the lowest error rate (15.0%) compared to the strategies based on terrains (20.4%) and landmarks (19.3%).

Since the results suggest that the operator’s ability to compare the map with the
Figure 4.4: Human geolocation strategies observed during the experiments. A true positive (TP) refers to a geolocation task confirming discrepancies while a GPS spoofing attack is active.

Table 4.1: List of the GIS layers and their semantics.

<table>
<thead>
<tr>
<th>GIS Layer</th>
<th>Labels Set</th>
<th>Labeling Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terrains</td>
<td>$\Sigma_{tr} = {rural, urban, water,\ldots}$</td>
<td>$L_{tr}: Ev(x) \to \mathcal{P}\Sigma_{tr}$</td>
</tr>
<tr>
<td>Roads</td>
<td>$\Sigma_{rd} = {rect, star, dense,\ldots}$</td>
<td>$L_{rd}: Ev(x) \to \mathcal{P}\Sigma_{rd}$</td>
</tr>
<tr>
<td>Landmarks</td>
<td>$\Sigma_{lm} = {building, tower, block,\ldots}$</td>
<td>$L_{lm}: Ev(x) \to \mathcal{P}\Sigma_{lm}$</td>
</tr>
</tbody>
</table>

Figure 4.5: An example of map segments and their associated GIS labels.

camera feed depends heavily on the discrepancies between the geographic features of both scenes, capturing the map’s geographic information system (GIS) in the model becomes critical to the synthesis problem. To this end, we represent a GIS layer as a labeling function that maps a location on the grid to a set of labels (i.e., atomic propositions) capturing the features of interest for that location. Table 4.1 lists the GIS layers relevant to the geolocation strategies, as obtained from our experiments, while Fig. 4.5 presents a 4-map example segments and their associated labels.
4.4.3 Human Geolocation Task (GT) Predictor

The abstract goal of a geolocation task is to determine whether the UAV’s reported belief $x_B = x_{\text{map}}$ (see Fig. 4.1(b)) matches the ground truth $x_T = x_{\text{cam}}$ observed via the camera feed. The operator’s ability to correctly perform the task is affected by the GIS features of both $x_{\text{cam}}$ and $x_{\text{map}}$. For instance, if $x_{\text{cam}} = x_{\text{map}}$ (i.e., they refer to the same location), the presence of distinctive features is likely to increase the chances of a correct decision. Conversely, if $x_{\text{cam}} \neq x_{\text{map}}$, then the probability of a correct decision relies heavily on the relative distinctiveness between both locations. Therefore, we define a GT predictor as follows.

**Definition 25.** GT predictor is a tuple $\mathcal{K} = (\Sigma, \mathcal{H}, f)$ where $\Sigma$ is the set of GIS layers, $\mathcal{H} = \{r_i : X^2 \to \mathbb{R}, i = 1, ..., n\}$ is a set of $n$ numerical measures of similarity between two locations in $X$, and $f : \mathcal{P}\Sigma^2 \times \mathbb{R}^n \to [0, 1]$ is a prediction function of the GT correctness.

Given two sets of labels and a tuple of numeric measures of similarity between two given locations, the function $f$ predicts the correctness of a GT. In this case-study, we used $\Sigma = \bigcup_{j \in \{tr, rd, lm\}} \Sigma_j$. As a similarity measure, we used the maximum normalized 2D cross-correlation coefficient $r_{rd} \in [0, 1]$ between the road patterns of two locations; that is, $\mathcal{H} = \{r_{rd}\}$.

We derive the prediction function $f$ (and hence $\mathcal{K}$) using machine learning-based techniques. Basically, a predictive model is trained on a database of image pairs, where the training inputs are the GIS labels and similarity measures of each pair; the training output is the corresponding human estimation of the similarity of each pair; and the model outcome is $\hat{f} : \mathcal{P}\Sigma^2 \times \mathbb{R}^n \to [0, 1]$. A survey was administrated to human participants to collect their opinion of how similar two locations are. During

\[\text{In this work, as we focus on attack-detection, we only address the determination of whether two locations are the same, rather than how the actual coordinates can be found.}\]
the survey, each participant was shown a total of 100 pairs of locations. For each pair, the participant was asked to select their estimation on a scale from 1 (very similar) to 5 (very dissimilar), as shown in Fig. 4.6. Next, labels from Σ were manually assigned to the selected locations. The function \( \hat{f} \) was realized using a bagged-trees ensemble learner (RMSE = 0.635, RS = 0.650).

Finally, for two locations \( x_{\text{cam}} \) and \( x_{\text{map}} \) we have that\(^5\)

\[
f(x_{\text{cam}}, x_{\text{map}}) = \begin{cases} 
\hat{f}(x_{\text{cam}}, x_{\text{map}}) & x_{\text{cam}} = x_{\text{map}} \\
1 - \hat{f}(x_{\text{cam}}, x_{\text{map}}) & \text{otherwise.}
\end{cases}
\]

That is, the measure of correctness is the human tendency to consider two images similar or dissimilar if they represent the same or different locations, respectively.

\(^5\)For simplicity, we employ \( f(x_{\text{cam}}, x_{\text{map}}) \) to refer to \( f(L(x_{\text{cam}}), L(x_{\text{map}}), \mathcal{H}(x_{\text{cam}}, x_{\text{map}})) \), where \( L(x) \) is the suitable layer-labeling function, capturing the GIS layers at location \( x \) (as in Table 4.1).
4.5 Protocol Synthesis

4.5.1 Synthesis Objectives

The primary synthesis objective is to find protocols for the H-UAV coalition based on the following requirements.

(a) *Reach the target location.* As one subgame may not yield a feasible flight plan to directly reach the target, checkpoints can be set as intermediate targets to render the objective feasible. By assigning the label `reach` to the set of states with acceptable checkpoint locations, the objective can then be formalized as $\Pr_{\text{max}}[\mathcal{F}\ reach] \geq p_{\text{min}}$ for some bound $p_{\text{min}}$, and has to hold for all encountered subgames.

(b) *Avoid hazard zones.* To reach the target, the UAV must avoid all known hazard zones, where the UAV is likely to endure damage. If the corresponding states are assigned the label `hazard`, the objective can be formalized as $\Pr_{\text{max}}[\mathcal{G}\ \neg\text{hazard}] \geq p_{\text{min}}$ for some bound $p_{\text{min}}$. We assume here that encountering a hazard zone results in losing the asset, hence the global operator. Consequently, this objective has to hold for all encountered subgames.

These objectives are refined to elicit rPATL queries [CFK+13]. Specifically, for a subgame $\hat{G}_s$, the following query is used

$$\phi_{\text{syn}}(k) := \langle\langle\text{uav}\rangle\rangle\Pr_{\text{max}=?}[\neg\text{hazard} U^{\leq k} (\text{locate} \land \text{reach})],$$

i.e., find a strategy that maximizes the probability of not encountering a hazard until a checkpoint is reached (`reach`) and a geolocation task is successful (`locate`) within a horizon $k$.

The procedure described in Algorithm 6 is then used to solve the synthesis problem. Starting with the initial subgame, the procedure checks for each horizon whether $\phi_{\text{syn}}$ is satisfied (hence a strategy exists) if the geolocation task is performed at the last stage. These checks are terminated when reaching the maximum search horizon
or failing to satisfy the objectives. Next, subgames are pruned to discard strategies that fail to satisfy local bounds. The remaining strategies are used to populate the set of end locations, where each end-location represents an initial location to a reachable subgame that needs to be explored. The procedure is repeated for all subgames with initial state in the end locations set to obtain a strategy $\pi_i$ for each reachable subgame $\hat{G}_i$. Thus, for a $q$ number of reachable subgames, the supergame is reduced to an MDP $\hat{G}^{(\pi_i)}_{i=1}$ (whose states are the reachable subgames); the MDP is then checked against the query

$$\hat{G}^{(\pi_i)}_{i=0}: \phi_{ana}(n) := \langle\langle adv\rangle\rangle Pr_{min=?} [F^\leq n \text{ target}]$$

to find the minimum probability of eventually reaching the target within a maximum number of geolocation tasks $n$.

### 4.5.2 Synthesis Procedure

The protocol synthesis procedure is summarized in Algorithm 6. Starting from the initial location $x_0$ and horizon $h = 1$, the first subgame is constructed and explored. For each horizon, the synthesizer searches for a corresponding optimal strategy and set of reachable locations, until either the maximum search horizon $h_{\text{max}}$ is reached or no feasible strategy can be found. The same process is repeated using unexplored locations in the reachability set. Note here that the set $\Pi$ can be pruned (e.g., by discarding strategies that violates auxiliary requirements) to reduce computation time.

### 4.6 Experimental Results

Fig. 4.7(a) shows the environment setup used for evaluation. The map was discretized into a $10 \times 10$ grid, where crossing the map boundaries is penalized for both $x_B$ and $x_T$.

---

6If no strategy exists for a stage $i$, then the same holds for all stages of size $j > i$.  
74
Algorithm 6: Protocol synthesis procedure

Input: Initial location \( x_0 \), synthesis query \( \phi_{\text{syn}} \), max horizon \( h_{\text{max}} \)

Output: H-UAV protocols \( \Pi = \{ (\pi_{\text{uav}}, \pi_h) \} \)

1. \( X \leftarrow \{ x_0 \} \) initialize set of initial locations (subgames)

2. \textbf{foreach} unexplored initial location \( x_i \in X \) \textbf{do}

3. \( \hat{s}_0 \leftarrow (\text{UAV}, x_i, \epsilon) \) set subgame initial state

4. \( \text{stop} \leftarrow \bot, h \leftarrow 1 \) reset stopping flag and horizon

5. \textbf{while} \( h \leq h_{\text{max}} \land \neg \text{stop} \) \textbf{do}

6. \( (\pi_{\text{uav}}, \varphi) \leftarrow \text{synth} \left( \hat{G}^\pi_{\hat{s}_0}, \phi_{\text{syn}} \right) \) find a winning strategy

7. \textbf{if} \( \pi_{\text{uav}} \) exists \textbf{then}

8. \( \Pi \leftarrow \Pi \cup (\pi_{\text{uav}}, \pi_h, \varphi) \) add to the protocol

9. \( X \leftarrow X \cup \text{reach} (\pi_{\text{uav}}) \) update reachability set

10. \( h \leftarrow h + 1 \) explore next horizon

11. \textbf{else} \( \text{stop} \leftarrow \top \)

12. \textbf{prune} (\Pi)

(a) Mission setup.  
(b) Synthesized protocols.

Figure 4.7: The mission setup used for experimental evaluation and the corresponding protocols. For clarity, the map colors are altered and the protocols are partially displayed.

Also, for the UAV to ever arrive to the designated target, the adversary is prohibited from launching attacks for at least the first horizon. The UAV mission is to reach the target without encountering any of the hazard zones. The model shown in Fig. 4.2 was implemented using the PRISM-games [CFK+13b] model checker on an Intel Core i7 4.0 GHz CPU with 16GB RAM.

75
From the synthesis procedure (Algorithm 6), the protocols to complete the mission were obtained as shown in Fig. 4.7(b). For the first subgame, Fig. 4.8(a) shows how the horizon at which the geolocation task is performed impacts the probability of correctness that can be guaranteed, regardless of the adversarial actions, while Fig. 4.8(b) shows both the remaining distance to target and the increase in traveled distance, relative to the shortest path. The trends show that, as time passes without performing a geolocation task, the probability of its correctness decreases, while the remaining distance mostly decreases as the UAV approach the target. Interestingly, the strategy where the geolocation task is scheduled at horizon $h = 6$ provides a local minimum for the increase in traveled distance while maintaining a probability of correctness close to $h = 7$ and $h = 8$. This probability drops at $h \geq 9$ as guaranteeing that the UAV never encounter a hazard zone before the scheduled geolocation task becomes infeasible.

For the supergame, Fig. 4.8(c) shows the impact of the geolocation task budget on the probability of a successful mission. The graph conveys that more than 4 ge-
olocation tasks are required to guarantee a non-zero minimum probability of reaching the target against the worst-case attack. Fig. 4.8(d) shows the total expected mission time given a budget for the geolocation tasks, relative to the shortest-path time.

For subjective evaluation, an experiment was conducted where a number of participants were shown triplets of images $(I_a, I_b, I_c)$ as shown in Fig. 4.9(left). In each triplet, $I_a$ depicts a target location to which a shortest-path $\pi_b$ and safest-path $\pi_c$ plans are generated, where $I_b$ and $I_c$ are the sets of reachable locations if the UAV is under attack, respectively. The two images $I_b$ and $I_c$ are randomly withdrawn such that $I_b \in I_b \setminus I_c$ and $I_c \in I_c \setminus I_b$, i.e., each image is exclusive to the corresponding reachability set. Participants were asked to indicate which image from $(I_b, I_c)$ is less likely to be confused with $I_a$ (and hence safer). Fig. 4.9(right) shows that, while group A selections indicated no significant difference, group B indicated that locations reachable by the safest-path plan are less likely to be mistaken with $I_a$. In retrospective interviews, group A explained that they picked the upper right image most of the time as the tight time constraints gave them no time to inspect the lower right image, which may explain why $I_b$ and $I_c$ had similar chances of being selected since they were randomly ordered. On the contrary, group B found the given time to be sufficient to inspect both images. As these results highlight the effect of individual imagery skills on the geolocation task, further investigation may be needed to confirm this observation.

The performance results obtained for this case study are listed in Table 4.2. Note that, for the same grid size, more complex maps require more time for model checking, while the state space size remains unaffected. Although the number of states is $O\left((|A_{uav}|+|A_{adv}|)^h\right)$, the growth rate is typically reduced by the presence of hazard zones as the game stops branching at such states. Following the construction of the subgames, exploring the supergame itself consumes significantly less resources for
Figure 4.9: (left) An example of the image triplets used for subjective evaluation, and (right) the evaluation results.

Table 4.2: Performance results for using queries $\phi_{\text{syn}}$ and $\phi_{\text{ana}}$.

<table>
<thead>
<tr>
<th>Subgame $\hat{\phi}_a$</th>
<th>Model Size</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map $h$</td>
<td>States</td>
<td>Transitions</td>
</tr>
<tr>
<td>10x10</td>
<td>3</td>
<td>948</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5,976</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>28,345</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>119,078</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>490,021</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2,165,888</td>
</tr>
<tr>
<td>Supergame $\hat{\phi}$</td>
<td>12,704</td>
<td>18,171</td>
</tr>
</tbody>
</table>

4.7 Discussion and Conclusion

In this chapter, we have presented an approach to synthesize collaboration protocols for human-UAV command and control systems. The approach utilizes delayed-action games to model the system such that the ground truth is hidden from the UAV, rendering the model useful for synthesis using off-the-shelf tools. As the geolocation task is used to confirm the UAV location, the model includes a measure of geolocation task correctness. Moreover, experimental results have shown that the operators mostly adopted three geolocation strategies. By extracting the GIS features relevant to these strategies, machine learning techniques were deployed to predict the aforementioned model construction and analysis.
measure of correctness. Based on the developed model and a formal representation of system objectives, the DAG synthesis procedure uses PRISM-games model checker to synthesize and analyze the collaboration protocols. Finally, experimental analyses and subjective feedback were used to evaluate the synthesized protocols.

Though DAGs exploit parallel computation to reduce total processing time, the time to explore subgame horizons exponentially grows. Nevertheless, the DAGs as a formalism can benefit from approximate model checking techniques to explore larger horizons and higher orders of map discretization, where accuracy is traded-off with speed [KZ18].

Another improvement to this study would be to consider other factors affecting the operator’s performance, such as perceived workload, operator skill level, map complexity, and proper quantification of task correctness. Albeit challenging, this may open the door for individualized protocol synthesis. Moreover, available datasets for aerial images [XBD+18] can be examined to study the proposed approach on a wider scale. When it comes to advisory systems, however, one must carefully examine factors affecting human trust in autonomy — a quality that, if absent, may render collaborative protocols ineffective. Therefore, reinforcing trust through explanatory communication of those protocols is an avenue for future work.
Chapter 5

Error Recovery Protocols for MEDA Biochips

This chapter is an adapted reproduction of “Mahmoud Elfar, Zhanwei Zhong, Zipeng Li, Krishnendu Chakrabarty, and Miroslav Pajic. Synthesis of error-recovery protocols for micro-electrode-dot-array digital microfluidic biochips. ACM Transactions on Embedded Computing Systems (TECS), 16(5s):1–22, 2017” [EZL+17a]; and hence is not available under a Creative Commons license.

5.1 Introduction

Digital microfluidics enables the manipulation of droplets of picoliter volumes based on the principle of electrowetting-on-dielectric (EWOD) [ILC15]. Digital microfluidic biochips (DMFBs) are revolutionizing point-of-care diagnostics [SHT+08], high-throughput DNA sequencing [WLMF11], drug discovery [BNYPW08], and environmental toxicity monitoring [FKT+07]. However, today’s DMFBs suffer from several limitations, mainly: (i) inability to vary droplet volume in a fine-grained manner, (ii) the lack of integrated sensors for real-time detection, and (iii) the need for special fabrication processes and the associated reliability/yield concerns. To overcome the above limitations, a micro-electrode-dot-array (MEDA) architecture has been proposed recently [HWL+16, LYL15, LSL+15, LSL+15]. MEDA is based on the concept of a sea-of-micro-electrodes with an array of identical basic microfluidic unit components called microelectrode cells (MCs) [LSL+15], as illustrated in Fig. 5.1. Each MC consists of a microelectrode and a control/sensing circuit. MEDA allows microelectrodes to be dynamically grouped under program control to form a micro-component (e.g.,
mixer or diluter) that can perform different microfluidic operations on the chip. Prototypes of MEDA-based biochips have been fabricated using TSMC 0.35 µm CMOS technology.

A major obstacle that impedes reliable microfluidic operations on MEDA biochips is the lack of adaptive and efficient techniques that can facilitate recovery from unexpected errors. Faults in MEDA biochips may arise during bioassay execution. For example, excessive actuation voltage may lead to electrode breakdown and charge trapping [DPB08], and DNA fouling may lead to the malfunction of multiple electrodes in the biochip [SPPF04]. Such faults may eventually result in errors (e.g., a splitting operation with unbalanced droplets), which can adversely impact the correctness of fluidic operations.

In order to ensure robust fluidic operations and high confidence in the outcome of biochemical experiments, error-recovery techniques have recently been proposed [ZXC10, LCH13a, IC15, JBG15]. Zhao et al. [ZXC10] proposed an efficient
control-path design method based on error-propagation estimates for fluidic operations. Luo et al. [LCH13a] presented a cyber-physical approach that uses sensor data at intermediate checkpoints to dynamically reconfigure the biochip. Jaress et al. [JBG15] proposed a roll-back-based error-recovery technique that leverages a virtual topology to quickly obtain resynthesis results.

All the above techniques are targeted at conventional DMFBs; therefore, they cannot fully exploit the advantages specific to MEDA-based biochips, e.g., real-time droplet sensing. For MEDA biochips, Li et al. [LLY+16a] recently presented an error-recovery strategy based on local adaptation (i.e., “local recovery”), and analyzed it using probabilistic timed automata (PTA). A control flow was also proposed to connect local recoveries with global recovery. However, despite its benefits, this method has several shortcomings:

- It uses static error-recovery protocols for local errors, i.e., the error-recovery protocols are identical for the same type of errors irrespective of when or where on the chip they occur. This approach is not efficient for minimizing the recovery overhead, e.g., sample cost and chip-area impact.

- The solution in [LLY+16a] makes an unrealistic assumption that resources are always available for local recovery, e.g., there is always a sufficient number of fluidic modules for re-execution of operations. In practice, however, the available resources must be considered in making decisions about local recovery.

- In [LLY+16a], the same recovery time is assigned for each operation, which is not efficient in minimizing the time spent on error recovery. The recovery-time assignment must consider various factors, e.g., the type and the extent of detected errors, to determine the recovery time for each detected local error.

- Finally, [LLY+16a] uses all the assigned recovery time for local errors, and
it does not consider the likelihood that the local recovery can be successfully completed using less recovery time. Thus, a more adaptive synthesis technique is needed to derive online synthesis results to guide execution of the remaining microfluidic operations.

To overcome the above drawbacks, this chapter advances error-recovery by exploring (i) a flexible error-recovery solution derived using formal methods, and (ii) adaptive online synthesis based on real-time sensing results. The proposed approach provides error tolerance with significantly reduced recovery cost and higher probability of success. The main contributions of this chapter are:

- We model the error-recovery procedure using Markov decision processes (MDPs). We also describe the platform as another MDP, composing a unified model using stochastic multi-player games.

- We formalize the error-recovery objectives and use them to synthesize optimal error-recovery protocols for local recovery. The system behaviors under the optimal protocols are then analyzed.

- We describe an efficient method to determine available recovery resources for each detected error. Based on the available resources, we dynamically assign the recovery time for different local errors.

- We present an adaptive online synthesis flow to recompute new schedules, module placements, and droplet routes on-the-fly in response to errors.

The rest of the chapter is organized as follows. Sec. 5.2 presents an overview of MEDA biochips and formal methods that are relevant to this work. In Sec. 5.3, we present the problem formulation before describing the synthesis of error-recovery protocols for local errors in Sec. 5.4. Sec. 5.5 presents an adaptive online synthesis
technique for global error recovery. Experimental results on real-life benchmarks, as well as comparison with \cite{LLY+16a}, are presented in Sec. 5.6. Finally, Sec. 5.7 concludes the chapter.

5.2 Preliminaries

In this section, we provide an overview of digital microfluidics and MEDA, as well as experimental results that illustrate the need for probabilistic modeling of MEDA operations. We then provide an introduction to the formal methods used for modeling, design and analysis of stochastic systems, which are utilized for the synthesis of optimal error-recovery protocols in Sec. 5.4.

5.2.1 Digital Microfluidics and MEDA

A DMFB is able to manipulate and move picoliter droplets containing biological samples on a two-dimensional electrode array. MEDA extends this basic architecture by adding more flexibility \cite{LSL+15}. Furthermore, the size of the microelectrodes can be 10 times smaller (e.g., 100 µm in length) than conventional electrodes. The MEDA biochip consists of microelectrode cells (MCs). Each MC includes a microelectrode, an activation circuit, and a sensing circuit \cite{LYL15}.

Sensing Scheme in MEDA

In contrast to conventional DMFBs, real-time sensing can be achieved on MEDA biochips. MEDA biochips can detect the property (droplet-property sensing) and the location (droplet-location sensing) of on-chip droplets. Sensing results are presented in the form of a sensing map. Fig. 5.2 shows the diagram of the sensing circuit in a MEDA biochip. The parasitic capacitance with and without droplets between the top reference electrode and the bottom microelectrode is defined as $C_{Drop}$ and
\(C_{Empty}\), respectively. During droplet-location sensing, droplet precharge circuit first charges the parasitic capacitor. Transistor \(M_{N1}\) is then turned on to discharge the parasitic capacitor. Due to the difference between capacitances \(C_{Drop}\) and \(C_{Empty}\), \(INV\) outputs different voltage levels. Similar to droplet-location sensing, droplet-property sensing utilizes the capacitance difference between different types of droplets to generate different charging and discharging time.

Sensing on MEDA biochips can provide users with detailed information about the outcomes of on-chip operations. Therefore, errors can be detected in real-time, and error-recovery techniques can then be invoked for error correction.

**Outcome Classification**

As described in [LLY+16a], the outcomes of fluidic operations can be experimentally classified into three categories: *minor error*, *major error*, and *no error*. The classification is based on the level of completeness (LOC) of an operation. The LOC measures the extent to which an operation is complete within a pre-specified window, compared with the ideal case (e.g., a uniform mixing and a balanced splitting), and it ranges between 0 and 1 — a larger LOC represents a higher degree of completeness. Here, we use formulas from [LLY+16a] to calculate the LOC for every operation.
To obtain outcome probabilities for mixing and splitting operations, we performed 100 mixing and 100 splitting operations on a MEDA biochip. The corresponding LOC for each operation was obtained and the distribution of LOCs for mixing and splitting operations are shown in Fig. 5.3. Based on Fig. 5.3, the probabilities that the outcome of a mixing operation is a major error, minor error, and no error are 0.05, 0.13, and 0.82, respectively. Also, the probabilities that the outcome of a splitting operation is a major error, minor error, and no error are 0.09, 0.23, and 0.68, respectively.

5.2.2 Formal Modeling of Stochastic Processes

Markov Decision Processes (MDPs)

MDPs are a formalism widely used to model systems that exhibit both stochastic and nondeterministic behaviors. MDPs are similar to Finite-State Machines, with standard guards, specifying the logical conditions that enable transitions [BKL08], and nondeterministic actions/events that can be used in addition to probabilistic transitions. Stochastic behaviors capture scenarios when a system randomly, with predefined probability distributions, evolves from one state to another [Put14]. In contrast, nondeterministic behavior means that the system can evolve to the next state via any enabled transition, in a way that is unknown. Intuitively, nondetermin-
ism captures choices that a system (e.g., controller or environment) could make; in every state, once a nondeterministic choice is made, the next state is selected in a probabilistic manner, as in Discrete-Time Markov Chains (DTMC).

Formally, an MDP is specified as a tuple \( M = \langle S, A, \delta, s_0 \rangle \), where: (a) \( S \) is a finite set of states and \( s_0 \in S \) is the initial state; (b) \( A \) is a finite set of actions; and (c) \( \delta : S \times A \times S \rightarrow [0,1] \) is a transition probability function such that for all \( s \in S \) and \( \alpha \in A \), it holds \( \sum_{s' \in S} \delta(s, \alpha, s') \in \{0,1\} \). Note that if action \( \alpha \) is enabled in \( s \), no matter if it is active or inactive, the sum should be equal to 1; otherwise, it is equal to 0. We use \( A(s) \subseteq A \) to denote the set of enabled actions in \( s \). For any state \( s \in S \), it is required that \( A(s) \neq \emptyset \). When there is more than one action available in state \( s \) (i.e., \( |A(s)| > 1 \)), a nondeterministic choice should be made; on the other hand, if for all \( s \) it holds that \( |A(s)| \leq 1 \), then the MDP is effectively a DTMC.

The system evolution starts in \( s_0 \). At each step, the system moves from \( s \) to \( s' \) with probability \( \delta(s, \alpha, s') \) if there exists a transition \( s \xrightarrow{\alpha} s' \) such that the guard \( g \) holds and action \( a \) is active. Note that this transition is probabilistic if \( \delta(s, \alpha, s') \in (0,1) \) or deterministic if \( \delta(s, \alpha, s') = 1 \).

**Model Composition**

When modeling complex systems, it is more favorable to use a number of MDPs to capture the behavior of different system components instead of adopting a monolithic approach. Model composition can then be used to merge those models after defining the synchronization rules among them. Let \( M_1 = \langle S_1, A_1, \delta_1, s_1 \rangle \) and \( M_2 = \langle S_2, A_2, \delta_2, s_2 \rangle \) be two MDPs. The parallel composition of the MDPs is defined as

\[
M_1 \parallel M_2 = \langle S_1 \times S_2, A_1 \times A_2, \delta, (s_1, s_2) \rangle.
\]
Intuitively, MDPs are composed by synchronizing on common actions (the probability function is then the product of distributions for $\delta_1$ and $\delta_2$) and interleaving otherwise. In the former case, MDPs can synchronize over *channels*. Here, a transmitter over a channel $\alpha$ (denoted by $\alpha!$) must synchronously execute with all receivers (denoted by $\alpha?$) only if all guards associated with $\alpha$ are satisfied. More about MDP semantics and composition can be found in [Put14].

**Stochastic Multi-Player Games (SMGs)**

SMGs are similar to MDPs, where nondeterministic choices are resolved by more than one entity, called players. SMGs facilitate modeling systems with more than one source of nondeterminism. Formally, an SMG is defined as a tuple $G = \langle \Upsilon, S, A, \delta, s_0 \rangle$, where: (a) $\Upsilon$ is a finite set of players; (b) $S$ is a finite set of states, partitioned into disjoint set of states $S_\upsilon$, $\upsilon \in \Upsilon$; (c) $A$ is a finite set of actions; (d) $\delta : S \times A \times S \rightarrow [0, 1]$ is a partial transition function; and (e) $s_0 \in S$ is the initial state. A state $s \in S_\upsilon$ is controlled by player $\upsilon$, i.e., the nondeterministic choices from $s$ are controlled by $\upsilon$. If $s \in S_0$, then the next state is chosen probabilistically. More details on SMGs can be found in [CFK+13a].

**Specifications**

We use temporal logic to formally specify model properties that we would like to analyze. Specifically, to express properties for SMGs we use rPATL — Probabilistic Alternating-time Temporal logic with Rewards [CFK+13a]. For instance, consider a game $G$ where $\upsilon$ is a player and $\varphi$ is a predicate that is satisfied in some state $s$ (i.e., $s \models \varphi$). Consider the rPATL formulas

$$
\phi_1 := \langle \langle \upsilon \rangle \rangle \delta \geq q [F \varphi], \quad \phi_2 := \langle \langle \upsilon \rangle \rangle R^c_{\geq x} [F \varphi].
$$
Formula $\phi_1$ asks if $v$ can eventually reach a state $s$ satisfying property $\varphi$ (i.e., $s \models \varphi$) with probability that is greater than or equal to $q$, while $\phi_2$ checks if the accumulated rewards (associated with transitions and states) before reaching a state $s$ satisfying $\varphi$ (i.e., $s \models \varphi$) is greater than or equal to $x$. Alternatively, the quantitative queries

$$\langle \langle v \rangle \rangle_{\delta_{\text{max}}} = \text{?} [F \varphi], \quad \langle \langle v \rangle \rangle_{R_{\text{max}}} = \text{?} [F \varphi]$$

seek the maximal numerical values rather than assertions. More about specification semantics and rPATL can be found in [CFK13a].

**Strategies and Synthesis Problem**

Given an SMG $G$, a strategy $\pi$ is a set of rules to resolve all nondeterministic choices of a player $v$, reducing the model into an MDP $M^\pi$. The synthesis problem is an attempt to answer the following question: *Given an SMG $G$ and a specification $\phi$, find a strategy $\pi$ for player $v$ such that for all opponents strategies $\sigma$, the induced DTMC satisfies the specification — i.e., $G^\pi.\sigma \models \phi$ for all possible opponent strategies $\sigma$. More about strategy synthesis can be found in [SK16].

**5.3 Problem Formulation**

To map a bioassay to a MEDA biochip, synthesis techniques [LLY17] are required to bind the assay operation to on-chip resources and generate an optimized schedule of fluidic operations. The input bioassay is modeled by a sequencing graph [ILC15], which denotes the dependencies between fluidic operations. As discussed in Sec. 5.1, errors may occur during the execution of bioassays on a MEDA biochip. A MEDA biochip is said to have an error if its operation does not match its specified behavior. In this work, we assume that chips have been carefully tested before they can be used for bioassay execution, i.e., we assume that droplets will never be stuck during their
transportation and droplet dispensing can always be successfully achieved. However, some manufacturing defects may be latent, and they may produce errors during field operation [LLY+16a]. Thus, we focus on developing error-recovery approaches for online errors, i.e., mixing and splitting errors during bioassay execution.

For a given bioassay, we use on-chip sensing to evaluate the quality of output droplets of each mixing and splitting operation. Recall that MEDA can provide real-time sensing results for on-chip operations, and thus the time required for outcome evaluation is negligible. Once an error is detected, a local recovery approach is desired to recover from the error. *Local recovery* for a specific error refers to adaptation actions that are invoked to maximize probability of success for the operation that was affected by that error. However, local error recovery requires extra chip area and consumes time, which may result in a conflict with the original schedule [JBG15]. Consequently, we also focus on *global error recovery* to efficiently coordinate local-recovery procedures for the complete bioassay. The focus of global recovery is on (i) dynamic assignment of resources used for local recoveries, and (ii) generating new synthesis results (i.e., operation scheduling and module placement) to reduce the interference between error-recovery procedures and bioassay execution. The relationship between local and global error recovery is shown in Fig. 5.4. The formal models reviewed in Sec. 5.2 and described in more detail in Sec. 5.4 allow us to make informed decisions about the local recovery procedures, and hence they also play a role in efficient global recovery.

The objective of this work is to develop an error-recovery strategy that can minimize the impact on bioassay completion time and maximize the probability of successful error recovery. The strategy includes approaches for both local recovery (for erroneous operations) and global recovery (for the complete bioassay). For local recovery, we introduce an optimal recovery approach that maximizes the probability
of successful recovery based on available resources. For global recovery, we introduce an online synthesizer to seamlessly connect local recovery with global recovery.

### 5.3.1 Strategy Overview

The proposed error-recovery strategy (Fig. 5.5) contains two major components: (i) online synthesizer, which is used to dynamically adjust operation scheduling and module placement, and (ii) local error recovery model that, based on available resources, provides an optimal local-recovery protocol. These components seamlessly coordinates with each other for the implementation of the proposed error-recovery strategy on the hardware platform.

Details of the error-recovery model and online synthesizer are presented in Sec. 5.4 and Sec. 5.5 respectively. The error-recovery model is used offline to generate optimal error-recovery protocols for any level of available resources. Then, all these optimal error-recovery protocols are stored in a lookup table. At runtime, if an error is detected during the execution of a bioassay, the online synthesizer first performs...
an analysis on available resources and then obtains the optimal recovery protocol from the constructed lookup table. Finally, the online synthesizer dynamically adjusts synthesis results to avoid interference between error-recovery protocols and the execution of following operations.

5.4 Synthesis of Local Error-Recovery Protocols

In this section, we introduce a method to design optimal error recovery protocols for any level of available system resources (e.g., time for recovery, mixers). We start by constructing MDP models of the error recovery process for various fluidic operations. These models, along with an abstracted model of the platform, are obtained from the experimental data presented in Sec. 5.2.1 and then integrated using model composition to obtain an SMG model of the system. Finally, we formalize the error-recovery process objectives and use them to automatically synthesize optimal error-recovery protocols.
5.4.1 Error-Recovery Model

Consider a bioassay that includes a combination of mixing, splitting and dilution operations. Fig. 5.6 shows the data flow structure of the error-recovery process, which forms the basis of our formal model. The model captures the control execution as well as the probabilistic behavior of the platform. When a bioassay operation is performed, the resource manager updates the amount of resources that can be allocated for error recovery (i.e., available time $r_{time}$, mixers $r_{mix}$, and backup droplets $r_{cbu}$), while the platform provides the information about the operation outcome. Based on this information, the error-recovery model identifies both optimal actions and resources required to recover from the error.

Mixing Error-Recovery Model ($M_{mix}$)

We start by considering the error-recovery model $M_{mix}$ for mixing operations (see Fig. 5.7). Initially, the model moves to Check state, where the process outcome is checked. This outcome falls into one of three categories as shown in Fig. 5.3, namely successful, unsuccessful with minor error or with major error, upon which the model moves to the corresponding state. For successful outcome, the model returns to Start, signaling the action doNothing. Otherwise, in Minor Error or Major Error states, four actions are available to choose from.
The first three actions are to reuse both the same droplet and mixer (\(a_{\text{mix}}\)), to move the droplet to a new mixer (\(a_{\text{mov}}\)), and to use a backup droplet to complete the operation (\(a_{\text{cbu}}\)). Each of these actions is guarded by the availability of the required resources, and further ensures that such resources are consumed by synchronizing with the resources model (described in Sec. 5.4.1). Depending on the selected action, the model progresses to \(\text{Mix}\), \(\text{Move}\) or \(\text{Call BU}\) state, before returning to \(\text{Check}\) by according updating the variable \(\text{action}\). The fourth action \(a_{\text{rsc}}\) captures rescheduling of the mixing operation, when the process aborts by moving from \(\text{Fail}\) to \(\text{Start}\).

**Splitting Error-Recovery Model (\(M_{\text{split}}\))**

Similar to the mixing operation, an error-recovery protocol for a splitting operation starts by reading its outcome. The process aborts by moving from \(\text{Check}\) to \(\text{Start}\) states if the operation is successful. Otherwise, it moves to either \(\text{Minor Error}\) or \(\text{Major Error}\) states, from which it can proceed with \(a_{\text{merge}}\), \(a_{\text{reroute}}\) or \(a_{\text{rsc}}\) actions to
Merge, Reroute or Schedule states, respectively. While actions $a_{merge}$ and $a_{reroute}$ request merging the defective droplets or rerouting to a new splitter before any further splitting attempts, the action $a_{resc}$ requests rescheduling. The variable $action$ is used to store the decision made upon synchronizing over return action.

Dilution Error-Recovery Model

In a dilution operation, a droplet first undergoes a mixing operation, after which it is split into two droplets. Hence, we construct the dilution error-recovery model $M_{dilute}$ as $M_{mix}$ followed by $M_{split}$.

Resources Model

Each error-recovery action is associated with specific resource demands, as shown in Table 5.1. We model these resources as DTMCs from Fig. 5.9 with: $M_{time}$ tracking time through $r_{time}$ variable, $M_{mixers}$ tracking the number of mixers $r_{mix}$, $M_{drop}$ tracking the number of backup droplets $r_{cbu}$, $M_{actMix}$ tracking current mixer condition through mixer variable, and $M_{actDrp}$ tracking current droplet condition droplet.
All five variables are initialized by receiving \textit{init} action, triggered by $\mathcal{M}_{\text{plat}}$ (Fig. 5.10). These variables are also updated by one or more of the actions $a_{\text{mix}}$, $a_{\text{mov}}$, $a_{\text{cbu}}$, $a_{\text{mrg}}$, and $a_{\text{rrt}}$, triggered by $\mathcal{M}_{\text{mix}}$ (as in Fig. 5.7) or $\mathcal{M}_{\text{split}}$ (Fig. 5.8). The action \textit{use}, triggered by $\mathcal{M}_{\text{plat}}$ (Fig. 5.10 described in Sec. 5.4.1), marks the current droplet and mixer as used by assigning the value \textit{isOld} to both \textit{droplet} and \textit{mixer} variables, while actions $a_{\text{mov}}$ and $a_{\text{cbu}}$ set the variables back to \textit{isNew}.

**Platform Model ($\mathcal{M}_{\text{plat}}$)**

Since our focus is on the local error-recovery process, it is imperative that the system model captures two fundamental aspects — operation outcome and resources.
Table 5.2: Probability distributions over the possible outcomes of mixing and splitting process.

<table>
<thead>
<tr>
<th>Resource Droplet</th>
<th>Previous Mixer Outcome</th>
<th>Mixing Outcome p_s</th>
<th>p_{f1}</th>
<th>p_{f2}</th>
<th>Splitting Outcome p_s</th>
<th>p_{f1}</th>
<th>p_{f2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>New</td>
<td>0.820</td>
<td>0.130</td>
<td>0.050</td>
<td>0.680</td>
<td>0.230</td>
<td>0.090</td>
</tr>
<tr>
<td>New</td>
<td>Used minor error</td>
<td>0.656</td>
<td>0.248</td>
<td>0.096</td>
<td>0.408</td>
<td>0.426</td>
<td>0.166</td>
</tr>
<tr>
<td>New</td>
<td>Used major error</td>
<td>0.164</td>
<td>0.232</td>
<td>0.604</td>
<td>0.204</td>
<td>0.224</td>
<td>0.572</td>
</tr>
<tr>
<td>Used</td>
<td>New minor error</td>
<td>0.492</td>
<td>0.367</td>
<td>0.141</td>
<td>0.544</td>
<td>0.328</td>
<td>0.128</td>
</tr>
<tr>
<td>Used</td>
<td>New major error</td>
<td>0.246</td>
<td>0.209</td>
<td>0.545</td>
<td>0.136</td>
<td>0.243</td>
<td>0.621</td>
</tr>
<tr>
<td>Used</td>
<td>Used minor error</td>
<td>0.328</td>
<td>0.485</td>
<td>0.187</td>
<td>0.272</td>
<td>0.523</td>
<td>0.205</td>
</tr>
<tr>
<td>Used</td>
<td>Used major error</td>
<td>0.082</td>
<td>0.255</td>
<td>0.663</td>
<td>0.068</td>
<td>0.262</td>
<td>0.670</td>
</tr>
</tbody>
</table>

availability. In a typical MEDA biochip, on-chip sensors provide real-time measurements of a process outcome. Being unknown at design stage, we assume that the outcome follows a uniform probability distribution, an assumption supported by the experimental results presented in Section 5.2.1. Nevertheless, one limitation of the aforementioned results is that they only characterize the outcome probability of an operation at its first attempt. Hence, in Table 5.2 we augment these results with a model of how the error type of last trial and droplet and mixer conditions may influence the outcome probability of a recovery operation, mostly due to mixer degradation as discussed in [ICS17, NFKS14]. We assume that the probability of the outcome to be successful is the highest when the droplet and mixer used are both new. Similar model is used for splitting operations.

Fig. 5.10 shows the MDP $M_{plat}$ that captures the probabilistic behavior of the platform by abstracting the scheduler, controller, sensors and biochip. The model moves from Start to Do Operation state when an operation is being processed. The transition from Do Operation to Check is associated with the init action, which synchronizes with the resources model (from Fig. 5.9) to initialize the available resources. The transition exiting Check state is sampled from a uniform distribution characterized by $p_s$, $p_{f1}$ and $p_{f2}$. These probabilities, rather than being constant, depend on the variables droplet, mixer, and err (from Table 5.2), where err denotes the last
type of error. Any of these three probabilistic transitions triggers the action use, marking both the droplet and mixer as used, and assigns a new value to err.

If the sampled outcome leads to either Minor Error or Major Error, the model proceeds to Error Recovery. The next state is defined according to the action defined by $M_{mix}$ or $M_{split}$ through the variable action. Any action other than rescheduling leads to Do Action state and subsequently back to Do Operation state, where the operation is repeated after the corrective action is done. In contrast, if action = doResch, the model moves to Reschedule state, which marks unsuccessful attempt to recover from the error. On the other hand, if the sampled outcome leads to Success, the operation ends. While $M_{plat}$ in Fig. 5.10 models a single bioassay operation, for a sequence of operations the model can be augmented with the grayed extension (Fig. 5.10) to execute in a loop parsing a list of operations.

5.4.2 Model Composition

The aforementioned models present the components and functionalities of a MEDA-based system. The overall system model is obtained by composing those models
along with a synchronization model $M_{sync}$ from Fig. 5.11 to ensure a turn-based synchronization — i.e., that only one action is triggered at a time. This is achieved using the variable $player$ that indicates which model is allowed to execute actions.

In addition, all transitions of models capturing error-recovery process (i.e., $M_{mix}$ and $M_{split}$) are guarded by $player = erp$, while all transitions of the platform model $M_{plat}$ are guarded by $player = plat$. All other models, capturing available resources, are reactive as they do not trigger any actions, and thus none of the player-based global guards are needed.

As the error-recovery protocol is triggered after a platform error occurs, $M_{sync}$ is initialized with $player = plat$, i.e., $M_{plat}$ progresses first. Whenever the platform model decides on the operation (i.e., $init$ is triggered — see Fig. 5.10), $M_{plat}$ is halted (i.e., $player = erp$). Then, depending on the selected operation, either $M_{mix}$ or $M_{split}$ becomes active. Similarly, when an operation either succeeds or fails (i.e., $return$ is triggered — see Fig. 5.7 and Fig. 5.8), the control goes back to the platform model $M_{plat}$.

The overall system model can be established through parallel composition. If initial values for the resources are assumed to be fixed then only one source of nondeterminism exists, and hence the system model is an MDP. However, synthesized protocols based on this assumption are only optimal for that specific number of available resources. We relax this assumption by allowing $M_{plat}$ to nondeterministically select the initial (i.e., available) resources. Thus, the overall system model is an SMG,
denoted by $G_{sys}$, and defined as
\[
G_{sys} = M_{plat} \parallel M_{mix} \parallel M_{split} \parallel M_{time} \parallel M_{mixers} \\
\parallel M_{drop} \parallel M_{actMix} \parallel M_{actDrp} \parallel M_{sync}.
\]

Finally, the game is played between the error-recovery process, denoted by $erp$, and the platform, denoted by $plat$.

5.4.3 Strategy Synthesis

In order to synthesize optimal strategies for error recovery, we use the composed SMG model $G_{sys}$ to find an error-recovery policy that satisfies a specification capturing the desired optimality objective. We define the objective (i.e., specification) of the optimal protocol as follows: *Given a set of resources, find the recovery action that maximizes the probability that the process is eventually successful.* This objective can be formally expressed with the rPATL query [CFK+13a]

\[
\phi := \langle \langle erp \rangle \rangle P_{\text{max}}=? \varphi := F Succeed,
\]

where $F$ is the temporal operator $eventually$. Hence, the synthesis problem aims to find the optimal strategy $\pi^*$ such that

\[
G_{sys}^{\pi^*,\sigma} \models \phi \quad \forall \sigma \in \Sigma,
\]

where $\Sigma$ is the set of all possible opponent (i.e., platform) strategies. Although the query $\phi$ results in a single value for $P_{\text{max}}$, the synthesis problem seeks the strategy that maximizes the probability $P$ for every possible opponent strategy $\sigma \in \Sigma$, i.e., for every possible assigned resources, resulting in an optimal error-recovery protocol.

We use PRISM-games [CFK+13b] to implement $G_{sys}$ along with the query $\phi$. The maximum probability of satisfying $\varphi$ at state $s$ can be found using value iteration algorithm [CFK+13a], which is implemented in PRISM-games. For the obtained
optimal recovery policies, in Sec. 5.6.1 we investigate the relationship between several attributes, such as the effect of resource availability on the probability of success for these operations.

5.5 Global Error Recovery

The local error-recovery protocols, derived offline using the techniques from Section 5.4, depend on the upper limit on the recovery time and the set of available resources that can be used for recovery. However, to adaptively invoke error recovery for the bioassay, we need a global error-recovery technique that can tackle two key problems. First, to determine an optimal error-recovery policy, the local error recovery model must have knowledge of the available resources, e.g., the number of fluidic modules and backup droplets, as well as the time available for recovery. Second, multiple errors can occur at different locations on the chip at nearly the same time, and the error-recovery procedures for these errors can be intertwined through resource sharing or droplet-path overlap. Therefore, there is a need to dynamically generate new schedules and module placements for error recovery and other bio-protocol-related operations, such that the adverse impact of error-recovery procedures on protocol execution is minimized. In this section, we first describe how we determine resource availability. Next, an online synthesis approach is presented to dynamically generate new schedules and module placements in response to local-recovery decisions.

5.5.1 Inputs to Local Error Recovery

Recall that the local error recovery requires information about the number of backup droplets, the number of fluidic modules available, and the maximum allowable recovery time. We next describe how these parameters are determined.
Available Backup Droplets

For a splitting or a dilution operation, if only one of its output droplets is used as an input for an immediate successor, the other (redundant) droplet is referred to as a backup droplet for possible error recovery. In addition, dispensing operations can be scheduled for execution as early as possible and extra droplets can be stored on the biochip as backup. Unused backup droplets are sent to the waste reservoir upon the completion of the bioassay.

The number of backup droplets available for an operation can be determined from the sequencing graph. In a sequencing graph, each node represents an operation. Let $BU(O_i)$ denote the number of backup droplets generated by operation $O_i$. Suppose operation $O_j$ is an immediate successor of the $N$ operations denoted by $O_{j_1}, O_{j_2}, \ldots, O_{j_N}$. Then, the number of available backup droplets for the operation $O_j$ is $\min(BU(O_{j_1}), BU(O_{j_2}), \ldots, BU(O_{j_N}))$, since operation $O_j$ requires one droplet from each of its immediate predecessors.

Available Fluidic Modules

A fluidic module is a group of microelectrodes on a MEDA biochip that can be configured to perform a type of operation (e.g., mixing). In order to determine the number of fluidic modules available for local error recovery, we use a search algorithm based on the notion of a forbidden set introduced in [LLY+16b]. A forbidden set refers to a set of locations where new fluidic modules cannot be placed.

If an error occurs for an operation, the algorithm attempts to place appropriate fluidic modules on the biochip so that they can be used for local error recovery. The forbidden set is used to avoid placement conflicts with other operations. This algorithm terminates when either (1) a total of $N$ fluidic modules are placed on the biochip, where $N$ refers to the maximum number of fluidic modules designated for
use in local error recovery (determined by the local error-recovery model), or (2) not enough space is available to place a fluidic module. When this algorithm terminates, it provides the number and the locations of fluidic modules for local error recovery.

The pseudo-code for determining the number of fluidic modules is shown in Fig. 5.12. The parameter fluidic_modules is a container, which stores the placements of available fluidic modules (line 1). According to the current module placement MP, we use the function getPlacementSet() to obtain the set of available locations PS for a fluidic module (line 2). If PS is not empty, i.e., there is sufficient space to place a new fluidic module, the function findBestPlacement() is used to find the module placement MP_{new} with minimum cost (lines 3–4). The cost is defined as the Manhattan distance \[ LLY + 16b \] between the target fluidic module and the erroneous fluidic module. The newly identified fluidic-module placement is stored in fluidic_modules (line 5). Next, we examine the number of fluidic-module placements in the container: (1) if we already have N fluidic-module placements, the search is stopped; (2) otherwise, the newly found fluidic-module placement MP_{new} is added to the on-chip module placement set MP, and we continue the search for another fluidic-module placement based on the updated MP (lines 6–9). Finally, when the search stops, we delete the newly added fluidic module placements from MP (lines 12–14) and get the number and the locations of fluidic modules from fluidic_modules (line 15). For a MEDA biochip with an \( M \times N \) microelectrode array, the computational complexity of this search algorithm is \( \mathcal{O}(MN) \).

Maximum Recovery Time

When more recovery time is available, additional error-recovery operations can be executed and the probability of success will be correspondingly higher. However, if excessive time is allowed for error recovery, there is increased risk that the bioassay
**Input:** Module placement $MP$;  
**Output:** Number of fluidic modules $N$ and locations of fluidic modules $LO$;  
1: $\text{fluidic\_modules} := \{\}$;  
2: $PS := \text{getPlacementSet}(MP, \text{module\_type})$;  
3: while not $PS$.empty do  
4: $MP_{\text{new}} := \text{findBestPlacement}(PS)$;  
5: $\text{fluidic\_modules}.\text{append}(MP_{\text{new}})$;  
6: if $\text{fluidic\_modules}$.size() = $N$ then break;  
7: else  
8: $MP.\text{add}(MP_{\text{new}})$;  
9: $PS := \text{getPlacementSet}(MP, \text{module\_type})$;  
10: end if  
11: end while  
12: for $MP_{\text{new}}$ in $\text{fluidic\_modules}$ do  
13: $MP.\text{delete}(MP_{\text{new}})$;  
14: end for  
15: $N := \text{fluidic\_modules}$.size(); $LO := \text{fluidic\_modules}$;  
16: return $N$, $LO$;  

**Figure 5.12:** Pseudo-code for determining the number of available fluidic modules.

will miss the completion-time deadline. Therefore, it is important to assign a maximum recovery time for the recovery procedure when an error occurs in an operation. Here, we utilize the scheduler from [LLY+16b] to generate 20 feasible schedules for the remaining operations by varying the recovery time $t_r$ from 1 s to 20 s. As a result, we get 20 completion times corresponding to these 20 schedules. Note that we set the upper limit of $t_r$ to 20 seconds, since there is no significant increase in the probability of success for $t_r > 20$ s for the range of laboratory bio-protocol that we studied (as we illustrate in Section 5.6). Finally, the maximum recovery time $\hat{t}_r$ for local error recovery is equal to the maximum value of $t_r$ that satisfies the inequality $T(t_r) \leq T_0 + \Delta T_{\text{max}}$, where $T_0$ is the completion time of the bioassay when no error occurs, $\Delta T_{\text{max}}$ is the maximum allowable increase in the completion time due to errors, and $T(t_r)$ denotes the completion time when $t_r$ time is allocated for the error recovery.

We evaluate the CPU time needed for global recovery for three bioassays described
in Section 5.6. The maximum time to generate a schedule for these bioassays is 0.05 s on an Intel Core i3 with a 3.7 GHz CPU and 8 GB memory; thus, generating 20 schedules requires 1 s. To speed up the search for the maximum allowable recovery time, we use binary search to generate fewer schedules and quickly find the largest value of $t_r$ that leads to an admissible schedule.

### 5.5.2 Online Synthesis

We now describe how online synthesis can be used to integrate local error recovery with global error recovery, as shown in Fig. 5.13. When a bioassay is mapped to the biochip, the completion-time deadline is also specified by the user. If errors occur, the completion time of the bioassay will be larger than $T_0$, so we use $T_0 + \Delta T_{\text{max}}$ as an upper bound on the acceptable completion time. If the completion time $T$ satisfies the condition $T \leq T_0 + \Delta T_{\text{max}}$, the bioassay is deemed to be successful; otherwise, it fails.
We first generate an initial schedule and module placement. All operations are stored in the operation queue in ascending order of start times and retrieved sequentially. Before executing new operations, the controller of the MEDA biochip examines the status of all on-going local error recoveries and: (1) if any local error-recovery procedure fails (e.g., the recovery time exceeds the maximum allowable recovery time), the bioassay fails; (2) if a local error recovery finishes within the maximum allowed recovery time, the schedule and module placement will be updated to carry out the remaining operations as soon as possible.

Following this step, the controller on the MEDA biochip checks if there is any new error. If a new error occurs, the controller determines the resources and time available for recovery from that error, and obtain the optimal error-recovery policy generated by the techniques described in Section 5.4. Note that these error-recovery policies are obtained offline (i.e., at design time) and stored in a lookup table, and therefore no time is required for their computation at runtime. To accommodate the obtained optimal error-recovery policy, the schedule is updated appropriately.

5.6 Experimental Results

In this section, we first evaluate our local error-recovery approach by focusing on the composed model from Sec. 5.4. Next, we demonstrate the effectiveness of our global error-recovery technique by presenting results of simulating our technique on three representative real-life bioassays. Finally, using the same benchmarks, we compare our approach with the static protocol proposed in [LLY+16a], since the latter significantly outperforms other existing methods.
5.6.1 Model Analysis

Since the optimal synthesized strategy $\pi^*$ resolves the nondeterminism of erp, the resulting game $G^{\pi^*}$ is reduced to an MDP where platform controls the available resources. For a specific strategy $\sigma$ of platform, the game $G^{\pi^*, \sigma}$ is reduced to a DTMC. Thus, we use the induced model to study the system under the optimal policy.

To understand the impact of available resources on the error-recovery process, we run experiments on $G^{\pi^*, \sigma}$ where $\sigma \in \Sigma$ and $\Sigma$ is a set of finite plat strategies constructed by choosing a finite set of values for each resource, namely

$$r^{init}_{time} \in \{0 : 2 : 20\}, \quad r^{init}_{mix} \in \{0 : 1 : 4\}, \quad r^{init}_{cbu} \in \{0, 1\}. \quad (5.1)$$

We can then obtain the expected success probability $p_{max}$ under the optimal policy $\pi^*$ and all possible initial sets of available resources $(r^{init}_{time}, r^{init}_{mix}, r^{init}_{cbu})$. We formally describe the query as

$$p^{{\pi^*, \sigma}}_{max} =? \ [ F \ Succede \ ] \ \forall \sigma \in \Sigma. \quad (5.2)$$

The computational complexity of solving (5.2) is $O(N_t^3 N_m^3 N_c^3)$, where $N_t$, $N_m$ and $N_c$ are the number of values for time units, mixers, and backup droplets, respectively (e.g., 11, 5 and 2 in (5.1)). For the set of initial resources from (5.1), PRISM-games solved the query and computed the optimal local error-recovery policy in 2.68 s on an Intel Core i7 4.0 GHz CPU. Recall that this computation is performed only once offline. Fig. 5.14 shows the obtained results, illustrating the effect of resource availability on the probability that a mixing operation is successfully performed under the optimal error-recovery protocol. The general trend shows that the more the resources available, the higher the success probability is. Since the mixing process takes 2 s to complete, time increments of 1 s do not result in change in probabilities.
Figure 5.14: Effect of resource availability for a mixing operation on its success probability under the optimal error-recovery protocol.

With the same available amount of time, the improvement in probabilities between having (0 mixers, 0 droplets) and (3 mixers, 1 droplet) peaks when the time is at least 4 s (5.38%) then takes a downward trend as time is increased. The combination (4 s, 2 mixers, 1 droplet) has the lowest time required to guarantee a success probability of at least 95%. If more time is afforded, (6 s, 2 mixers, 0 droplets) can guarantee approximately the same probability.

5.6.2 Benefits of the Dynamic Protocol

One significant advantage of our proposed dynamic recovery protocol over the static one from [LLY+16a] is the higher probability to recover from erroneous operations, especially those residing on the side branches of a sequencing graph. The reason is that a static protocol follows a fixed policy to recover from erroneous operations, failing to exploit the variance of time and other on-chip resources. On the other hand, a dynamic protocol generates the optimal recovery policy based on the currently avail-
able resources. To illustrate this, consider the bioassay protocol shown in Fig. 5.15 which has a long critical path (from $D_1$ to $S_4$) and a long side branch path (from $D_8$ to $M_5$). The schedule of this particular bioassay when no error happens is shown in Fig. 5.16(a). Here, the completion-time deadline is 24 s, which means the maximum allowed completion time increment $\Delta t_{max}$ is 2 s. We discuss the following two cases:

1. If an error occurs in the critical path (e.g., in $M_4$). As shown in Fig. 5.16(b), the maximum recovery time $\hat{t}_r$ for both the static and dynamic protocols is 2 s. Within this period of time, the probability of success for local recovery is 0.3 and 0.57 for the static and dynamic protocols, respectively.

2. If an error occurs in the side branch path (e.g., in $M_7$). Since the static protocol relies on a fixed policy, as shown in Fig. 5.16(c), the maximum recovery time for $M_7$ to recover from an error is 2 s, and the probability of success for error recovery remains 0.3. However, the dynamic protocol examines the available time and other on-chip resources for error recovery and generate the optimal error-recovery policy. Fig. 5.16(d) shows that 8 s are used to recover $M_7$, and the probability of success for error recovery significantly increases to 0.9.
Figure 5.16: The schedule when error (a) never occurs, (b) occurs in $M_4$, (c) occurs in $M_7$ using static protocol, (d) occurs in $M_7$ using dynamic protocol.

### 5.6.3 Results for Bioassays

We evaluated the proposed global error-recovery method on three real-life benchmarks shown in Fig. 5.17, namely CEP, master-mix, and serial-dilution [ICS17]. CEP is a combination of three small bioassays: cell lysis, mRNA extraction and mRNA purification. The experimentally characterized module library for MEDA is presented in [LLY+16a]. The size of one electrode in a conventional DMFB is equal to a $4 \times 4$ microelectrode array in MEDA biochips. Therefore, a $2 \times 2$ array in the MEDA platform actually represents an $8 \times 8$ microelectrode array. We then set the chip size to be an $8 \times 8$ array.
The error-recovery capability of a MEDA-based biochip can be evaluated based on both the bioassay completion time and probability of success (POS) when errors are detected. In our simulation, we randomly inject up to four errors into each benchmark and simulate 1000 execution scenarios under each configuration. Then, we calculate the average completion time and probability of success under different $\Delta t_{\text{max}}$, where $\Delta t_{\text{max}}$ denotes the maximum allowed bioassay completion time increment. For example, if $\Delta t_{\text{max}}$ is set to 10 s and the completion time for a bioassay without errors is 20 s, then the bioassay completion-time deadline is 30 s.

As shown in Fig. 5.18, a larger $\Delta t_{\text{max}}$ and a smaller number of inserted errors result in higher probability of success for all three bioassays. Nevertheless, this comes at the cost of a slight increase in the completion time. Therefore, a trade-off between the completion time and the probability of success exists.

To compare the proposed dynamic protocol with the static one from [LLY+16a], we simulate 1000 times per each number of errors from one to four, where the location at which the errors are injected is randomized. We compare the completion time.
Figure 5.18: The average completion time using the proposed dynamic error-recovery protocol for (a) CEP, (b) master-mix, and (c) serial-dilution; the probability of success for (d) CEP, (e) master-mix, and (f) serial-dilution.

Figure 5.19: A comparison between the average completion time of the static protocol from [LLY+16a] and the proposed dynamic protocol for (a) CEP, (b) master-mix, (c) serial-dilution; and their POS for (d) CEP, (e) master-mix, and (f) serial-dilution.
under the same probability of success, as well as the probability of success under different $\Delta T_{\text{max}}$ — the results for both protocols are shown in Fig. 5.19. When only one error occurs, the dynamic protocol shows modest improvement. However, when two or more errors occur, it significantly outperforms the static protocol. Finally, from the experimental data presented in Section 5.2.1, the probabilities of two or more errors occurring in the CEP, master-mix, and serial-dilution benchmarks are 0.625, 0.947, and 0.986, respectively, highlighting the need to employ the proposed dynamic error-recovery protocol.

5.7 Discussion and Conclusion

We have presented the first work that can automatically synthesize optimal error-recovery protocols for MEDA biochips. We first model the error-recovery procedure using Markov Decision Processes (MDPs). Along with the abstract model for the platform, we obtain a stochastic multi-player game (SMG)-based system model. We then formalize the error-recovery procedure objectives and use them to generate optimal error-recovery protocols for local recovery. We also propose a global-recovery technique to dynamically (i) assign resources for local recovery and (ii) generate new synthesis results, i.e., operation-scheduling and module-placement results. Although the local recovery plans are optimal, the global scheduling policy is based on heuristics. As part of future work, we will focus on deriving more efficient scheduling algorithms. Finally, simulation results from three real-life bioassays demonstrate the effectiveness of the proposed error-recovery technique.

As a biochip ages, the probability that an error occurs in a fluidic operation is more likely to increase over time. Hence, future work can extend the model such that the probability distribution over possible outcomes is dynamically updated, potentially by gathering statistical data from the biochip. In that case, online protocol synthesis
can be useful to realize such implementation.

Furthermore, the life of a biochip can be prolonged if less error-recovery operations are executed. The objective of the error-recovery protocol can then be relaxed such that, in the early stages of a bioassay, a number of minor errors are tolerated before performing any local error-recovery attempt. Thus, a balance between minimizing the time required to complete a bioassay and extending the life of the biochip can be achieved.
Chapter 6

Stochastic Game-based Modeling for MEDA Biochips

This chapter is an adapted reproduction of “Mahmoud Elfar, Tung-Che Liang, Krishnendu Chakrabarty, and Miroslav Pajic. Formal synthesis of adaptive droplet routing for MEDA biochips. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2021” [ELCP21a]; and hence is not available under a Creative Commons license.

6.1 Introduction

Digital microfluidic biochips (DMFBs) are being adopted for biomolecular recognition, point-of-care diagnostics, and air-quality monitoring applications [CLY+15, S+20, HCKF20]. A microfluidic biochip manipulates tiny amounts of fluids to automatically execute biochemical protocols for point-of-care clinical diagnosis with high efficiency and fast sample-to-result turnaround [GMB+20, She20]. Because of these characteristics, the Rapid Acceleration of Diagnostics (RADx) initiative from the National Institutes of Health has recently awarded grants to several biomedical diagnostic companies to develop microfluidic technologies that can dramatically increase COVID-19 testing capacity and throughput [NIH]. This technology has been commercialized in recent years for clinical diagnostics and immunoassays [Genne, Baene].

Micro-electrode-dot-array (MEDA) biochips have been proposed in recent years to further advance DMFB technology [LYL15]. A MEDA biochip manipulates fluids as discrete droplets of picoliter volume using the principle of electro-wetting-on-dielectric (EWOD) [QB01]. In addition, MEDA biochips also offer fine-grained fluidic control
and real-time droplet sensing on a two-dimensional array of microelectrodes \cite{QB01,LYL15}. Multiple microelectrodes can be dynamically grouped to form a fluidic module (e.g., splitter or mixer). MEDA biochips have been fabricated using TSMC 0.35\,\mu m CMOS technology \cite{HWL+16}.

In the MEDA platform, a real-time capacitive-sensing circuit is integrated with each microelectrode to detect the location and properties of a droplet. In each operational cycle, the sensing circuit discharges and charges the microelectrode, and measures the charging time. The charging time is used to detect whether a droplet is present over the microelectrode. To obtain the positions of on-chip droplets, the sensing results of all the microelectrodes are shifted out using a scan chain.

Prior work has identified a number of failure mechanisms for DMFBs \cite{XC09}. Some of these are related to manufacturing defects; post-fabrication testing can be used to screen for such defects \cite{SOC05}. However, charge trapping in the dielectric layer and degradation of the insulator can result in microelectrode degradation \cite{VP99,DPB09}. If an electrode is degraded during bioassay execution, fluidic operations associated with this degraded electrode will fail, resulting in bioassay failure \cite{ZLC+18b,LZ20}. Hence, to reliably execute bioassays on MEDA biochips, we must proactively avoid the use of degraded microelectrodes. The majority of literature on reliable execution of bioassays on both DMFBs and MEDA biochips has focused on error recovery techniques. Fundamentally, error recovery techniques aim to reactively recover from errors in microfluidic operations after they occur. However, error recovery techniques may require discarding current droplets and repeating a number of microfluidic operations, potentially losing expensive and/or hard to reproduce droplets.

In this chapter, we address the problem of bioassay failure due to microelectrode degradation by proposing an adaptive routing strategy synthesis framework. We
first present a new microelectrode circuit design that provides dynamic health information about the microelectrode degradation level in real time. Next, we develop a stochastic game-based model for MEDA biochips that incorporates the dynamic health information. The developed model is further used to induce Markov decision processes (MDPs) to present a scalable formal synthesis method for droplet routing that can dynamically change droplet transportation routes based on the real-time health information obtained. The main contributions of this chapter are as follows.

- We introduce a MEDA biochip microelectrode design that enables the sensing of microelectrode health level in real time.

- We study how the microelectrode degradation patterns are affected by droplet characteristics for various bioassays. We also present experimental results on electrode degradation for fabricated PCB prototypes. These results are used to validate the degradation model used throughout the chapter.

- We develop a stochastic game-based model for droplet manipulation in MEDA biochips that incorporates the health information obtained from the new proposed design.

- We propose a formal synthesis framework where MDPs induced from the developed model are employed in the automatic synthesis of adaptive droplet routing strategies that maximize the likelihood of successful bioassay execution by adapting to the microelectrode health information in real time.

- We develop a MEDA biochip simulator based on the SMG model to evaluate the proposed framework and present simulation results for six benchmark bioassays.

The rest of this chapter is organized as follows. Sec. 6.2 provides the notation used in this chapter, as well as the necessary background on bioprotocol synthesis.
on DMFBs, microelectrode degradation, and error recovery techniques for biochips. In Sec. 6.3, we present the new proposed microelectrode design and further explore degradation patterns in MEDA biochips. We also present experimental results on electrode degradation for fabricated PCB prototypes. Sec. 6.4 describes both the experimental and theoretical relationship between microelectrode degradation and the number of actuations. Next, we develop a stochastic game-based model for MEDA biochips in Sec. 6.5. Based on the developed model, we propose an adaptive routing strategy synthesis framework in Sec. 6.6. The experimental evaluation of the proposed framework is presented in Sec. 6.7. Finally, we draw our conclusions in Sec. 6.8.

6.2 Background

6.2.1 Notation

$\mathbb{N}_0$ denotes the set of non-negative integers. For $a, b \in \mathbb{N}_0$, $[a, b]$ denotes the discrete interval $\{x \mid x \in \mathbb{N}_0, a \leq x \leq b\}$. For a variable $x$, $x^{(k)}$ denotes its value at time $k \in \mathbb{N}_0$. We use bold symbols for matrices, e.g., $M = (M_{ij}) \in \mathbb{R}^{m \times n}$; here, $M_{ij}$ is the element in the $i$-th row and $j$-th column. The $i$-th element of a tuple or list $w$ is denoted by $w[i-1]$, $i \in \mathbb{N}_0$. For a set $A$, $\mathcal{P}(A)$ is its power set. For a random variable $x$, $x \sim U(x_1, x_2)$ denotes that $x$ follows a uniform distribution with the interval $[x_1, x_2] \subset \mathbb{R}$. We use N, S, E and W to denote north, south, east and west, respectively.

6.2.2 Bioprotocol Synthesis on DMFBs

In the DMFB synthesis flow [KWH14], a bioassay protocol with specified fluidic operations is first developed by biologists. Next, a synthesis tool maps fluidic operations to fluidic modules on the electrode array [CFZ10, OGB17]. Droplets need to be transported as part of the bioprotocol synthesized on the DMFB. Various droplet-routing
methods have been proposed in the literature [XC07, KWCD15, WLC+16, MZGB16], including techniques that are specific to MEDA [KLG+17, ZLC+18b]. However, these methods do not consider electrode degradation. Recently, reinforcement learning has been proposed to transport droplets in a reliable manner [LZ20]. However, this approach does not monitor the dynamic health condition of electrodes and therefore fluidic operations associated with degraded electrodes may still fail.

6.2.3 Error Recovery Techniques

For both DMFBs and MEDA biochips, the goal of error recovery techniques is to detect and further correct errors that occur during a bioassay execution [LCH12, LCH13b]. Techniques in literature can be categorized based on the type of corrective action into two groups: retrial and roll-back. In retrial-based recovery techniques, attempts are carried out to correct errors detected in a microfluidic operation without discarding the droplets involved. Depending on the error type, the droplets involved might get rerouted, reshaped, re-mixed, resplit, or undergo a combination of these corrective actions.

6.3 Microelectrode Cell Design

6.3.1 MEDA Biochips and Microelectrode Cell

A MEDA biochip is composed of an array of identical microelectrode cells (MCs) and a controller; the schematic of an MC is shown in Fig. 6.1(a). Each MC consists of a microelectrode, an electronic control circuit, and a sensing module. To carry out a bioassay on a MEDA biochip, a synthesis tool is first used to generate a schedule of fluidic operations, module placement, and droplet routes for the bioassay [ZLC+18b]. These are next mapped to a sequence of actuation patterns. The actuation patterns are sequentially shifted to the MC array through a scan chain. The MCs are actuated
Figure 6.1: Schematics of (a) the original and (b) the new proposed microelectrode circuit models for MEDA biochips.

based on the scanned-in data, and the corresponding fluidic operations are carried out based on EWOD. After MC actuation, all the MCs are set to the sensing mode to capture droplet locations. The sensing results are then scanned out as a bitstream. The process of shifting an actuation bitstream, MC actuation, droplet sensing, and shifting the sensing results is referred to as an operational cycle.

6.3.2 Microelectrode Degradation and Health Monitoring

MC sensing is used to detect droplet locations by measuring the capacitance between the top plate and bottom plate. The controller sets \( \text{ACT} = 0 \), \( \text{ACT}_b = 1 \), and \( \text{SEL} = 1 \); it also connects the top plate to ground. When this happens, transistors \( \text{T1}, \text{T2}, \text{and T4} \) are switched on while transistor \( \text{T3} \) is switched off, the bottom plate is connected to \( \text{VDD} \) (3.3 V) and the voltage of the bottom plate increases to 3.3 V. Next, the control circuit set \( \text{ACT}_b = 0 \), and transistors \( \text{T1}, \text{T3} \) and \( \text{T4} \) are switched on while transistor \( \text{T2} \) is switched off. As a result, the bottom plate is now connected to ground, and the voltage of the bottom plate decreases due to discharging. By applying a rising edge of MC-CLK at a preset time, a value of “0” or “1” can be stored in the DFF.

A major contributor to microelectrode degradation is the gradual trapping of
charge in the dielectric layer [DPB09, ZCGF03, LHS+05]. Thus, a proactive approach to ensure reliable fluidic operations is to estimate the degradation status of all microelectrodes in real-time and utilize only the healthy ones. To achieve this, we introduce a new MC design (Fig. 6.1(b)).

Charge trapping in the dielectric layer results in a higher capacitance between a degraded microelectrode and the top plate [BBB+03]; therefore, we can use capacitive sensing to detect degradation. An extra D flip-flop (DFF) is added to the MC design, and the rising edge of the CLK signal for this DFF is designed to arrive later than that of the other (original) DFF. For a healthy microelectrode, the 2-bit sensing result is “11”. If a microelectrode is partially degraded, the charging/discharging time is slightly less than that of a healthy microelectrode, and the original DFF is able to capture this difference by registering a different value compared to the newly added DFF (“0” versus “1”). If a microelectrode is completely degraded, the charging/discharging time is significantly lower than that of a healthy microelectrode, and both DFFs record “0”. This dynamic 2-bit sensing result provides the health-status information for the formal analysis model and synthesis method described in Sections 6.5–6.6.

We simulated the new MC design in HSPICE, using the macro-model for the extended-drain MOS transistors in the MC and a 350 nm library from a foundry; these models and parameters match the characteristics of fabricated biochips. We calculated the capacitance of microelectrodes using the parameters listed in Table 6.1. The simulation results are shown in Fig. 6.2, where the rising edge of the clock signal of the added DFF needs to be asserted 5 ns later than that of the original DFF. Note that MCs are fabricated using CMOS technology and CMOS-based frequency dividers in the range of GHz are available [RLY94]. Hence, by carefully controlling the rising edges of the two DFFs, we can dynamically measure the health status of a
Figure 6.2: Simulation results for the new microelectrode design.

microelectrode. The added DFF has no impact on the chip footprint because its area (~27 \( \mu \)m\(^2\)) is much less than the area of a microelectrode (2,500 \( \mu \)m\(^2\)) minus the area of the electronics underneath it (~88.2 \( \mu \)m\(^2\)) [LYL15]; the microfluidics part clearly dominates the overall area of the MC.

### 6.3.3 Degradation Patterns

In this subsection we examine the actuation patterns for which MC faults appear due to degradation. To this end, we design a set of experiments where we study the correlation between the number of actuations for two MCs and the Manhattan distance between them. In this set of experiments, we simulated the execution of three bioassays: ChIP, multiplex in vitro, and gene expression [LZPC20], on a 60 × 30 MEDA biochip. For each execution, the actuations of an MC at location \((i, j)\) is

Table 6.1: Notation used for the simulation.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>Area of a microelectrode</td>
<td>50×50 ( \mu )m(^2)</td>
</tr>
<tr>
<td>(\epsilon_o)</td>
<td>Silicon-oil permittivity</td>
<td>19 \times 10^{-12} (F/m)</td>
</tr>
<tr>
<td>(C_o)</td>
<td>Capacitance of healthy microelectrodes</td>
<td>(2.375 \times 10^{-15}) (F)</td>
</tr>
<tr>
<td>(C_{dl})</td>
<td>Capacitance of partially degraded microelectrodes</td>
<td>(2.380 \times 10^{-15}) (F)</td>
</tr>
<tr>
<td>(C_{d2})</td>
<td>Capacitance of completely degraded microelectrodes</td>
<td>(2.385 \times 10^{-15}) (F)</td>
</tr>
</tbody>
</table>
Figure 6.3: Simulation results for the correlation coefficient versus the Manhattan distance between two microelectrodes.

recorded as a Boolean vector $A_{ij} \in \mathbb{B}^N$, where $N$ is the number of operational cycles. The correlation coefficient between two MCs at locations $(i, j)$ and $(k, l)$ is defined as

$$\rho_{A_{ij}, A_{kl}} = \frac{\text{cov}(A_{ij}, A_{kl})}{\sigma_{A_{ij}} \sigma_{A_{kl}}},$$

where $\text{cov}(A_{ij}, A_{kl})$ is the covariance between the two vectors, and $\sigma_{A_{ij}}$ is the standard deviation of $A_{ij}$.

Fig. 6.3 shows the simulation results for droplet sizes $3 \times 3$, $4 \times 4$, $5 \times 5$, and $6 \times 6$, and Manhattan distances $d \in \{1, 2, 3, 4, 5\}$. The general trend shows an inverse correlation relationship between the distance between two MCs and the number of actuations for each. The correlation coefficient is lowest for droplet size $3 \times 3$, and increases as the droplet size increases. For the same droplet size, however, the correlation coefficient is insensitive to the executed bioassay. The results suggest that, in general, two adjacent MCs are more likely to have similar number of actuations during execution. This can be explained by the fact that in MEDA biochips, MCs are typically actuated in clusters at any given actuation cycle. This implies that faulty MCs are more likely to appear in clusters as well. Hence, it is imperative to evaluate the proposed framework against clustered faults, as we cover later in Sec. 6.7.
6.4 Microelectrode Degradation Model

Given the new microelectrode hardware design, we can monitor the health level of individual MCs in real-time. In this section, we study the impact of the MC number of actuations and its degradation and health levels. Using the obtained results, we show how to utilize the MC health level to estimate the EWOD force exerted by the MC on an adjacent droplet.

6.4.1 Experimental Assessment of Electrode Degradation

Previous work has shown that charge trapping in a dielectric layer follows an exponential model [MC83, ZCGF03, LSCD06, DCG+15]. To independently validate this claim, we design an experiment where we monitor electrode degradation in a PCB-based digital microfluidic biochip (DMFB), which manipulates droplets using the same EWOD principle as MEDA biochips.

The DMFB contains electrodes in three sizes, namely, 2×2 mm², 3×3 mm², and 4×4 mm²; see Fig. 6.4(a). Four reservoir modules are placed on two sides of the biochip, and the modules can dispense different reagent droplets. Each electrode can be controlled individually, and these control signals come from a control board underneath the DMFB. For the fabricated DMFB, the activation/de-activation status of each electrode is controlled by a high voltage relay (Part No. Panasonic AQW212). Each high-voltage relay IC is controlled by a configuration bit, and these configuration bits are stored in the register ICs (Part No. Texas Instrument SN74AHC595). The hardware setup used to operate the digital microfluidic biochip is shown in Fig. 6.4(b). A micro-computer (Part No. Raspberry Pi 4) on the left is used to generate control signals to the control board. We used a voltage source of 1.5 kHz and 200 Vpp to actuate the electrodes. To avoid inducing high current, a resistor $R = 1$ MΩ is placed in series between each electrode and the high-voltage source.
Figure 6.4: (a) The fabricated DMFB and (b) the experimental setup.

We developed an actuation sequence for the electrodes that leads to repeated fluidic operations on the biochip. When we execute the actuation sequence on the DMFB, each electrode is actuated for 1 s for hundreds of times. After executing the actuation sequence, we actuated an electrode and measured the charging times needed using an oscilloscope. Because the electrode and the top plate form a capacitor, and a resistor is placed in series with the electrode, the charging path is a simple RC circuit. The effective capacitance of an electrode can be derived using the equation

$$V_C(t) = V_{pp} \left( 1 - e^{-t/RC} \right),$$

where $C$ is the effective capacitance of the electrode, $V_C$ is the voltage of the electrode, and $t$ is time. The degradation results are shown in Fig. 6.5(a). The results show that the capacitance of an electrode grows linearly as we repeatedly actuate the electrode. This increase in the capacitance can be attributed to charge trapping in the dielectric layer.

In addition to charge trapping in the dielectric layer, electrode degradation can also result from residual charge, which happens when an electrode is excessively actuated \[\text{[HHCT]}\]. Excessive actuation of an electrode substantially increases the
amount of charge that accumulates in the actuated electrode. If the next electrode is in turn actuated whereas the present electrode has residual charge, the droplet may not be moved toward the next direction as expected. We design an experiment that is similar to the previous experiment, but in which each electrode is actuated for 5 s instead of 1 s. The degradation results are shown in Fig. 6.5(b). The results show that the capacitance of an electrode also grows linearly as we repeatedly actuate the electrode, but the growth is much faster than that of the previous experiment.

Note that the microelectrode size of a MEDA biochip is different from the electrode size of a conventional DMFB. Nevertheless, our experimental results using fabricated devices with different electrode sizes show that the relative force on the electrodes follows the same degradation trend, irrespective of electrode size. As a MEDA biochip manipulates discrete fluids using the same EWOD principle as DMFBs, the electrodes and relative force of a microelectrode on a MEDA biochip is also subject to the same degradation mechanism.

The work in [DCG+15] showed that the charge trapping phenomenon can be alleviated using a negative actuation voltage, i.e., AC actuation voltage. However, the mainstream commercialized DMFBs, such as Baebies, use DC actuation voltage (as in our work) because of simpler design, cheaper cost for the control circuit, and
less concerns about Joule heating [SPF04].

6.4.2 Microelectrode Health Model

Based on our experimental results for electrode degradation, in this section we develop a model that characterizes the relationship between the number of actuations $n$ and the microelectrode health level. The EWOD force exerted by a microelectrode $MC_{ij}$ (relative to the same EWOD force at full health) can be estimated as [ZLC+18b]

$$F_{ij}^{(n)} \approx \left( \frac{V_{ij}^{(n)}}{V_a} \right)^2,$$

(6.1)

where $V_{ij}^{(n)}$ is the actuation voltage on $MC_{ij}$ (potentially affected by the MC degradation), and $V_a$ is the nominal actuation voltage. By plugging our experimental results to (6.1), the impact of the microelectrode number of actuations and the relative EWOD force is shown in Fig. 6.6. We note that there is a good fit between the model and the measured data. The model fitting results show that the relationship between the number of actuation $n$ and the relative EWOD force $\tilde{F}_{ij}^{(n)}$ can be modeled as

$$\tilde{F}_{ij}^{(n)} \approx \tau^{2n/c},$$

(6.2)

where $\tau \in [0, 1]$ and $c \in \mathbb{R}$ are constants capturing the $MC_{ij}$ degradation rate. For example, the parameter values in Fig. 6.6 are $(\tau_2, c_2) = (0.556, 822.7)$, $(\tau_3, c_3) = (0.543, 805.5)$, and $(\tau_4, c_4) = (0.530, 788.4)$, where $R_{adj}^2 > 0.94$ for all curves.

To deduce the relation between the measured health level $H_{ij}$ and the number of actuations $n$, we define the degradation level of $MC_{ij}$, denoted by $D_{ij}$, as

$$D_{ij}^{(n)} = \frac{V_{ij}^{(n)}}{V_a} \approx \tau^{n/c} \in [0, 1],$$

(6.3)

where $D$ denotes the MC degradation matrix for the biochip. Moreover, given a number of bits $b \in \mathbb{N}_0$ for measuring the health level (recall that we use two bits in
the MC design in Sec. 6.3), we define $MC_{ij}$ health level, $H_{ij}$, as

$$H_{ij}^{(n)} = \lfloor 2^b \cdot D_{ij}^{(n)} \rfloor = \lfloor 2^b \cdot \tau^{n/c} \rfloor,$$

where $H$ denotes the MC health matrix for the biochip. Fig. 6.7 shows the impact of the number of actuations ($n$) on the observed $H_{ij}^{(n)}$ and the actual MC health $D_{ij}^{(n)}$ for various parameter configurations. The MC health exponentially decays as the number of actuations increase. The reliability model is valid for any general $b$, even though we use $b = 2$ for the results reported in this chapter.

6.5 MEDA Biochip Model

6.5.1 Droplet Model

Consider a MEDA biochip with $W \times H$ MCs. A Boolean matrix of size $W \times H$ could be used to capture which MCs are covered by a droplet. However, a typical MEDA biochip can have over $1,800$ MCs [ZLC+18b], which would result in a model with the state-space that is too large for formal synthesis.

Hence, we use the properties of microfluidic operations to develop a scalable droplet actuation model. Specifically, we adopt an approach where we model a droplet using the underlying actuation pattern since the droplet size, shape and location are...
Degradation $D_{ij}(\tau,c): (0.7,500)$ and $D_{ij}(\tau,c): (0.5,500)$; $D_{ij}(\tau,c): (0.7,200)$ and $D_{ij}(\tau,c): (0.5,200)$.

Figure 6.7: Impact of the no. of actuations $n$ on the actual degradation $D_{ij}$ and observed MC health $H_{ij}$ under different configurations.

tightly coupled with the used actuation pattern. For example, actuation patterns typically take a rectangular, fully-filled form where free-roaming of droplets (i.e., leaving them without actuation) is not allowed; and under- or over-actuation of droplets is of no use. Thus, by restricting the state-space to the actuation patterns of interest, we significantly reduce the model size; thus, enabling runtime formal strategy synthesis.

We use $U$ to indicate the biochip actuation matrix, where $U_{ij}^{(k)} \in \mathbb{B}$ indicates whether MC$_{ij}$ is actuated ($\top$) or not actuated ($\bot$) at time $k$. We model a droplet as a tuple $\delta = (x_a, y_a, x_b, y_b) \in \mathbb{N}_0^4$, where $(x_a, y_a)$ and $(x_b, y_b)$ are the coordinates of the lower-left and upper-right corners (i.e., $x_b \geq x_a$, $y_b \geq y_a$), and $U_{ij} = \top$ for all $(i,j) \in [x_a, x_b] \times [y_a, y_b]$. We use $\Delta \subset \mathbb{N}_0^4$ to denote the set of all possible droplets. We also use the center distance between two adjacent MCs as the unit length when describing droplet geometry. For a droplet $\delta = (x_a, y_a, x_b, y_b)$, the droplet width, height and area can be computed as $w = x_b - x_a + 1$, $h = y_b - y_a + 1$ and $A = wh$, respectively. We define a droplet’s aspect ratio as $AR = w/h$.

Example 8 (Droplet Model). Fig. 6.8 shows a droplet $\delta^{(k)} = (x_a, y_a, x_b, y_b)$ where
Figure 6.8: MEDA biochip segment with droplet $\delta$ at time $k$.

$x_a = 3$, $y_a = 2$, $x_b = 7$ and $y_b = 5$, i.e., $\delta^{(k)} = (3, 2, 7, 5)$. The droplet’s width, height, area and aspect ratio are $w = 5$, $h = 4$, $A = 20$ and $AR = 5/4$, respectively. The biochip actuation matrix satisfies that

$$U_{ij}^{(k)} = \begin{cases} \top & \forall (i, j) \in [3, 7] \times [2, 5], \\ \bot & \text{otherwise}. \end{cases}$$

### 6.5.2 Actuation Model

MEDA biochips support three classes of droplet manipulations: cardinal movement, ordinal movement, and shape morphing. We define the set of microfluidic actions as $A = A_d \cup A_{dd} \cup A_{dd'} \cup A_{\downarrow} \cup A_{\uparrow}$, where

- $A_d = \{a_N, a_S, a_E, a_W\}$ are single-step movements in the cardinal directions,
- $A_{dd} = \{a_{NN}, a_{SS}, a_{EE}, a_{WW}\}$ are double-step movements in the cardinal directions,
- $A_{dd'} = \{a_{NE}, a_{NW}, a_{SE}, a_{SW}\}$ are movements in the ordinal directions,
- $A_{\downarrow} = \{a_{\downarrow NE}, a_{\downarrow NW}, a_{\downarrow SE}, a_{\downarrow SW}\}$ are morphing transformations that aim to increase droplet width, and
Figure 6.9: Effect of microfluidic actions on droplets. Blue (solid) markers show initial locations. Red (dashed) markers show locations after successful execution.

- $\mathcal{A}_\uparrow = \{a_{\uparrow \text{NE}}, a_{\uparrow \text{NW}}, a_{\uparrow \text{SE}}, a_{\uparrow \text{SW}}\}$ are morphing transformations that aim to increase droplet height.

Single- and double-step movements aim to move the droplet a distance of a single and double units (i.e., one or two MCs in one cycle) in one of the cardinal directions, respectively. While $\mathcal{A}_d$, $\mathcal{A}_{dd}$, and $\mathcal{A}_{dd'}$ actions do not change droplet shape, $\mathcal{A}_i$ and $\mathcal{A}_\uparrow$ actions aim to change the droplet aspect ratio. Fig. 6.9 illustrates the microfluidic actions and their impact on droplets.

For an action $a \in \mathcal{A}$ and a droplet $\delta^{(k)} \in \Delta$, at time $k$, it holds that $\delta^{(k+1)} = a(\delta^{(k)})$. We define the frontier-set function $Fr(\delta; a, d)$, capturing the subset of MCs that affect the movement of a droplet $\delta$ in direction $d \in \{\text{N, S, E, W}\}$ due to action $a$ – i.e., $Fr(\bullet; a, d) : \Delta \rightarrow \mathcal{P}(\mathbb{N}_0^2)$. Table 6.2 shows the list of microfluidic actions and the respective frontier sets. Note that $Fr$ is not defined for ordinal directions.

Example 9 (Frontier Set). Fig. 6.10 shows a droplet $\delta = (3, 2, 7, 5)$ actuated under $a_{\text{NE}}$ to initiate a movement in the NE direction. The set of MCs pulling the droplet to the east and north directions are $Fr(\delta; a_{\text{NE}}, \text{E}) = [8, 8] \times [3, 6]$ and $Fr(\delta; a_{\text{NE}}, \text{N}) = [4, 8] \times [6, 6]$, respectively.

The degradation level of the MCs used in the movement (i.e., the MCs in the frontier set) impacts the EWOD driving force. Thus, a microfluidic action may not always result in the intended droplet movement. From (6.1), the relative EWOD
force exerted on δ in direction $d \in \{N, S, E, W\}$ by action $a$ can be estimated as

$$\bar{F}(\delta; a, d) = \sum_{(i,j)\in Fr(\delta; a, d)} \bar{F}_{ij} = \sum_{(i,j)\in Fr(\delta; a, d)} x^{2m_{ij}/c}.$$  

As a larger EWOD force is more likely to move the droplet in the intended direction, the probability of whether an action $a$ successfully moves droplet $\delta$ in direction $d$ is a function of the degradation level of the MCs in $Fr(\delta; a, d)$. Let $\Sigma_d = \{N, S, E, W, \epsilon\}$ be the event space of executing $a_d \in \mathcal{A}_d$, where $\epsilon$ is the event of the droplet not moving. Assuming that all MCs in $Fr(\delta; a, d)$ equally contribute to the movement,
the probability of an event $E$ can be expressed as

$$p(E \mid \delta, a_d) = \begin{cases} 
\frac{\bar{F}(\delta; a_d, d)}{|Fr(\delta; a_d, d)|} & E = d, \\
1 - \frac{\bar{F}(\delta; a_d, d)}{|Fr(\delta; a_d, d)|} & E = \epsilon, \\
0 & \text{otherwise}.
\end{cases}$$

If $a_d$ is successfully executed on $\delta^{(k)}$, the resulting droplet location is $\delta^{(k+1)} = a_d(\delta^{(k)})$. Otherwise, the droplet location remains unchanged, i.e., $\delta^{(k+1)} = \delta^{(k)}$. In case of a double-step movement $a_{dd} \in A_{dd}$, the probability that the second step is successful is conditioned on the success of the first step. Hence,

$$p(E \mid \delta, a_{dd}) = \begin{cases} 
\frac{\bar{F}(\delta; a_d, d)}{|Fr(\delta; a_d, d)|} \cdot \frac{\bar{F}(\delta'; a_d, d)}{|Fr(\delta'; a_d, d)|} & E = dd, \\
\frac{\bar{F}(\delta; a_d, d)}{|Fr(\delta; a_d, d)|} \cdot \left(1 - \frac{\bar{F}(\delta'; a_d, d)}{|Fr(\delta'; a_d, d)|}\right) & E = d, \\
1 - \frac{\bar{F}(\delta; a_d, d)}{|Fr(\delta; a_d, d)|} & E = \epsilon, \\
0 & \text{otherwise}.
\end{cases}$$

where $\delta' = a_d(\delta)$ is the droplet location shifted by one step in the same direction. Similarly, the possible outcomes of an ordinal movement $a_{dd'} \in A_{dd'}$ are mov-
Figure 6.11: Model for droplet $\delta = \langle x_a, y_a, x_b, y_b \rangle$ showing two microfluidic actions $a_\text{E}$ and $a_\text{NE}$ and their transitions, where $q(E | \delta, a) = 1 - p(E | \delta, a)$.

Continuing the running example, suppose that the frontier set MCs have degradation values $D_{(8,3,6)} = (0.6, 0.5, 0.8, 0.9)$ and $D_{(4,8,6)} = (0.9, 0.4, 0.9, 0.7, 0.9)$ as shown in Fig. 6.10. The probability of successfully moving in two directions, one direction, or none, captured by the event space $\Sigma_{dd'} = \{\text{NE, NW, SE, SW}\} \cup \Sigma_d$. The probability of each event can be expressed as

$$p(E | \delta, a_{dd'}) = \begin{cases} 
\frac{\bar{F}(\delta; a_{dd'}, d')}{|F_r(\delta; a_{dd'}, d')|} & \text{if } E = dd', \\
\frac{\bar{F}(\delta; a_{dd'}, d')}{|F_r(\delta; a_{dd'}, d')|} & \text{if } E = d, \\
\frac{1 - F_r(\delta; a_{dd'}, d')}{|F_r(\delta; a_{dd'}, d')|} & \text{if } E = d', \\
\frac{1 - F_r(\delta; a_{dd'}, d')}{|F_r(\delta; a_{dd'}, d')|} & \text{if } E = \epsilon, \\
0 & \text{otherwise.}
\end{cases}$$

Fig. 6.11 illustrates part of the droplet model, where the actions $a_\text{N}$ and $a_\text{NE}$ and their probabilistic transitions are displayed. If $a_\text{NE}$ is executed, the droplet can successfully move east to $\delta_\text{E}(x^+_a, y_a, x^+_b, y_b)$ with probability $p(E | \delta, a_\text{NE})$, or remain at $\delta(x_a, y_a, x_b, y_b)$ with probability $q(E | \delta, a_\text{NE})$, where $q(E | \delta, a) = 1 - p(E | \delta, a)$. 

Example 10 (Transition Probabilities). Continuing the running example, suppose that the frontier set MCs have degradation values $D_{(8,3,6)} = (0.6, 0.5, 0.8, 0.9)$ and $D_{(4,8,6)} = (0.9, 0.4, 0.9, 0.7, 0.9)$ as shown in Fig. 6.10. The probability of successfully...
moving the droplet in the NE direction is

\[
p(\text{NE} \mid \delta, a_{\text{NE}}) = \frac{.9 + .4 + .7 + .9}{5} \cdot \frac{.6 + .5 + .8 + .9}{4} = 0.532.
\]

Similarly, the probability of moving north is \(p(\text{N} \mid \delta, a_{\text{NE}}) = 0.168\), and the probability of moving east is \(p(\text{E} \mid \delta, a_{\text{NE}}) = 0.228\).

The morphing actions in \(A_\downarrow\) and \(A_\uparrow\) aim to decrease and increase the droplet’s height, respectively. The difference amongst the actions of each set lies in the direction towards which the droplet’s width and height are increased or decreased (see Fig. 6.9). For instance, \(a_{\text{NE}}\) decreases the droplet’s height by increasing its width towards the north-east direction. In contrast, \(a_{\text{SW}}\) decreases the droplet’s width by increasing its height towards the south-west direction. The probability of a morphing action being successful mainly depends on the set of MCs responsible for pulling the droplet. The frontier sets of morphing actions is listed in Table 6.2.

In practice, the degree to which a droplet can be successfully morphed depends on its current size and shape. For instance, droplet aspect ratio may not go above 2/1 or below 1/2 to avoid unintentional splitting of the droplet. To model such constraints, we use guards on actions. A guard on action \(a\) is a Boolean expression \(g\) that represent a necessary condition for \(a\) to be enabled. Let \([r/1, 1/r]\), \(r \geq 1\), be the allowed range for droplet aspect ratio. For shape morphing actions \(a_\uparrow \in A_\uparrow\) and \(a_\downarrow \in A_\downarrow\), we define two guards

\[
g_\uparrow: \frac{y_b - y_a + 2}{x_b - x_a} \leq r \quad \text{and} \quad g_\downarrow: \frac{x_b - x_a + 2}{y_b - y_a} \leq r,
\]

respectively. For example, for a maximum aspect ratio \(r = 3/2\) and a droplet \(\delta = (3, 2, 7, 5)\), \(g_\uparrow = 1\), while \(g_\downarrow = 0\). That is, \(a_\uparrow\) is enabled while \(a_\downarrow\) is disabled.

The model also supports moving two steps in a cardinal direction. Practically, a droplet can be reliably moved a distance no longer than half its length in one cycle.
Hence, we impose a guard on double-step movements such that they are only enabled if such condition is satisfied. That is, double-step movements in the north or south directions are enabled for droplets with height \( h \geq 4 \), while double-step movements in the east or west directions are enabled for droplets with width \( w \geq 4 \). Consequently, the guards \( g_{NN}, g_{SS} : y_b - y_a + 1 \geq 4 \), and \( g_{EE}, g_{WW} : x_b - x_a + 1 \geq 4 \) are defined for \( a_{NN}, a_{SS}, a_{EE} \) and \( a_{WW} \), respectively.

### 6.5.3 MEDA Biochip Model

While a droplet \( \delta \) can be manipulated via various microfluidic actions as discussed earlier in Sec. 6.5.2, the action outcomes are probabilistic. Moreover, such outcomes also depend on the MC health matrix \( H \). To accommodate the controller’s choices and the probabilistic behaviors, we model the MEDA biochip using the stochastic-multiplayer games (SMGs) formalism.

Intuitively, in the MEDA SMG, denoted by \( \mathcal{G} \), the game state is a triplet \( s = (\delta, H, \lambda) \), where \( \lambda \in \{I, II\} \) is the current player. Let \( \mathcal{H} = [0, 2^h - 1]^{\text{W} \times \text{H}} \) be the set of all possible \( H \). The state-space \( S \subseteq \Delta \times \mathcal{H} \times \{I, II\} \) captures all possible droplet locations, health states, and players’ turn. A droplet controller constitutes the first player, I, with an action set \( \mathcal{A}_1 = \mathcal{A} \) (see Sec. 6.5.2). The biochip degradation constitutes the second player, II, with an action set \( \mathcal{A}_2 = \mathcal{P}(\{a_{ij} | 1 \leq i \leq \text{W}, 1 \leq j \leq \text{H}\}) \), where \( a_{ij} \) is the action of reducing \( H_{ij} \) by one. Note that II can simultaneously take multiple actions (i.e., degrade multiple MCs at the same time). The initial state \( s_0 = (\delta(0), H(0), I) \) defines the initial droplet location and health matrix. Finally, the MEDA SMG is formally defined as the tuple \( \mathcal{G} = (S, \mathcal{A}_1 \cup \mathcal{A}_2, \gamma, s_0) \), where \( \gamma : S \times \mathcal{A}_1 \cup \mathcal{A}_2 \times S \to [0, 1] \) is the transition probability function defined using the transition probabilities previously described in Sec. 6.5.2.

Abstracting the biochip degradation as a player with nondeterministic actions
serves two purposes. First, it allows for modeling a wide range of assumptions regarding the degradation behavior and fault-injection modes. Second, it enables the usage of two levels of model fidelity, one suitable for routing strategy synthesis, and the other for experimental simulations to validate the former.

Since the health matrix $H$ is visible to the droplet controller, the resulting SMG is a full-information game. Hence, $G$ can be used to synthesize droplet routing strategies as described in Sec. 6.6. For simulation, the same model is used, except that the health matrix $H$ is substituted with the degradation matrix $D$. The resulting SMG is an incomplete-information game since the droplet controller cannot observe $D$. Further details on the simulation environment are covered in Sec. 6.7.

## 6.6 Synthesis Framework

A typical bioassay is comprised of a series of droplet transportation and microfluidic operations that can be carried out on a MEDA biochip. In this section, we introduce a framework for adaptive strategy synthesis using the SMG-based MEDA model. We first show how to map various microfluidic operations into a set of droplet routing problems, called routing jobs. Next, we explore different ways to formalize requirements for routing strategies, and show how to use such requirements along with the SMG-based MEDA model to formally synthesize the routing strategies. Finally, we compare offline and online strategy synthesis methods.

### 6.6.1 Sequence Graphs and Microfluidic Operations

In this chapter, we assume that a given bioassay is represented as a sequencing graph (SG), describing the list of microfluidic operations (MOs). We also assume that the SG is preprocessed by a planner that determines the dependencies and module placements of MOs, resulting in an MO list (e.g., see ZLC+18). Each item in the
Table 6.3: List of microfluidic operations and the corresponding number of input and output droplets.

<table>
<thead>
<tr>
<th>MO Type</th>
<th>Description</th>
<th>#Droplets (In, Out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dis</td>
<td>Dispense a droplet (enter biochip)</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>out/dsc</td>
<td>Output/Discard a droplet (exit biochip)</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>mix</td>
<td>Mix two droplets into one</td>
<td>(2, 1)</td>
</tr>
<tr>
<td>spt</td>
<td>Split a droplet into two</td>
<td>(1, 2)</td>
</tr>
<tr>
<td>dlt</td>
<td>Dilute a droplet using another</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>mag</td>
<td>Magnetically sense a droplet</td>
<td>(1, 1)</td>
</tr>
</tbody>
</table>

Example 11 (SG and MO List). Fig. 6.12 shows a simple sequence graph of four microfluidic operations. M1 and M2 are dispensing operations for 4 × 4 droplets (16, 1, 19, 4) and (16, 27, 19, 30), respectively, they have no predecessor operations. The center location for M1, for instance, is computed as \([(16 + 19)/2, (1 + 4)/2] = (17.5, 2.5)\). M3 is a mixing operation between the two droplets from M1 and M2.
6.6.2 Routing Jobs

To synthesize routing strategies, each MO is decomposed into a set of single-droplet routing problem, called a routing job (RJ). Each RJ stores the information necessary to synthesize a routing strategy for a single droplet. Formally, an RJ is a tuple \( RJ = (\delta_s, \delta_g, \delta_h) \), where \( \delta_s \) is the droplet start location; \( \delta_g \) is the droplet goal location; and \( \delta_h \) is the hazard bounds for the routing job, defining the area within which the droplet is allowed to move. We design an RJ helper function that performs the aforementioned decomposition. Algorithm 7 summarizes the procedure that the RJ helper function follows to convert a given MO into a set of single-droplet routing jobs.

For dispensing operations, \( \delta_s = (0, 0, 0, 0) \) since the initial location of the droplet is outside the biochip, while \( \delta_g \) is where the droplet is dispensed. Note that the droplet size and shape are inferred from \( \delta_g \) (see Sec. 6.5.1). Since the dispensing operation is straightforward, the routing strategy is generated as a movement perpendicular to the edge from which the droplet is dispensed. For output and discard operations, however, routing is required for one droplet. The droplet’s initial location is determined by the end location of the preceding MO, i.e., \( \delta_s = \delta_{g_{\text{pre}[0]}} \). The end location \( \delta_g \) is the last on-chip location before the droplet exits the biochip through one of the four edges.

In a mixing operation, two droplets are routed from two different locations, \( \delta_{s_0} \) and \( \delta_{s_1} \), to a single destination \( \delta_g \). In contrast, a splitting operation features two droplets that are routed from the same location \( \delta_s \) to two different locations, \( \delta_{g_0} \) and \( \delta_{g_1} \), that can be specified directly by the MO, or can be automatically placed at locations \( \text{loc}_{\text{pre}[0]} + \text{disp} \) and \( \text{loc}_{\text{pre}[0]} - \text{disp} \), respectively, where \( \text{disp} \) is some displacement from the location where the splitting occurs. A dilution operation comprises mixing two droplets that start at \( \delta_{s_0} \) and \( \delta_{s_1} \), followed by a splitting operation, resulting in \( \delta_{g_0} \) and \( \delta_{g_1} \).

To compute the values of \( \delta_s \) and \( \delta_g \), the droplet size needs to be known. For
droplet-generating operations (i.e., dispensing), the desired size is already specified by the MO. For all other operations, the RJ helper computes the respective droplet sizes by first computing the droplet area and then obtain the droplet length and width that provide the minimum error in the computed area while satisfying the condition $|w - h| \leq 1$.

The hazard bounds $\delta_h$ represent the rectangular area within which the routing can occur, while the droplet is forbidden from moving outside such area. Computing $\delta_h$ depends on the provided scheduler and resource allocation. In this work, we assume that the hazard bounds are computed as the rectangular area including both $\delta_s$ and $\delta_g$, in addition to a safety margin of 3 MCs from each of the four sides to prevent accidental merging of droplets. That is, for $\delta_s = (x_a, y_a, x_b, y_b)$ and $\delta_g = (x'_a, y'_a, x'_b, y'_b)$, the hazard bounds are computed as $\delta_h = ZONE(\delta_s, \delta_g)$ where

\[
ZONE(\delta_s, \delta_g) := \left( \min(x_a - 3, x'_a - 3, 1), \min(y_a - 3, y'_a - 3, 1), \right.
\]
\[
\left. \max(x_a + 3, x'_a + 3, W), \max(y_a + 3, y'_a + 3, H) \right).
\]

More advanced computations of the hazard bounds can incorporate other information such as the droplet size and the number of concurrent operations.

**Example 12** (RJ Helper). Continuing the previous example, let the dispensed droplets have size $4 \times 4$. Table 6.4 shows the list of MOs and the associated RJs generated by the RJ helper for a MEDA biochip of size $60 \times 30$. Notice that $M3$ is decomposed into two routing jobs, RJ3.0 and RJ3.1, with the same goal location. The mixing operation results in a droplet area $A = 32$, which is approximated to a $6 \times 5$ actuation pattern. The location at which $M4$ occurs is centered at $(40.5, 15.5)$, and hence the target location for the corresponding routing job is estimated as $(38, 14, 43, 18)$. 

140
Table 6.4: Example for converting MOs to RJs for a MEDA biochip of size $W \times H = 60 \times 30$.

<table>
<thead>
<tr>
<th>MO type</th>
<th>pre</th>
<th>loc</th>
<th>Size ($w \times h$)</th>
<th>Size Error</th>
<th>RJ</th>
<th>Start Location $\delta_s$</th>
<th>Goal Location $\delta_g$</th>
<th>Hazard Bounds $\delta_h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>dis</td>
<td>$\emptyset$</td>
<td>(17.5, 2.5)</td>
<td>16 (4 x 4)</td>
<td>0.0%</td>
<td>RJ1.0 (00, 00, 00)</td>
<td>(16, 01, 19, 04)</td>
<td>(13, 01, 22, 07)</td>
</tr>
<tr>
<td>M2</td>
<td>dis</td>
<td>$\emptyset$</td>
<td>(17.5, 28.5)</td>
<td>16 (4 x 4)</td>
<td>0.0%</td>
<td>RJ2.0 (00, 00, 00)</td>
<td>(16, 27, 19, 30)</td>
<td>(13, 24, 22, 30)</td>
</tr>
<tr>
<td>M3</td>
<td>mix</td>
<td>{M1, M2}</td>
<td>(10.5, 15.5)</td>
<td>16 (4 x 4)</td>
<td>0.0%</td>
<td>RJ3.0 (01, 19, 04)</td>
<td>(09, 14, 12, 17)</td>
<td>(06, 01, 22, 20)</td>
</tr>
<tr>
<td>M4</td>
<td>mag</td>
<td>{M3}</td>
<td>(40.5, 15.5)</td>
<td>32 (6 x 5)</td>
<td>6.3%</td>
<td>RJ1.0 (08, 14, 13, 18)</td>
<td>(38, 14, 43, 18)</td>
<td>(05, 11, 46, 21)</td>
</tr>
</tbody>
</table>

Algorithm 7: MO-to-RJ Helper Procedure

1 Function MO_TO_RJ(MO)
2 Data: Microfluidic operation MO = (type, pre, loc)
3 Result: Routing job list (RJ)
4 switch type do
5     case dis do RJ[0] ← (0, loc[0], loc[0])
6     case out, dsc, mag do RJ[0] ← (pre[0], loc[0], loc[0])
7     case mix do
8         RJ[0] ← ($\delta_{pre[0]}$, loc[0], ZONE(pre[0], loc[0]))
9         RJ[1] ← ($\delta_{pre[1]}$, loc[0], ZONE(pre[0], loc[0]))
10    case spt do
11       RJ[0] ← ($\delta_{pre[0]}$, loc[0], ZONE(pre[0], loc[0]))
12       RJ[1] ← ($\delta_{pre[1]}$, loc[1], ZONE(pre[0], loc[1]))
13    case dlt do
14       RJ[0] ← ($\delta_{pre[0]}$, loc[0], ZONE(pre[0], loc[0]))
15       RJ[1] ← ($\delta_{pre[1]}$, loc[0], ZONE(pre[1], loc[0]))
16       RJ[2] ← ($\delta_{RJ[0]}$, loc[0], ZONE(pre[0], loc[0]))
17 return (RJ)

6.6.3 Routing Strategy Synthesis

Let $G = (S, A_1 \cup A_2, \gamma, s_0)$ be the MEDA biochip model, RJ = ($\delta_s, \delta_g, \delta_h$) be a routing job, and $H$ be the current health matrix. A droplet routing strategy is a mapping from $I$ states to the set of microfluidic actions $A_1 = A$ (see Sec. 6.5.2), denoted by $\sigma: S_1 \rightarrow A_1$. The routing strategy problem is concerned with finding a strategy $\sigma$ such that it satisfies a set of requirements. For the routing job RJ, the basic requirement is to for the droplet to eventually reach the goal location $\delta_g$, i.e., a state $s = (\delta_g, H, \lambda)$, while avoiding the hazard bounds $\delta_h$.

To formalize this notion, we define two state labels, goal and hazard, that mark
goal and hazard states, respectively. Each label is a propositional formula over state-space variables that can be evaluated as true or false at a given state. For a routing job where \( \delta_g = (x_{ag}, y_{ag}, x_{bg}, y_{bg}) \) and \( \delta_h = (x_{ah}, y_{ah}, x_{bh}, y_{bh}) \), the two labels are defined as

- **goal**: \((x_a \geq x_{ag}) \land (y_a \geq y_{ag}) \land (x_b \leq x_{bg}) \land (y_b \leq y_{bg})\),
- **hazard**: \((x_a < x_{ah}) \lor (y_a < y_{ah}) \lor (x_b > x_{bh}) \lor (y_b > y_{bh})\).

Note that **goal** utilizes inequalities rather than \( \delta == \delta_g \) to allow for a less restrictive specification of the goal location. For instance, a \( 3 \times 3 \) droplet may have a \( 5 \times 5 \) goal location to indicate that the goal is to reach anywhere within the specified location. Using **goal** and **hazard**, the linear temporal logic (LTL) formula \( \varphi: \square (\neg \text{hazard}) \land \diamond \text{goal} \) captures all possible executions that satisfy the routing requirement, where the temporal operator \( \diamond \) means *eventually true*, \( \square \) *always true*\(^1\).

Next, the LTL formula can be used to form a synthesis query. In this work, we explore the usage of two types of queries, namely, probabilistic and reward-based queries. The probabilistic query \( \phi_p: P_{\text{max}} = \exists [\square (\neg \text{hazard}) \land \diamond \text{goal}] \) can be used to synthesize a strategy that maximizes the probability of satisfying \( \varphi \). By feeding \( G \) and \( \phi_p \) into a model checker, both an optimal strategy \( \pi_p \) (if existing) and the corresponding probability \( p_{\text{max}} \) are obtained.

Similarly, a reward-based query can be used for routing strategy synthesis. To this end, a reward function \( r: S_1 \cup A_1 \rightarrow \mathbb{R}_{\geq 0} \) is defined to reflect the reward associated with states and/or actions. For example, the reward function

\[
r_k(a) = \begin{cases} 
1 & \text{if } a \in A_1, \\
0 & \text{otherwise}
\end{cases}
\]

tracks the number of cycles (i.e., microfluidic actions) required to reach the goal

---

\(^1\)More on LTL formulas can be found in [BKL08].
Algorithm 8: Routing Strategy Synthesis Procedure

1 Function SYNTH(RJ, H)
2     Data: RJ = (δs, δg, δh); health matrix H
3     Result: Strategy σ : A → A; expected completion time k ∈ R
4     Initialize model G using H
5     Let goal: (xa ≥ xag) ∧ (ya ≥ yag) ∧ (xb ≤ xbg) ∧ (yb ≤ ybg)
6     Let hazard: (xa < xah) ∨ (ya < yah) ∨ (xb > xbh) ∨ (yb > ybh)
7     Let ϕ: Rmin=? [□ (¬hazard) ∧ ◇goal]
8     (σ, k) ← PRISMg (G, ϕ, δs)
9     return (σ, k)

The computational complexity of the synthesizer is \( O((W-w)(H-h)|A|) \), where \( w \times h \) is the droplet size. To reduce the computational complexity, we apply partial order reduction to each routing job individually. First, for each routing job, the state-space is limited to the locations within the hazard bounds \( δ_h = (x_{ah}, y_{ah}, x_{bh}, y_{bh}) \). We
will use $\Delta_h \subseteq \Delta$ to denote the set of all possible droplet locations within $\delta_h$. In the span of one routing job, the number of actuations required for a given MC is relatively small. Consequently, we can assume that the change in $H$ is insignificant after a single action, rendering the order in which the degradation actions $A_2$ occur irrelevant to the synthesis problem. By fixing $H$ to the initial value $H_0$, the SMG is reduced to the Markov decision process (MDP) $G_{\text{RJ}} = (\hat{S}, A_1, \hat{\gamma}, \hat{s}_0)$ where $\hat{S} \subseteq \Delta_h \times \{H_0\} \times \{I\}$, $\hat{\gamma}$ is defined by the rule

$$\hat{\gamma}((\delta, H_0, I), a, (\delta', H_0, I)) = \gamma((\delta, H_0, I), a, (\delta', H_0, II))$$

and $\hat{s}_0 = (\delta_s, H_0, I)$. In this way, the computational complexity is reduced to

$$O((w_h - x) \cdot (h_h - y) \cdot |A_1|),$$

where $w_h = x_{bh} - x_{ah} + 1$ and $h_h = y_{bh} - y_{ah} + 1$. The impact of the RJ area and droplet size on the runtime performance of the synthesis algorithm is further investigated later in Sec. 6.7.4.

### 6.6.4 Adaptive Routing Framework

The overall data-flow diagram of the proposed adaptive routing framework is shown in Fig. 6.13. The planner provides the list of MOs to the helper function to generate the list of routing jobs. Next, the synthesizer utilizes routing jobs $\text{RJ}$ and the current health matrix $H$ to synthesize the corresponding routing strategies. The low-level controller combines such strategies to generate the actuation pattern $S$ at any time.

To synthesize a routing strategy, the synthesizer requires the health matrix value $H$. Since the degradation characteristics of the MCs differ from one biochip to another, the exact value of $H$ is unknown until the current routing job is due for execution. In an online scheduling scheme, strategies are synthesized on-demand and whenever $H$ value is available. While such scheme is straightforward to implement,
on-demand strategy synthesis introduces delays between subsequent microfluidic operations, which can be undesirable for time-sensitive bioassays. On the other hand, in an offline scheduling scheme, strategies are synthesized offline based on a range of expected values of $H$. In runtime, the scheduler retrieves the pre-synthesized strategy that corresponds to the actual value of $H$. While this scheme avoids synthesis delays, synthesizing, storing and retrieving strategies for all possible ranges of $H$ is practically intractable. For instance, for a $20 \times 20$ biochip with $b = 2$ and a $4 \times 4$ droplet, the number of states is $|\hat{S}| > 10^{77}$.

To overcome this problem, we adopt a hybrid scheduling scheme that takes advantage of both the online and offline scheduling. In this scheme, a library of presynthesized strategies is first created offline for a range of droplet sizes and assuming no degradation. In runtime, the scheduler checks whether a strategy is available for the current $H$ and retrieves it. Next, if a change in $H$ value (i.e., degradation) is detected during the execution of a routing strategy, the scheduler relays the new value to the synthesizer to asynchronously resynthesize new strategies. Once the new strategies are available, they replace the previous ones since the value of a $H$ element cannot regain its previous value once changed.

Algorithm 9 summarizes the procedure for the hybrid scheduler. First, the helper function populates the list of routing jobs for the given MO list, setting their statuses
Algorithm 9: Hybrid Scheduler Procedure

Input: MO list (MO); offline strategy library Lib

1. foreach MO ∈ (MO) do
2.   (RJ)MO ← Mo_To_RJ(MO), stateMO ← init
3. k ← 0
4. while ∃MO ∈ (MO) s.t. stateMO ≠ done do
5.   k++, U ← ⊥, Read H(k)
6.   Read Y(k) and update the droplet location δ(k) of each MO
7.   foreach MO ∈ (MO) do switch stateMO
8.   case init do
9.     if ∀MO’ ∈ (pre)MO : stateMO’ == done then
10.    stateMO ← active
11.     foreach RJ ∈ (RJ)MO : do
12.        if a strategy for RJ exists in Lib then
13.           (σRJ, kRJ) ← Lib(RJ)
14.        else
15.           (σRJ, kRJ) ← SYNTH(RJ, H(k))
16.           Add (σRJ, kRJ) to Lib
17.   case active do
18.     if ∀RJ ∈ (RJ)MO : δg == δ(k) then
19.       stateMO ← done
20.     else a ← σ(δ), U(a(δ)) ← ⊤
21. Apply U

In each cycle k, the scheduler reads the current sensor measurements Y(k) and health matrix H(k). Next, the scheduler checks if any MOs are ready for execution by confirming whether their corresponding predecessors are done and, if any, forwards the corresponding RJs to the synthesizer to retrieve their routing strategies. For an active MO, the optimal action a is retrieved from the current strategy (i.e., a = σ(δ)), and the corresponding MCs are set to be actuated (i.e., U(a(δ)) = ⊤). Finally, the actuation matrix U is applied to the biochip, and the process is repeated until all MOs finish execution.
6.7 Experimental Evaluation

6.7.1 Experimental Setup

We implemented the online scheduler, helper, synthesizer, and a MEDA biochip simulator in MATLAB. The synthesizer automatically generated and passed routing jobs to PRISM-games [KNPS20] to obtain routing strategies. Results were obtained on an Intel Core i7 2.6 GHz CPU with 16 GB RAM.

The MEDA biochip simulator enables configuring the biochip width $W$ and height $H$, in addition to the MC degradation behavior. In general, a microelectrode $MC_{ij}$ is assigned uniformly sampled degradation parameters $c_{ij} \sim U(c_1, c_2)$ and $\tau_{ij} \sim U(\tau_1, \tau_2)$ (see Sec. 6.4.2). The simulator generates two types of MCs — normal and faulty — by sampling the degradation constants from two different uniform distributions, with the percentage of faulty MCs being configurable. The way in which faulty MCs are placed across the biochip depends on the fault-injection mode selected for the experiment. In uniform fault-injection mode, the faulty MCs are randomly placed. In clustered fault-injection mode, however, clusters of $2 \times 2$ faulty MCs are randomly placed.

We simulated six benchmark bioassays in our experiments: (i) Master-Mix, (ii) CEP, (iii) Serial Dilution [EZL+17a], (iv) nucleosome immunoprecipitation (NuIP) [LZ20], (v) COVID-RAT, and (vi) COVID-PCR. The CEP bioprotocol comprises three bioassays, namely, cell lysis, mRNA extraction, and mRNA purification. The NuIP bioprotocol is used for studying the epigenetic relationship between DNA and its supporting proteins [CoI10]. Two COVID-19 tests, namely PCR-based and rapid antigen-based, are widely used to detect the presence of the SARS-CoV-2 virus or the body’s response to infection [Gug20]. PCR-based test, which is more accurate, detects small amounts of viral genetic material. Rapid antigen test, which is only
effective for the first week of infection, detects the presence of viral proteins.

Fig. 6.14 shows the control flow of the simulation environment used in this section. First, the simulator instantiates a MEDA biochip according to the biochip configuration, the degradation parameters and the fault-injection mode selected for the experiment. Next, the online scheduler creates a new bioassay execution request based on the bioassay selected for the experiment, and further uses the helper to generate the list of routing jobs. Once the execution starts, at each cycle, the scheduler checks whether any MOs are ready for execution, reads the current health status from the biochip simulator. The synthesizer uses this information to generate the corresponding models and synthesis queries, passing them to the model checker to obtain the synthesized strategies. Subsequently, the controller uses the current biochip state and the synthesized strategies to populate the optimal action for each droplet, aggregate the actuation patterns for the current control cycle, and further pass the control matrix $U$ to the biochip simulator. Using the current control matrix $U$, both the number of actuations $N$ and the actual degradation matrix $D$ are updated accordingly. The next state of each droplet is randomly sampled from the probability distributions described in Sec. 6.5.2, the droplets are checked for other conditions (e.g., merging and splitting), and the next cycle starts. The process continues until either the bioassay execution is successful, or the maximum number of cycles is reached, in which case the bioassay execution is aborted.

Two routing algorithms were implemented and used for the experiments in this section. The first (baseline) algorithm is unaware of degradation and generates the shortest-path strategy, minimizing the distance traveled by each droplet. The second (adaptive) algorithm follows the proposed synthesis framework to synthesize adaptive routing strategies based on the proposed framework (see Algorithm 8). Neither approach utilized the error-recovery techniques described in [LLM+17, ZLC18a] as
Figure 6.14: Control flow for the simulation environment.

our goal is to proactively avoid errors and the cost associated with error recovery.

6.7.2 Probability of Successful Completion

Since MEDA biochips are fabricated in a CMOS foundry, it is desirable to reuse them as much possible (e.g., for a panel of diagnostic tests for the same patient), as opposed to disposable devices fabricated on a plastic or glass substrate. Therefore, we examined the likelihood of successfully completing multiple runs of a bioassay for a given upper limit on the completion time ($k_{\text{max}}$). We first simulated a fabricated MEDA biochip with $30 \times 60$ MCs [LYL15]. Each MC followed the reliability model in (6.3), with degradation constants $c \sim U(200, 500)$ and $\tau \sim U(0.5, 0.9)$, randomly sampled to simulate microelectrode degradation. Once assigned, both $c$ and $\tau$ remained constant during each set of experiments.

Fig. 6.15 shows that the proposed method ensures a significantly higher probability of successful bioassay completion (PoS) within the given limit on the time-to-result, especially for longer bioassays. For example, with $k_{\text{max}} = 300$ cycles, the proposed approach guarantees the PoS for the Serial Dilution bioassay to be 0.8 com-
Figure 6.15: Probability of successful bioassay completion versus the number of cycles allocated.

pared to 0.1 for the baseline method. Even with more cycles (e.g., 320), the baseline method provides a PoS of only 0.7, while the PoS for the proposed method is 0.99. As expected, the proposed solution is more effective for longer bioassays. Lower $k_{\text{max}}$ values imply fewer actuations per bioassay, increasing the number of successful executions before the biochip fully degrades.

### 6.7.3 Fault Injection During Bioassay Execution

In the next set of experiments, we randomly injected faults in MCs, wherein a droplet can get stuck at a group of faulty microelectrodes. The MCs were divided into two groups: normal and faulty. While both groups follow the degradation model described in Section 6.5, a faulty MC exhibits a sudden failure at random actuation $n$, i.e., $D_{ij}^{(n)} = 0$. Moreover, two modes of fault injection were simulated: uniform and clustered. In the former, faulty MCs are randomly distributed across the biochip, while faults in the clustered mode appear as randomly-placed clusters of four adjacent MCs ($2 \times 2$).
Fig. 6.16 compares the mean number of cycles \( k \) required to repeatedly execute each bioassay (referred to as a “trial”) on the same MEDA biochip (i.e., the same degradation profile) under different routing strategies and fault-injection modes. A trial was terminated after five successful executions or if \( k \) exceeded the maximum allowed number of cycles \( k_{\text{max}} = 1,000 \), in which case the execution was aborted because of excessive chip degradation. The probabilistic behavior in the actuation model implies that every trial uses potentially different droplet routes, therefore we also report standard deviation (SD) values.

The results show that the proposed adaptive method consistently requires fewer cycles to execute a bioassay compared to the baseline method. This gap becomes more pronounced when clustered faults are injected as such clusters act as roadblocks, obstructing droplet movements. In longer bioassays (e.g., Serial Dilution and NuIP), trials featuring the baseline method fail prematurely due to the excessive actuation of the same set of MCs. In contrast, the proposed method leads to successful bioassay execution by proactively avoiding degraded microelectrodes. The mean number of executions to first failure for the proposed method was greater than five in all bioassays, while the baseline method failed as early as in the first execution. Moreover, the relatively small variability (i.e., SD values) in \( k \) for the proposed method indicates its robustness against various distributions of fault occurrences.

### 6.7.4 Synthesis Runtime Performance

In this set of experiments, we examine the runtime performance of the synthesis framework. In particular, we observe the time required for adaptive strategy synthesis under various droplet and biochip sizes. Moreover, we study the impact of this time overhead on bioassay executions.

In the formal synthesis of strategies, the time required for synthesis is impacted
Figure 6.16: Average number of cycles required to execute a bioassay under different routing strategies and fault-injection modes (standard deviations are indicated).

by the state-space size, the number of transitions, and the synthesis query. Hence, we simulate a range of droplet sizes and hazard areas. Since all microfluidic operations are reduced to a number of routing jobs, it suffices to examine the routing jobs directly. The specific values of the health matrix does not impact the model size, except for cases where the health is 0 for a number of adjacent microelectrodes, which can lead to zero-probability transitions. Thus, we enforce the worst-case scenario by assuming a health matrix with no zero elements.
Table 6.5 shows the range of droplet and biochip sizes used in the experiments, and the corresponding model sizes. Note that the RJ area here refers to the size of the area specified by the hazard bounds of the given routing job, regardless of the total biochip size. As expected from the synthesis time complexity (see Sec. 6.6.3), for the same routing job area, the models of smaller droplets are larger in size. The time required to construct the model constitutes at least 90% of the total time required for strategy synthesis. For any droplet size, routing jobs with $20 \times 20$ or less area requires less than 3 seconds for strategy synthesis, which is a tolerable delay for most applications. On the other hand, RJs featuring larger areas may require as long as 10 seconds before the corresponding strategy is synthesized. Such delays can be catastrophic for time-sensitive bioassays; it can also lead to excessive degradation of MCs that are used to hold the droplets in place during that time.

Table 6.5: Performance results for various droplet and biochip sizes.

<table>
<thead>
<tr>
<th>Input Size (MC)</th>
<th>Model Size</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ Area Droplet</td>
<td>#States</td>
<td>#Transitions</td>
</tr>
<tr>
<td>10 × 10 3 × 3</td>
<td>67</td>
<td>1,913</td>
</tr>
<tr>
<td>10 × 10 4 × 4</td>
<td>52</td>
<td>1,419</td>
</tr>
<tr>
<td>10 × 10 5 × 5</td>
<td>39</td>
<td>997</td>
</tr>
<tr>
<td>10 × 10 6 × 6</td>
<td>28</td>
<td>647</td>
</tr>
<tr>
<td>20 × 20 3 × 3</td>
<td>327</td>
<td>10,813</td>
</tr>
<tr>
<td>20 × 20 4 × 4</td>
<td>292</td>
<td>9,599</td>
</tr>
<tr>
<td>20 × 20 5 × 5</td>
<td>259</td>
<td>8,457</td>
</tr>
<tr>
<td>20 × 20 6 × 6</td>
<td>228</td>
<td>7,387</td>
</tr>
<tr>
<td>30 × 30 3 × 3</td>
<td>787</td>
<td>26,720</td>
</tr>
<tr>
<td>30 × 30 4 × 4</td>
<td>732</td>
<td>24,793</td>
</tr>
<tr>
<td>30 × 30 5 × 5</td>
<td>679</td>
<td>22,938</td>
</tr>
<tr>
<td>30 × 30 6 × 6</td>
<td>628</td>
<td>21,155</td>
</tr>
</tbody>
</table>
6.8 Conclusion

We have addressed the problem of microelectrode degradation in MEDA biochips by first introducing a new microelectrode-cell design that provides the health status of the microelectrodes. We have studied how the microelectrode degradation patterns are affected by droplet characteristics. We have also presented experimental results on electrode degradation for fabricated PCB prototypes, validating the degradation model throughout the chapter. We have developed a stochastic game-based model for droplet manipulation that incorporates the health status, and used it to formally synthesize droplet routing strategies that dynamically adapt to the real-time microelectrode health information. Simulation results on six benchmark bioassays show that the proposed framework reduces the number of cycles required to successfully complete a bioassay in realistic microelectrode degradation scenarios. These results open the door for designing a scheduler that can optimize the order in which the microfluidic operations are executed in runtime.
Chapter 7

Adaptive Droplet Routing for MEDA Biochips via Deep Reinforcement Learning

This chapter is an adapted reproduction of “Mahmoud Elfar, Tung-Che Liang, Krishnendu Chakrabarty, and Miroslav Pajic. Adaptive droplet routing for MEDA biochips via deep reinforcement learning. In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 640–645, IEEE, 2022” [ELCP22]; and hence is not available under a Creative Commons license.

7.1 Introduction

Digital microfluidic biochip (DMFB) technology enables the automated manipulation of fluid droplets on the micro-scale. The ability of DMFBs to efficiently execute biochemical protocols has led to its usage in a wide range of applications, including point-of-care clinical diagnostics, biomolecular recognition, air-quality monitoring, and Rapid Acceleration of Diagnostics (RADx) [CLY+15, S+20, HCKF20, GMB+20, She20]. A recent enhancement to DMFB technology is microelectrode-dot-array (MEDA), which like DMFBs, uses microelectrodes to manipulate droplets based on the electrowetting-on-dielectric (EWOD) principle. In contrast to DMFBs, MEDA biochips are fabricated using TSMC 0.35µm CMOS technology [LYL15, HWL+16], resulting in relatively smaller microelectrodes that can be dynamically clustered in runtime to form various microfluidic modules such as droplet mixers or splitters. In addition, MEDA biochips offer real-time droplet sensing capabilities via capacitive-sensing at the microelectrode level [PGCB16, LZPC20].
Microelectrode degradation is a major concern associated with both DMFBs and MEDA biochip [SCF06]. Specifically, repetitive charging and discharging of a microelectrode, charge trapping, and the degradation of the insulating layer drastically reduces the EWOD force generated by the microelectrode, potentially leading to failures at the bioassay level. Techniques proposed to mitigate the impact of microelectrode degradation on bioassay execution can be classified into reactive and preventive. Reactive techniques aim to recover from errors after their occurrence during bioassay execution; examples include droplet remixing and resplitting [ZLC18a], dynamic reconfiguration of the biochip [LLY+16a, LLM+17], and adaptive routing based on the knowledge of the fault map [LZ20]. In contrast, preventive techniques aim to proactively predict and avoid failures. In this category, a recently proposed technique is adaptive routing based on the sensing of microelectrode health levels [ELCP21b].

Reinforcement learning-based droplet routing has been proposed for adaptive droplet routing in DMFBs [LZ20]. A drawback of this method is that it is reactive, i.e., it detects microelectrode degradation after a fault occurs during runtime and adapts the learned policy based on the fault occurrence. Such reactive adaptation to the relatively rapid degradation of microelectrodes is especially detrimental for applications such as flash chemistry that require fast time to response [YNY08].

Formal synthesis combined with the monitoring of microelectrode health status has been proposed to synthesize droplet routing strategies in MEDA [ELCP21b]. While such a framework can provide routing strategies with formal (probabilistic) guarantees, the lack of scalability with respect to the biochip size limits the practical applicability of this method. For instance, synthesizing a routing strategy for a mere 30×30 biochip takes an average of nine seconds [ELCP21b] and is repeated for each microfluidic operation, which is unacceptable for time-sensitive bioassays [YNY08]. In addition, state-of-the-art biochips incorporate a large number of microelectrodes. For
example, aQdrop from Sharp Life Science includes 41000 microelectrodes \cite{AHB21}, which is at least an order of magnitude larger than what is feasible using the synthesis method described in \cite{ELCP21b}. With the explosion of the state space and the inefficient storage of strategies, the problem of formally pre-synthesizing droplet routing strategies for such biochips becomes infeasible.

In this chapter, we address the problem of designing efficient droplet routing policies for MEDA biochips with proactive mitigation of microelectrode degradation via a deep reinforcement learning (DRL) framework. Our framework uses offline DRL to train a deep neural network (DNN) on a MEDA simulation environment, and online DRL to adapt to the degradation rates of individual biochips. In contrast to formally synthesized strategies \cite{ELCP21b}, the proposed framework efficiently stores droplet routing policies for all possible routing jobs as a DNN, eliminating run-time delays and making it more suitable for time-sensitive bioassays \cite{YNY08}. Since our framework does not require state space enumeration, the DNN adopts a parametrized action space where droplet movements depend on its size, reducing the time required to complete a routing job compared to the usage of single- or double-step movements \cite{LZ20,ELCP21b}. Moreover, our framework can be utilized for large MEDA biochips employed in practice \cite{AHB21}.

The contributions of this chapter are as follows.

- We propose a DRL droplet routing framework for MEDA that exploits feedback about the microelectrode health.

- We develop a stochastic model for MEDA that is suitable for the DRL framework, where we utilize a parametrized action space and adaptive droplet movement.

- We design a DNN specific for the MEDA environment to ensure the scalability of the framework, and we further show training results for various environment
configurations.

- We validate the degradation model used in this chapter by fabricating PCB prototypes and measuring the degradation of electrodes under voltage actuation.

- We evaluate the proposed framework for COVID-19 testing bioassays and compare this solution to exiting methods.

The rest of this chapter is structured as follows. Sec. 7.2 provides background on MEDA biochips, microelectrode degradation and adaptive routing. Sec. 7.3 introduces the proposed online DRL framework for adaptive droplet routing. Sec. 7.4 provides the DNN training details. Sec. 7.5 compares the performance of the proposed DRL framework to existing methods for adaptive droplet routing, before concluding the chapter in Sec. 7.6.

### 7.2 Background and Motivation

A MEDA biochip manipulates fluids as discrete droplets of picoliter volume using EWOD \[QB01\]. Multiple microelectrodes can be dynamically grouped to form a fluidic module (e.g., splitter or mixer) during bioassay execution. A typical MEDA biochip is composed of an array of identical microelectrode cells (MCs). Each MC consists of a microelectrode, an electronic control circuit, and a sensing circuit that enables real-time sensing of droplets. In each operational cycle, the sensing circuit discharges and charges the microelectrode, and measures the charging time. The charging time is used to detect whether a droplet is present over the microelectrode. To obtain the positions of on-chip droplets, the sensing results of all the microelectrodes are shifted out using a scan chain.

A number of failure mechanisms for DMFBs has been identified \[XC09\]. Some are related to manufacturing defects; post-fabrication testing can be used to screen for
such defects [SOC05, DYHC15]. However, charge trapping in the dielectric layer and
degradation of the insulator can result in microelectrode degradation [VP99, DPB09].
If an electrode is degraded during bioassay execution, fluidic operations associated
with this degraded electrode will fail, resulting in bioassay failure [ZLC+18b, LZ20].
Hence, to reliably execute bioassays on MEDA biochips, we must proactively avoid
the use of degraded microelectrodes.

To overcome the problem of electrode degradation in EWOD devices, an adaptive
droplet routing method using reinforcement learning was recently proposed [LZ20],
employing DRL models to dynamically learn the degradation process in a biochip.
However, this approach suffers from a key limitation: in order to provide reliable
routing pathways, the model needs to learn from the interaction history of the de-
graded electrodes. Thus, it is likely that some fluidic operations may fail when the
RL agent first encounters the degraded electrode, i.e., the agent has not yet learned
the degradation process.

Finally, a new MC design, using the inherent capacitive sensing for microelectrode
health monitoring, was recently proposed for MEDA [ELCP21b], along with effective
formal synthesis methods that exploits the health status of microelectrodes.

7.3 MEDA Modeling for DRL

We now introduce a MEDA biochip model that aids in effective learning of droplet
routing strategies. For notation used in the model: $\mathbb{Z}$, $\mathbb{N}_0$ and $\mathbb{R}$ denote the set
of integer, natural, and real numbers, respectively. We use $U\{i,j\}$ and $U(a,b)$ to
denote the discrete (i.e., over integers) and the continuous (i.e., over reals) uniform
distributions over $[i,j]$ and $[a,b]$, respectively.
**MEDA Training Environment**

Consider a MEDA biochip of size $W \times H$, denoting the number of MCs in each row and column, respectively. Following [ELCP21b], we model a droplet as a quadruple $\delta = (x_a, y_a, x_b, y_b) \in \Delta$, where $\Delta \subset \mathbb{N}_0^4$ is the set of all possible droplets. A routing task is characterized by the initial (start) and target (goal) droplet locations, denoted by $\delta_s$ and $\delta_g$, respectively. We use $\delta^{(k)}$, $k \in \mathbb{N}_0$, to denote the droplet location at the $k$th control step. Fig. 7.1 shows an example of the droplet location.

Let $(i, j)$ be the coordinates of a given MC; $D_{ij} \in [0, 1]$ be its degradation level, where 1 indicates a fully healthy MC and 0 a fully degraded; and $n_{ij} \in \mathbb{N}_0$ be the total number of control steps at which the MC was actuated. The degradation level can be estimated as $D_{ij}^{(n)} = \tau_{ij}^{n_{ij}/c_{ij}} \in [0, 1]$, where $c_{ij} \in \mathbb{R}_{>0}$ and $\tau_{ij} \in [0, 1)$ are parameters controlling the degradation rate. Those parameters are generally unknown, although their range can be experimentally estimated [VP99]. The degradation level of an MC can be measured through the health measurement unit [ELCP21b]. Given a health measurement unit with $b$-number of bits, the MC measured health is captured by

$$H_{ij}^{(n)} = \lfloor 2^b \cdot D_{ij}^{(n)} \rfloor = \lfloor 2^b \cdot \tau_{ij}^{n_{ij}/c_{ij}} \rfloor. \quad (7.1)$$

On the droplet manipulation level, MEDA biochips support single-step move-
ments in the cardinal directions $A_d$, double-step movements in the cardinal directions $A_{dd}$, and movements in the ordinal directions $A_{dd'}$. The action space for droplet manipulation is hence defined as $A = A_d \cup A_{dd} \cup A_{dd'}$. The probability that an action is successful largely depends on the health level of the group of microelectrodes — referred to as the *frontier set* — primarily responsible for generating the EWOD force for the action to be performed. We employ the probabilistic transitions modeling from [ELCP21b]. Each action, along with the current droplet location, determines the group of microelectrodes to be actuated. We use $U \in \mathbb{B}^{W \times H}$ to denote the actuation pattern matrix (pattern, for short), where $U_{ij} = 1$ indicates that the microelectrode is actuated.

### Parametrized Action Space

In contrast to the traditional action space where each action is associated with an exact number of steps to move, the parameterized action space determines only the direction towards which the droplet is to move, while the number of steps is defined based on the droplet size, shape, and its location relative to the goal. The parametrization of the action space serves multiple purposes. First, it reduces the dimensionality of the model by reducing the action space size. Second, it unifies the action set across different droplet shapes and sizes, enabling the usage of one trained agent for the entire range of droplet sizes. Finally, it allows for moving a droplet beyond two steps at a time.

We define the parameterized action space as the set

$$A = \{a_N, a_S, a_E, a_W, a_{NE}, a_{NW}, a_{SE}, a_{SW}\}.$$  

Let $(\lambda_x, \lambda_y) \in \mathbb{Z}$ be the signed distance (distance, for short) associated with the adaptive action $a \in A$. Algorithm [10] presents the procedure for computing $(\lambda_x, \lambda_y)$ given the current droplet location $\delta$, goal location $\delta_g$, and action $a$. Basically, the
procedure computes the distance based on the droplet size and the movement direction, while avoiding overshooting the goal location. For instance, the droplet shown in Fig. 7.1 is of size $4 \times 3$. Since the maximum reliable distance for the droplet to travel is $(\lambda_x, \lambda_y) = (\lceil w/2 \rceil, \lceil h/2 \rceil) = (2, 1)$, the adaptive action $a_{NE}$ attempts to move the droplet one and two steps in the east and north directions, respectively, within the same control cycle.

**Observation Space**

At each control step $k$, the DRL agent can observe the current sensor matrix $Y \in \mathbb{B}^{W \times H}$ and the health matrix $H^{(k)}$. For a droplet $\delta = (x_a, y_a, x_b, y_b)$, $Y_{ij} = 1$ for all $(i, j) \in [x_a, x_b] \times [y_a, y_b]$, while $Y_{ij} = 0$ indicates no droplet is sensed at the indicated MC. In addition, the observation space incorporates the goal location $\delta_g$ and the biochip area allocated for routing.

**Reward Function**

The primary goal in adaptive droplet routing is for the droplet to reach the target location. Performance metrics in this case include the time and distance traveled by the droplet. Since excessive actuations of individual microelectrodes can lead to their premature failure, the number of actuations per microelectrode is to be incorporated in the routing process.
Let $a^{(k)}$ be the action taken at step $k$ from state $s^{(k)}$, resulting in a new state $s^{(k+1)}$. Thus, the reward is defined as

$$r^{(k)} = \alpha_{\text{dis}} r^{(k)}_{\text{dis}} + \alpha_{\text{ter}} r^{(k)}_{\text{ter}} + \alpha_{\text{act}} r^{(k)}_{\text{act}},$$

where $r_{\text{dis}}$, $r_{\text{deg}}$ and $r_{\text{ter}}$ are the distance, terminal and action rewards, respectively, and $\alpha_i \in \mathbb{R}$ are the respective hyperparameters. To incentivize progression towards the target location, $r_{\text{dis}}$ is defined as

$$r^{(k)}_{\text{dis}} = D(\delta^{(k)}, \delta_g) - D(\delta^{(t+1)}, \delta_g),$$

where $D(\delta^{(k)}, \delta_g)$ denotes the Manhattan distance between two droplet locations. The terminal reward $r_{\text{ter}}$ aids in faster convergence by associating reaching the target location with an additional reward, defined as $r^{(k)}_{\text{ter}} = 1 \{\delta = \delta_g\}$. Finally, the action reward $r_{\text{act}}$ penalizes selecting an invalid action, i.e., an action that causes the droplet to exit the routing job area. The selection of the hyperparameters $\alpha_i$ is discussed in Sec. 7.4.

### 7.4 DRL Agent Design and Training

#### 7.4.1 Training Configurations

We first discuss configuration parameters that affect the training convergence speed – i.e., MEDA biochip size, droplet size, the initial and target droplet locations, the initial microelectrode degradation levels, and the degradation parameters.

**Biochip and Droplet Sizes**

For training, we considered biochips of sizes between $30 \times 30$ and $180 \times 180$. We trained the agent for the most common droplet sizes, with droplet width and height $w, h \in \{2, 3, 4, 5, 6\}$, where $w/h \in [0.8, 1.25]$. We assume that the droplet size is preserved.
throughout a single routing job. Hence, there are two approaches to droplet size selection during training. In the first, multiple agents are utilized, where each agent is trained for a specific droplet size. In the second, the same agent is trained against the range of droplet sizes. Note that a DNN can be feature-invariant by training against the range of values for such feature. Moreover, the exact size of droplets during execution might slightly vary outside those specific values. Consequently, we opt for training a single agent in this framework, i.e., the second approach.

**Initial and Target Locations**

In MEDA biochips, a droplet is either the result of a preceding microfluidic operation or dispensed by an on-chip dispenser. In the former case, the droplet location can be anywhere on the biochip; in the latter, the initial location $\delta_s$ is one of multiple, predefined dispenser coordinates. Similarly, the target location $\delta_g$ can be either where a microfluidic module is (e.g., a mixer or a splitter), or a predefined exit through one of the biochip reservoirs.

For benchmark bioassays, the percentage of routing jobs involving initial (e.g., dispensing operations) or target (e.g., discarding operations) droplets adjacent to one of the biochip edges is between 20% and 40% [Gug20].

Thus, during training both the initial and goal locations are sampled from a stratified distribution. Specifically, we randomly sample $\delta_s$ and $\delta_g$ at the start of each training episode such that $x_{as}, x_{ag} \sim U\{2, W-w-1\}$, and $y_{as}, y_{ag} \sim U\{2, H-h-1\}$.

**Degradation Parameters**

From (7.1), degradation parameters of microelectrodes affect their degradation rate, although they are not directly observable to the agent. For training, we randomly sample the degradation parameters as $c_{ij} \sim U(c_{\text{min}}, c_{\text{max}})$ and $\tau_{ij} \sim U(\tau_{\text{min}}, \tau_{\text{max}})$. 
where the distributions are experimentally obtained as described in Sec. 7.5. On the other hand, the number actuations \( n_{ij} \) is updated based on the actuation patterns applied by the agent at each step.

### 7.4.2 DNN Design and Training

We employ a convolutional neural network (CNN) to learn droplet routing policies due to its potential in preserving important features of the observation space. As illustrated in Fig. 7.2, the input to the CNN is a matrix of size \((H, W, 3)\). The three channels represent the microelectrode health levels and routing zone, the goal location, the current droplet location. The agent’s goal is to learn a policy that maximized the expected cumulative reward.

#### Table 7.1: CNN layers and their configurations.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>Activation</th>
<th>Size</th>
<th>Stride</th>
<th>Padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>Convolution</td>
<td>ReLU</td>
<td>64</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>L2</td>
<td>Convolution</td>
<td>ReLU</td>
<td>128</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>L3</td>
<td>Convolution</td>
<td>ReLU</td>
<td>128</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>L4</td>
<td>Fully-connected</td>
<td>ReLU</td>
<td>256</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>L5</td>
<td>Output</td>
<td>ReLU</td>
<td>8</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Figure 7.2: Channels comprising the observation space.
For training, we use the proximal policy optimization (PPO) algorithm \cite{MBM+16} with actor-critic architecture. Unlike policy gradient methods for reinforcement learning where policy gradients are updated after reading each data sample, PPO utilizes a surrogate objective to stabilize the training process via multiple workers. Algorithm \textbf{11} summarizes the procedure for training the agent to learn droplet routing policies. Each training episode starts with the initial state \((\delta_g, H)\), sampled according to the distributions described earlier. After each step, the number of actuations \(n_{ij}\) is updated for all MCs using the actuation pattern matrix \(U\). An episode is terminated when either the target location is reached, i.e., \(\delta = \delta_g\), or the maximum number of steps allowed has passed, i.e., \(k = 2(W + H)\). Using the accumulated rewards, the gradients for each encountered \((s, a)\) are updated.

To avoid catastrophic unlearning, we adopt a dynamic learning rate scheduler for training. Specifically, the training process starts with a base learning rate \(\eta_0\). At the end of the \(i\)-th epoch, the learning rate is discounted with factor \(\beta\) only if the agent performance is above a certain threshold, i.e.,

\[
\eta_{i+1} = \begin{cases} 
\max (\beta \eta_i, \eta_{\text{min}}) & \text{success rate} > 0.99, \\
\eta_i & \text{otherwise.}
\end{cases}
\]

Through hyperparameter optimization, we chose \(\eta_0 = 3.5 \times 10^{-4}\), \(\eta_{\text{min}} = 1.0 \times 10^{-6}\), and \(\beta = 0.7\).

To accelerate the training process, we apply transfer learning as follows. We first train a randomly initialized CNN on biochips of size \(30 \times 30\) with no fault injection. Next, we use the pre-trained base CNN to initialize the training of the CNNs used for the next biochip size and fault injection level, and the process is repeated using the new CNNs as shown in Fig. \textbf{7.3}. To facilitate the transfer learning, the input layer size is unified across all CNNs by scaling the observation matrix from the original size, i.e., \((W, H, 3)\), to \((30, 30, 3)\).
Algorithm 11: Procedure for Learning Routing Policies

**Input:** MEDA biochip size

for epoch do

resample ← 1

for iter = 1, 2, ..., N_iter do

for actor = 1, 2, ..., N_actor, running in parallel, do

if resample = 1 then

Sample δ_s, δ_g, (τ_ij), (c_ij), and N

resample ← 0

Run current policy π and obtain rewards and new state

if (k ≥ k_max) ∨ (δ = δ_g) then resample ← 1

if (iter · N_actor) mod minibatchsize = 0 then

Optimize PPO2 loss function, update current policy π

---

Figure 7.3: Dataflow diagram for transfer learning. An arrow indicates that the source CNN is used to initialize the training of the destination CNN.

A comparison of the training performance between random initialization and transfer learning approaches is discussed in Sec. 7.5.

7.5 Experimental Evaluation

7.5.1 Estimation of Degradation Parameters

The first set of experiments aims to validate the degradation model and to estimate the parameters in (7.1). To this end, we monitored electrode degradation in several identical PCB-based DMFBs. Note that these biochips manipulate droplets using the same EWOD principle as MEDA biochips. The DMFBs contain electrodes in three sizes: 2×2 mm², 3×3 mm², and 4×4 mm² (see Fig. 7.4). Four reservoir modules are placed on two sides of the biochip, and these modules can dispense different reagent droplets. Each electrode can be controlled individually, and these control signals come
from a control board underneath the DMFB. The activation/de-activation status of each electrode is controlled by a high voltage relay. Each high-voltage relay IC is controlled by a configuration bit, and these configuration bits are stored in the register ICs. The hardware setup used to operate the digital microfluidic biochip is shown in Fig. 7.4. To efficiently run the experiments, we executed the same actuation pattern on two DMFBs at the same time.

We developed an actuation sequence for the electrodes that leads to repeated fluidic operations on the biochip. When executing the actuation sequence, each electrode is actuated for 1 s for hundreds of times. After executing the sequence, we actuated an electrode and measured the charging times needed using an oscilloscope. Because the electrode and the top plate form a capacitor, and a resistor is placed in series with the electrode, the charging path is a simple RC circuit. The effective capacitance of an electrode at time \( t \) can then be derived using \( V_C(t) = V_{pp} \left( 1 - e^{-t/RC} \right) \), where \( V_C(t) \) is the electrode capacitance at time \( t \). Subsequently, the EWOD force \( F \) can be obtained from \( LLY^{+17} \) [Fai07]:

\[
F = \frac{C(V_C - V_T)^2}{2} \frac{dA(x)}{dx},
\]

where \( V_T = 130\text{V} \) is threshold voltage due to soldermask insulator [Fai07], \( A(x) \) is the area of the droplet over the activated electrode, and \( x \) is the droplet position.

The degradation results, i.e., increase in capacitance and decrease in EWOD force,
of five DMFBs are shown in Fig. 7.5 — the capacitance of an electrode grows linearly as we repeatedly actuate the electrode, decreasing the EWOD force. The degradation parameters are estimated as $\tau \in [0.5, 0.7]$ and $c \in [500, 800]$, and are further used for DRL agent training.

### 7.5.2 Evaluation

We next present the results for training agents for various configurations by showing the mean score of the agents after each training epochs for biochips of sizes $W \times H$, where $W = H \in \{30, 60, 120, 180\}$. Performance metrics consist of the mean score, the number of cycles, as well as the success rate. The metrics are collected after each training epoch by testing the agent for 500 random routing jobs, and each experiment is repeated five times. All experiments were carried out with eight parallel environments and $2^{14}$ total number of steps. The training and experiments were conducted on an Intel Xeon Silver 4208 CPU and an Nvidia RTX 6000 GPU with 24 GB of memory. The training and simulation environment were implemented using Python, including OpenAI Gym and Stable-Baselines libraries.
We first trained the CNN on a healthy MEDA biochip, i.e., the number of actuations per each microelectrode is reset at the beginning of each training episode. Fig. 7.6 presents the CNN performance metrics versus the number of training epochs. The trends show that after a low number of epochs, a CNN learns an effective policy — i.e., the success rate converges to 100% and the score and the average number of cycles stabilize — at a relatively small number of epochs that ranges from 10 to 40 and increases with the biochip size.

We then deployed transfer learning by using the CNN trained on healthy MEDA biochips of size $30 \times 30$ to initialize the training of the CNN for the next biochip size, i.e., $60 \times 60$. The process was repeated subsequently for the remaining biochip sizes. Fig. 7.7 compares the performance of the CNNs trained via random initialization against the ones trained via transfer learning. For all biochip sizes, the transfer learning-based CNNs were able to learn effective policies within the first training epoch, exhibiting the same performance that the CNNs trained via random initialization were able to achieve after 15 to 40 epochs.

We also tested the robustness of the trained agents against randomly injected faults at runtime. We used the agents trained on healthy biochips to initialize the training against biochips with randomly injected faults. Before each training episode, a fixed percentage of fully-degraded microelectrodes are randomly placed in clusters of size $2 \times 2$. Similar to the previous experiments, the trained agents were used to
initialize the training on a higher percentage of faults. Fig. 7.8 shows the performance results for training against 10% and 20% fault injection modes. The trends demonstrate that the agents were able to adapt to the faults within the first training epoch.

Finally, to evaluate the trained CNNs, we run experiments where we compare their performance against two baselines: (i) health-agnostic policies that aim to minimize the time to reach the target without knowledge of the MC health levels (baseline), and (ii) formally synthesized strategies using PRISM-games model checker (formal) [ELCP21b]. Each policy was used to execute two benchmark bioassays that are used for COVID-19 testing: PCR-based (COVID-PCR) and rapid antigen-based (COVID-RAT), are widely used to detect the presence of the SARS-CoV-2 virus or the body’s response to infection [Gug20].
Fig. 7.9 shows the probability of successful bioassay completion within a given number of cycles $k$. The graph shows that the DRL outperforms the policies from the literature by achieving a significantly higher probability of success. The gain in performance is primarily due to the utilization of adaptive movement distance (see Sec. 7.3).

For instance, DRL successfully executed COVID-PCR within $k = 762$ with probability $p > 0.9$, compared to $p < 0.4$ using the other policies. In addition, the time needed to obtain a routing policy from the trained CNN is negligible ($t < 0.1$ sec) when compared to the formally synthesized policies where $t$ ranged from 5 to 48 sec before each routing job.

7.6 Conclusion

In this chapter, we have proposed a deep reinforcement learning (DRL) framework to address the problem of designing droplet routing policies for MEDA biochips with proactive mitigation of microelectrode degradation. The proposed framework utilizes the recently-developed technology of microelectrode health monitoring by incorporating the microelectrode health status into the observation space. Our framework uses
DRL to train CNNs for droplet routing policies in MEDA biochips. We have shown that the proposed framework is superior to existing formal synthesis techniques in terms of the probability of successful bioassay completion and the scalability with respect to the biochip size.
Chapter 8

Conclusions

This dissertation explored the problem of modeling and designing of adaptive and assured CPS in two domains: Human-Cyber-Physical System (HCPS) and Micro-Electrode-Dot Array (MEDA). This chapter summarizes the contributions of the dissertation, and further discusses potential directions for future research.

8.1 Summary and Contributions

This dissertation proposed formal frameworks to aid with the modeling and design of adaptive and assured CPS. The first half of this dissertation was motivated by security-aware CPS in which stealthy malicious actions (by an adversarial agent) are potentially hidden from the system. For such systems, Chapter 2 proposed Delayed Action Game (DAG) for modeling multi-agent (i.e., multi-player) systems with hidden information. The formalism deploys the concept of delaying player’s actions as means to hide them from the other players without the usage of private variables. Consequently, DAG-based models can be implemented and analyzed using off-the-shelf model checkers. The chapter provided the syntax and semantics of DAGs, and proved the simulation relation between DAGs and hidden information games (HIGs). It also proposed an algorithm that utilizes off-the-shelf model checkers to formally synthesize control strategies using DAG-based models. Experimental results were given for two case studies.

Since temporal logic can be used to formalize requirements during the CPS design process, the design of specification languages that are both expressive and easy to use is paramount. Chapter 3 addressed the problem of formalizing temporal re-
quirements for CPS that are naturally specified as a set of temporal objectives with an underlying priority structure. Formalizing such requirements can be challenging as they involve multiple objectives where the context upon which they are switched is of probabilistic nature. To this end, context-aware probabilistic temporal logic (CAPTL) was proposed. The logic provides the ability to formalize temporal requirements and prioritize them based on some probabilistic invariants. The chapter defined the CAPTL syntax, and specified the satisfaction semantics for CAPTL over Markov Decision Processes (MDPs). Furthermore, an algorithm was provided for synthesizing strategies for an MDP that satisfy a given CAPTL-based requirement. Two case studies were also provided for evaluation.

Chapter [4] demonstrated an application for the DAG synthesis framework in Human-Unmanned Aerial Vehicle (HUAV) systems where the operator can intermittently perform geolocation tasks to aid in detection of possible attacks. The system dynamics, operator geolocation task, and the adversarial behavior were modeled as an HIG. To realize the model parameters, experiments were conducted using RESCHU-SA testbed to understand operator strategies for performing geolocation tasks, while machine learning techniques were deployed to predict correctness of the geolocation task at a given location. Using the HIG model, the corresponding DAG was constructed, and the model checker PRISM-games was used to synthesize protocols for the HUAV system that minimizes time-to-target while ensuring the UAV security. The protocols specified the UAV path plans as well as the time instances at which the operator is advised to perform the geolocation tasks.

The second half of this dissertation was motivated by MEDA technology. In MEDA biochips, faults may arise during a bioassay execution, which in turn can lead to undesirable results in one or more of the microfluidic operations, and eventually leading to the failure of the entire bioassay. On the microfluidic operation level,
Chapter 5 showed how to improve the reliability of MEDA biochips by formally synthesizing error-recovery protocols. Specifically, we developed a high-level model for various microfluidic operations as Markov Decision Process (MDP), from which an SMG model was obtained. We also used temporal logic to formalize error-recovery objectives for bioassay executions. Using both the SMG model and the formal objectives, we utilized PRISM-games model checker to synthesize optimal error-recovery protocols for each microfluidic operation. For evaluation, experimental results were obtained for three real-life bioassays.

On the droplet manipulation level, errors may occur when microelectrodes lose their ability to generate adequate Electrowetting-on-dielectric (EWOD) forces as they degrade due to repetitive activation cycles. In Chapter 5 we improved the reliability and expected lifetime MEDA biochips by developing a framework for adaptive droplet manipulation that takes advantage of the ability to read microelectrode health level in real-time. We first developed an SMG model for droplet manipulation in MEDA biochips, incorporating the microelectrode health measurements into the state space. We also formalized error-recovery objectives for bioassay executions using temporal logic. Using the developed model, we devised an algorithm that transforms a given microfluidic operation into a set of droplet routing jobs, for each of which model checkers are utilized to synthesize an optimal routing strategy that adapts to the current microelectrode health levels. We performed extensive experimental evaluation on six real-life bioassays. The results showed that our framework improved both the probability of successful bioassay execution and the expected number of cycles per each.

Although the framework proposed in Chapter 5 provides probabilistic guarantees on the performance of the synthesized routing strategies, the framework struggles with biochips featuring a large number of microelectrodes. To overcome this lim-
Chapter [7] proposed a Deep Reinforcement Learning (DRL) framework for the droplet routing problem. In lieu of model checking, we proposed a DRL training procedure that deploys the PPO2 algorithm to train Deep Neural Network (DNN). To this end, we adapted the SMG model for droplet movement to render it suitable for training, and further implemented the model as a simulation environment in Python using OpenAI Gym and Stable-Baselines libraries. To increase the efficiency of the training process, we adopted a parametric action space, effectively reducing the action space used for training. Moreover, we deployed transfer learning to accelerate training across various biochips sizes by unifying the observation space. For evaluation, we compared the times required to train CNNs using traditional and transfer learning, proving that the training is more efficient using the latter. To evaluate the trained CNNs, we compared their performance against the synthesized strategies from the previous chapter using the probability of successful bioassay execution and the expected number of cycles as metrics for comparison.

### 8.2 Limitations and Future Avenues

In Chapter [2] we showed how DAGs utilize the concept of delayed actions to model hidden information in multi-agent systems, allowing for the use of off-the-shelf model checkers. We also showed how a DAG can be obtained by transforming an existing HIG (where private variables are used to model hidden information). However, this transformation comes at the cost of increasing the size of the state space. Depending on the original HIG model size and structure, the obtained DAG size might surpass that of which model checkers can handle. Nonetheless, the use of a compositional approach for strategy synthesis, where subgames are used to simplify the model, can be used to mitigate the state space size problem. Since we only studied strategy synthesis based on single objectives, optimizing the framework for multi-objective is
a possible future avenue of investigation.

As demonstrated in Chapter 3, CAPTL can be used to formalize temporal requirements and prioritize them based on probabilistic invariants. We also derived the CAPTL satisfaction semantics and a synthesis algorithm over MDPs. Since many CPS models involve non-deterministic behaviors that are controlled by more than one actor, the satisfaction semantics and synthesis algorithm for CAPTL over SMGs are to be investigated. Given that the provided CAPTL-based synthesis algorithm relies on functions provided by a model checker, the algorithm performance and limitations (e.g., scalability with respect to model size) are bound by those for the model checker used. Hence, it is worth developing synthesis algorithms for CAPTL that relies on other methods such as reinforcement learning.

In Chapter 4 we exploited parallel computation to reduce the time required for DAG-based strategy synthesis. Nonetheless, the exploration time of each subgame grows exponentially with both the exploration horizon and the grid size. In this case, approximate model checking techniques can be beneficial, albeit at the cost of accuracy. While the case study described in Chapter 4 was given to demonstrate an application for the DAG-based strategy synthesis, the experimental evaluation conducted can be improved. For example, the survey conducted to study the impact of individual imagery skills on the geolocation task could be replaced with a properly designed experiment. Moreover, the study can be improved by utilizing available datasets for aerial imagery, incorporating other factors such as the altitude, travel speed and time-of-day at which the satellite images are examined. Since the synthesized protocols are to be used to build an advisory system, the impact of the trust level between the human operator and the system on the expected performance should be considered.

In Chapter 5 we developed stochastic models for microfluidic operations in MEDA
biochips. The probabilities used in those models were derived from data collected by performing experiments on real biochips. However, the probability of an error occurring in a microfluidic operation is more likely to increase over time as the microelectrodes degrade due to repetitive activation cycles. Consequently, the models can be improved by incorporating the degradation rate and the number of activation cycles into account. To this end, further experimental data on real MEDA biochips, where the number of activation cycles is collected, are required.

The problem of developing adaptive droplet routing policies for MEDA biochips was addressed in both Chapter 6 and Chapter 7 using two different methodologies. While the former proposed temporal logic and model checkers to capture routing requirements and to synthesize routing plans, respectively, the latter employed reward functions and DRL-based algorithms. In case of the former, the synthesized strategies provide probabilistic guarantees on the resulting performance. However, those guarantees come at a price since synthesizing a strategy for a given routing job can be computationally expensive and the synthesis process may take 10 to 30 seconds (in run-time) for a biochip of size $60 \times 30$. Such delay is oftentimes unacceptable for time-sensitive bioassays. Moreover, state-of-the-art biochips are now taking advantage of the MEDA technology by incorporating an even larger number of microelectrodes than what model checkers can reasonably handle. For those reasons, further investigation into optimizing the synthesis framework is essential to maintaining its relevance.

The DRL-based framework (described in Chapter 7) overcomes the limitations of the previous one in two aspects. First, the CNNs can be trained offline for droplet routing, significantly reducing any delays in obtaining the routing policies in runtime. Second, CNNs inherently provide an efficient way to store and retrieve such policies. With that said, the DRL-based framework comes with its own challenges.
For instance, the trained CNNs do not provide any guarantees on the performance of the routing policies. In fact, even with perfect training scores, a CNN may fail to successfully perform a rather-simple routing job in run-time. Such behavior can occur due to the use of a dataset (of routing jobs) for training that is not representative of the ones faced during execution. Even with a training dataset that is representative of the distribution of the routing jobs encountered during real bioassay execution, the frequency distribution of the encountered microelectrodes may still be skewed, rendering the trained CNNs sensitive to small variations in the actual routing jobs. For those reasons, future work shall consider developing training datasets that improves the robustness of the trained CNNs. In both frameworks, incorporating the scheduler for microfluidic operations into the scope of the routing problem is worth considering. Finally, developing a hybrid framework that combines both the model checking techniques to provide performance guarantees, and the DRL algorithms to enhance the run-time performance, is an avenue for future work.
Bibliography


182


[DPB09] Antonis I Drygiannakis, Athanasios G Papathanasiou, and Andreas G Boudouvis. On the connection between dielectric breakdown strength,


Tung-Che Liang, Zhanwei Zhong, Miroslav Pajic, and Krishnendu Chakrabarty. Extending the lifetime of MEDA biochips by selective


Biography

Mahmoud Elfar received the B.Sc. degree in mechatronics from Ain Shams University, Cairo, Egypt. This dissertation was submitted in partial fulfillment towards his Ph.D. degree in computer engineering from Duke University, Durham, NC, USA.

Previously, he was an embedded software engineer with Valeo. He was also a research assistant at the American University in Cairo, and a visiting researcher at the German University in Cairo. Prior to that, he was a research and development engineer with Schneider Electric. His recent publications include the following:


