FORMAL SYNTHESIS OF AUTONOMOUS CYBER-PHYSICAL SYSTEMS: RESILIENCE AND SECURITY

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Jin Dai

Hai Lin, Director

Graduate Program in Electrical Engineering
Notre Dame, Indiana
July 2019
© Copyright by

Jin Dai

Email: jdai1@nd.edu

ORCID: 0000-0002-8841-8270

2019

All Rights Reserved
FORMAL SYNTHESIS OF AUTONOMOUS CYBER-PHYSICAL SYSTEMS:
RESILIENCE AND SECURITY

ABSTRACT
by
Jin Dai

The ubiquitous development of information technology (IT) infrastructures has greatly enhanced the technological evolution from automated to autonomous Cyber-Physical Systems (CPSs), such as networked mobile robots, autonomous vehicles, intelligent transportation systems, flexible manufacturing systems and so on. Increasing demands for safety, resilience, security and operational efficiency of CPSs have motivated us to pursue a correct-by-construction design methodology. This dissertation is devoted to the study of the resilient and secure coordination and control problems of a class of CPSs that consist of multiple heterogeneous subsystems performing under uncertain environments, and contributions are made to both the theory and designs of provably correct local control policies and coordination rules to achieve complex goals.

By characterizing the cyber logical behaviors of the CPSs as Discrete Event Systems (DESs), we start our study with the supervisory control problems of DESs whose complete plant models are not necessarily given a priori. By modifying $L^*$ learning algorithm, we develop learning-based supervisor synthesis algorithms over unknown plant under both complete and partial observation. Furthermore, automatic synthesis methods of decentralized supervisors when the plant is unknown is also considered, and a learning-based algorithm that solves the decentralized supervisory control problem is proposed in terms of language controllability and co-normality. The correctness and finite convergence of the
proposed algorithm are proved.

We then investigate the formal coordination and control problem of cooperative multi-agent systems for a regular language specification. We propose an automatic synthesis framework to synthesize appropriate local mission plans and corresponding motion plans for each agent by integrating supervisory control theory and compositional verification techniques. Furthermore, by presenting a modified $L^*$ learning algorithm, we are able to reconfigure the synthesized motion plans whenever the real environment is different from the idealized prior knowledge.

Next, the resilience of the proposed coordination and control framework in the presence of potential sensor and/or actuator faults is investigated. The sensor fault tolerant control scheme involves local supervisor reconfiguration upon the detection of the sensor faults, which are captured by permanent loss of observability of certain sensor reading events; while for the actuator faults that are modeled as loss of controllability of local events, a controller switching mechanism is developed for each agent after the detection of the faults. An assume-guarantee post-fault coordination among different subsystems is applied so that fault tolerance can be achieved.

Apart from the high-level performances, we take the dynamics of the robots into consideration and study the path planning problem with application to robotic systems. Based on the $\mu$-tangency conditions, a computationally efficient path planning approach is presented to derive the existence conditions and computationally efficient construction for a class of sub-optimal continuous-curvature paths for nonholonomic mobile robots. The proposed approach not only allows fast determination of the existence of a continuous curvature path, but also guarantees to construct the desired continuous curvature path.

Finally, we also study a confidentiality notion named opacity that characterizes the information-flow between the cyber and the physical parts of an autonomous cyber-physical system within the DES formalism. More specifically, an enforcement mechanism based on the implementation of insertion functions is developed to assure decentralized and joint
opacity properties if the system can be observed by multiple intruders either with or without coordination.

The results on simulations on some illustrative examples are given to show that the theory established is correct and the proposed algorithms are efficient.
To Mom and Dad, who have given me their endless love and support,
To Ruiwen, my beloved, who has stood with me and offered me great encouragement,
and to my upcoming child.
# CONTENTS

FIGURES ................................................................. vii

TABLES ................................................................. x

ACKNOWLEDGMENTS .................................................. xi

CHAPTER 1: INTRODUCTION ............................................. 1

1.1 Background Information and Research Motivation ................. 1
  1.1.1 Research Motivation ........................................ 1
  1.1.2 Overview of the Dissertation ............................... 5

1.2 Organization and Main Contributions ................................ 7

CHAPTER 2: PRELIMINARIES ............................................ 10

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

2.2 Basics of Automata Theory and Formal Languages ................. 11
  2.2.1 Finite Automata and Regular Languages .................. 11
  2.2.2 Büchi Automata and ω-regular Languages ................ 18

2.3 The $L^*$ Learning Algorithm .................................... 21
  2.3.1 Details of the $L^*$ Learning Algorithm ................. 21
  2.3.2 Characteristics of the $L^*$ Algorithm ................... 24
  2.3.3 An Example of the $L^*$ Algorithm ....................... 25

2.4 Essentials of Compositional Verification Methods ................. 27
  2.4.1 Assume-guarantee Reasoning in Compositional Verification 27
  2.4.2 Assume-guarantee Reasoning for Multiple Subsystems .... 29

2.5 Conclusion ....................................................... 31

CHAPTER 3: LEARNING-BASED SUPERVISOR SYNTHESIS OF DISCRETE
EVENT SYSTEMS FOR FORMAL SPECIFICATIONS ...................... 32

3.1 Introduction ....................................................... 32

3.2 Learning-based Supervisor Synthesis of Discrete Event Systems 34
  3.2.1 The Ramadge-Wonham Supervisory Control Theory .......... 34
  3.2.2 Problem Formulation ........................................ 37
  3.2.3 The $L^*_C$ Algorithm: Supervisor Synthesis via Automata Learning 38
  3.2.4 Characteristics of the $L^*_C$ Algorithm ................. 43
  3.2.5 An Illustrative Example .................................... 47
CHAPTER 4: FORMAL MISSION AND MOTION PLANNING OF COOPERATIVE MULTI-AGENT SYSTEMS VIA AUTOMATA LEARNING

4.1 Introduction ................................................. 74
4.2 Related Work ............................................... 77
  4.2.1 Formal Planning of Autonomous Robots ............ 77
  4.2.2 Performance-guaranteed Coordination of Multi-agent Systems 79
4.3 Problem Formulation ......................................... 82
  4.3.1 Formal Models of Cooperative Multi-agent Systems .... 82
  4.3.2 Formal Mission and Motion Planning Problem of Cooperative Multi-agent Systems ......................... 84
4.4 Overview of the Approach ................................... 86
  4.4.1 The Formal Mission and Motion Planning Framework .... 86
  4.4.2 A Motivating Example of Multi-Robot Coordination .... 88
4.5 Counterexample-guided Mission Planning of Cooperative Multi-agent Systems ........................................... 92
  4.5.1 Learning-based Synthesis of Local Mission Supervisors .... 92
  4.5.2 The $L_C^V$ Algorithm: Assume-guarantee Coordination of Multiple Agents ......................................... 97
  4.5.3 Counterexample-guided Re-synthesis of the Mission Plans .......... 108
4.6 Formal Motion Planning of Cooperative Agents ............ 118
  4.6.1 Automatic Generation of Motion Plans ............... 119
  4.6.2 Counterexample-guided Re-planning in Uncertain Environments .... 121
4.7 Conclusion ..................................................... 125

CHAPTER 5: COORDINATION AND CONTROL OF DISTRIBUTED DISCRETE EVENT SYSTEMS IN THE PRESENCE OF ACTUATOR AND SENSOR FAULTS ............................................. 126
5.1 Introduction .................................................. 126
5.2 Related Work ................................................ 128
5.3 Problem Formulation .................................................. 130
  5.3.1 Actuator and Sensor Faults in Discrete Event Systems ....... 130
  5.3.2 Coordination and Control in the Presence of Faults .......... 133
5.4 The Fault Tolerant Coordination and Control Framework .......... 134
  5.4.1 The Active Fault Tolerant Coordination and Control Framework .. 134
  5.4.2 A Multi-robot Coordination Example ......................... 135
5.5 An Active Approach to Actuator Fault Tolerant Control ......... 140
  5.5.1 Safe Detection of Actuator Faults ............................ 140
  5.5.2 Construction of the Post-fault Subsystem .................... 141
  5.5.3 The Active Actuator Fault Tolerant Control Architecture ... 144
  5.5.4 Fault Tolerant Control with Multiple Actuator Faults ...... 149
5.6 Safe Detection and Active Fault Tolerant Control against Sensor Faults ... 155
  5.6.1 Supervisory Control System with Sensor Faults ............ 155
  5.6.2 SF-safe Controllability and Active Sensor Fault Tolerant Control .. 160
  5.6.3 Active Sensor Fault Tolerant Control Architecture .......... 164
  5.6.4 Fault Tolerant Control with Multiple Faulty Sensors ...... 167
5.7 Fault Tolerant Control in the Presence of Combinations of Faults ... 172
5.8 Assume-guarantee Post-fault Coordination of Distributed DESs ...... 173
5.9 Conclusion ............................................................ 180

CHAPTER 6: CONTINUOUS-CURVATURE PATH PLANNING OF NONHOLO-
NOMIC MOBILE ROBOTS .................................................. 181
6.1 Introduction ........................................................... 181
6.2 Robot Models and Problem Formulation ........................... 184
  6.2.1 Kinematic Model of Car-like Mobile Robots ................. 185
  6.2.2 Optimal Path Planning Problem ............................... 186
6.3 Continuous Curvature Paths via Line Segments and Clothoids .... 188
  6.3.1 The Class of Reeds-Shepp’s Paths ............................ 189
  6.3.2 Clothoid Curves and Clothoid Turns ........................ 190
  6.3.3 The $\mu$-Tangency Conditions for Clothoid Turns .......... 194
6.4 Existence Conditions for CC Paths with RS Path Patterns ..... 197
  6.4.1 The $CSC - 1$ Paths ............................................ 198
  6.4.2 The $CSC - 2$ Paths ............................................ 199
  6.4.3 The $C|C|C$ Paths ............................................ 200
  6.4.4 The $C|CC$ Paths ............................................. 202
  6.4.5 The $CC|C$ Paths ............................................. 204
  6.4.6 The $CC_u|C_uC$ Paths ....................................... 205
  6.4.7 The $C|C_uC_u|C$ Paths .................................... 208
  6.4.8 The $C|C_{\frac{1}{2}}SC - 1$ Paths ................................ 210
  6.4.9 The $C|C_{\frac{3}{2}}SC - 2$ Paths ................................ 213
  6.4.10 The $C|C_{\frac{5}{2}}SC_{\frac{1}{2}}|C$ Paths .......................... 215
6.5 Simulation Results .................................................... 217
6.6 Conclusion ............................................................. 219
FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>General architectures of automated and autonomous systems.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>The hierarchical control architecture for each subsystem.</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>A general system architecture of autonomous cyber-physical systems.</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Observation tables and conjectured DFA (Part 1 of 3).</td>
<td>25</td>
</tr>
<tr>
<td>2.2</td>
<td>Observation tables and conjectured DFA (Part 2 of 3).</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>Observation tables and conjectured DFA (Part 3 of 3).</td>
<td>27</td>
</tr>
<tr>
<td>3.1</td>
<td>The feedback loop of supervisory control with complete observation</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>The feedback loop of supervisory control under partial observation</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>The DFAs of $G$ and $L$.</td>
<td>56</td>
</tr>
<tr>
<td>3.4</td>
<td>The DFAs of $S$ and $S_P$.</td>
<td>57</td>
</tr>
<tr>
<td>3.5</td>
<td>The communication system $G$.</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>The plant $G$ with the specification $S$.</td>
<td>59</td>
</tr>
<tr>
<td>3.7</td>
<td>The $L_C^G$ supervisor synthesis procedure.</td>
<td>68</td>
</tr>
<tr>
<td>3.8</td>
<td>The plant $G$ with the specification $S$.</td>
<td>69</td>
</tr>
<tr>
<td>3.9</td>
<td>The conjecture supervisors $M^0$ and $M^1$ for $\sup C(L(S))$.</td>
<td>70</td>
</tr>
<tr>
<td>3.10</td>
<td>The conjecture DFA $M^1$ for $\sup C(L(S))$.</td>
<td>70</td>
</tr>
<tr>
<td>4.1</td>
<td>Learning-based mission and motion planning framework for cooperative multi-agent systems.</td>
<td>87</td>
</tr>
<tr>
<td>4.2</td>
<td>The multi-robot coordination example.</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>The motion DFA $G_i^m$ for the robot $G_i$ ($i = 1, 2, 3$).</td>
<td>89</td>
</tr>
<tr>
<td>4.4</td>
<td>The mission DFA of the robot $G_2$.</td>
<td>92</td>
</tr>
<tr>
<td>4.5</td>
<td>The DFA model of the robot $G_i$ ($i = 1, 3$).</td>
<td>92</td>
</tr>
<tr>
<td>4.6</td>
<td>The initialized mission $L_1^{mi}$ for the robot $G_1$.</td>
<td>94</td>
</tr>
<tr>
<td>4.7</td>
<td>The initialized mission $L_2^{mi}$ for the robot $G_2$.</td>
<td>95</td>
</tr>
<tr>
<td>4.8</td>
<td>The initialized mission $L_3^{mi}$ for the robot $G_3$.</td>
<td>95</td>
</tr>
<tr>
<td>4.9</td>
<td>The local mission supervisor $S_2$ for the robot $G_2$.</td>
<td>96</td>
</tr>
<tr>
<td>4.10</td>
<td>The local mission supervisor $S_1$ for the robot $G_1$.</td>
<td>96</td>
</tr>
<tr>
<td>4.11</td>
<td>The local mission supervisor $S_3$ for the robot $G_3$.</td>
<td>97</td>
</tr>
<tr>
<td>4.12</td>
<td>The compositional verification framework.</td>
<td>99</td>
</tr>
<tr>
<td>4.13</td>
<td>A robot equipped with a gripper.</td>
<td>104</td>
</tr>
<tr>
<td>4.14</td>
<td>The two-robot coordination system.</td>
<td>104</td>
</tr>
</tbody>
</table>
4.15 The two-robot coordination system. ........................................ 105
4.16 The observation table $T_1$ and $A_1^1$. ................................. 106
4.17 The observation table $T_2$ and $A_2^1$. ................................. 107
4.18 Updated local mission supervisor $\tilde{S}_1$. ...................... 116
4.19 Updated local mission supervisor $\tilde{S}_2$. ...................... 117
4.20 Updated local mission supervisor $\tilde{S}_3$. ...................... 117
4.21 Local motion plans $L_{mo}^i (i = 1, 2, 3)$ in the nominal environment $E_0$. ...................... 124
4.22 Integrated local plan $LP_1$ for the robot $G_1$. .............. 124
4.23 Integrated local plan $LP_3$ for the robot $G_3$. .............. 124
4.24 Integrated local plan $LP_2$ for the robot $G_2$. .............. 125

5.1 The fault tolerant coordination and control framework. ....... 135
5.2 The multi-robot coordination scenario. ............................... 136
5.3 The DFA model of the robot $G_2$. .................................. 138
5.4 The DFA model of the robot $G_i$ $(i = 1, 3)$. .................... 138
5.5 The nominal supervisor $S_1$. ........................................ 139
5.6 The nominal supervisor $S_2$. ........................................ 139
5.7 The nominal supervisor $S_3$. ........................................ 139
5.8 The subsystem $G_i$. ................................................ 143
5.9 The nominal supervisor $S_i$. ........................................ 143
5.10 The post-fault subsystem $G^a_i (t_{i,0})$. ......................... 144
5.11 The active fault tolerant control architecture of $G_i$ against the actuator fault $h_{i,m}$. ................................. 145
5.12 The post-fault model of the robot $G_3$. ...................... 148
5.13 The post-fault supervisor $S_3^{G_3 to D_1,a}$. .................... 149
5.14 The DFA model of $G_i$. ........................................ 158
5.15 The DFA models of $G_i^k$ and $S_i^k$. .................................. 159
5.16 The DFA model of $G_{i,s}^k$. ....................................... 160
5.17 The DFA model of $G_{i,s}^k$. ....................................... 163
5.18 The safe diagnoser $G_{d}^i$ of $G_{i,s}^k$. ............................. 164
5.19 The active fault tolerant control framework of $G_{i,s}^k$. .... 165
5.20 Sensor fault tolerant control with multiple faults. .......... 169
5.21 Fault tolerant assume-guarantee coordination. ................... 174
5.22 The coordination supervisor $S_1^{CO}$. ............................ 179

6.1 A car-like robot with reference point $R$ [63]. .................. 185
6.2 The curvature profile along with an RS path [63]. ............ 190
6.3 CC Circles $C^+_i (q_s), C^-_i (q_s), C^-_i (q_s)$ and $C^-_i (q_s)$ [63]. .................................................. 193
6.4 $\mu$-tangency between line segments and CC circles. ........ 195
6.5 $\mu$-tangency between CC circles (with no cusp). ............ 196
6.6 $\mu$-tangency between CC circles (with a cusp). ........................................ 197
6.7 Geometric planning of $CSC - 1$ paths. .................................................. 198
6.8 Geometric planning of $CSC - 2$ paths. .................................................. 199
6.9 Geometric planning of $C\mid C\mid C$ paths. ............................................. 201
6.10 Geometric planning of $C\mid CC$ paths. ............................................... 203
6.11 Geometric planning of $CC\mid C$ paths. ............................................... 204
6.12 Geometric planning of $CC_u\mid C_uC$ paths. ....................................... 206
6.13 Geometric planning of $C\mid C_uC_u\mid C$ paths .................................... 209
6.14 Geometric planning of $C\mid C_uSC - 1$ paths. ..................................... 211
6.15 Geometric planning of $C\mid C_uSC - 2$ paths ..................................... 213
6.16 Geometric planning of $C\mid CSC\mid C$ paths ....................................... 215
6.17 Reeds-Shepp's and continuous-curvature paths. ................................... 218
6.18 Reeds-Shepp and continuous-curvature paths. ..................................... 219

7.1 Monitoring and opacity-enforcement of discrete event systems. .......... 223
7.2 Opacity-enforcement based on event insertion. ................................. 231
7.3 Output of an insertion function. ................................................................. 233
7.4 A DES $G$ observed by non-coordinating intruders. .............................. 235
7.5 The DES plant $G$ (the secret states are shaded) ................................. 238
7.6 The observer automaton $Obs_1(G)$. ..................................................... 238
7.7 The observer automaton $Obs_2(G)$ .......................................................... 238
7.8 The AIS $AIS_1$ for intruder $I_1$. ......................................................... 239
7.9 The AIS $AIS_2$ for intruder $I_2$. ......................................................... 241
7.10 The NFM $M_1$ for $AIS_1$. .............................................................. 244
7.11 The NFM $M_2$ for $AIS_2$. .............................................................. 244
7.12 The DES $G$ observed by intruders with intersection-based coordination
    protocols. ............................................................................................. 246
7.13 The NFM $G$, where the states in the dashed box are pruned. ............ 251
TABLES

3.1 $T^0$ in Example 3.2.1 ............................................. 48
3.2 $T^1$ in Example 3.2.1 ............................................. 49
3.3 $T^2$ in Example 3.2.1 ............................................. 50

4.1 Mission Events of the Robots. ................................. 90

5.1 Events of the Multi-robot System ............................. 137

6.1 RS Path Classes ...................................................... 189
6.2 RS vs. CC Feasibility ............................................... 218
ACKNOWLEDGMENTS

I am so grateful to have the support and encouragement from so many people throughout the journey of my PhD program. This dissertation would not have been possible without their assistance and advice.

First of all, I would like to express my sincere gratitude to my advisor, Professor Hai Lin, who directed me to the world of Cyber-Physical Systems and fostered my interest in this research issue, for his encouragement and guidance during my PhD study and during my accomplishment of this thesis. I benefit a lot from his broad knowledge in analysis and control of cyber-physical systems and his genuine eagerness for pursuing high-quality research. He is also a life mentor who has taught me how to balance my academic study and personal lives.

Next, I would like to thank Professor Michael Lemmon, Professor Yih-fang Huang and Professor Taeho Jung for accepting to sit on my dissertation committee. Their insightful comments guided me to consider my research problem from different perspectives and make further improvements.

My deep gratitude also goes to my colleagues, Doctor Ali Karimoddini, Doctor Wu Bo, Alireza Partovi, Zhiyu Liu and Wei Zheng, for their fruitful discussions on Cyber-physical Systems and their companies in the Distributed Cooperative System Research (DISCOVER) Laboratory.

I wish to acknowledge the financial support from the National Science Foundation (Grant No. CNS-1446288, ECCS-1253488, IIS-1724070), the Army Research Laboratory (Grant No. W911NF-17-1-0072), and the Graduate School at the University of Notre Dame.
Finally, I sincerely thank my parents, Jin Jin and Hua Dai. This dissertation would not have been finished without their constant support, love and sacrifice. My gratitude to them is beyond what I can express in words. I owe my wife, Doctor Ruiwen Zhen, too much in past seven years. I would like to express my deepest gratitude to Ruiwen for her love, support and sacrifice. I love you.
1.1 Background Information and Research Motivation

1.1.1 Research Motivation

Nowadays, the rapid development of communication networks, control algorithms and information technologies (IT) has greatly enhanced the ubiquitous deployment of large-scale engineering systems with sophisticated architectures. These engineering systems, which involve not only logic-based control algorithms that are implemented via spatially-distributed communication networks, but also physical dynamical processes equipped with various sensors and actuators, are called Cyber-Physical Systems (CPSs) [172]. More specifically, the communication networks and embedded controllers that regulate the behaviors of the physical processes form the cyber part of a CPS, whereas the dynamic processes that characterize the physical evolution of the CPS affect the cyber computations as well. To this regard, behaviors of such kind of CPSs are determined by both logical and physical parts of the systems [164]. There is a large variety of applications of CPSs, and examples include, but not limited to, networked mobile robots, autonomous vehicles, intelligent transportation systems, flexible manufacturing systems and so on.

Compared to the conventional automated control systems as shown in Fig. 1.1(a), the deployment of the IT infrastructures in today’s large-scale CPSs has enabled timely collaborations between controllers, sensors and actuators, which makes persistent monitoring and control of these systems possible. Nevertheless, the application of heterogeneous system components and IT cyber-infrastructures has also resulted in more sophisticated
system architectures, introducing more vulnerabilities to system uncertainties, unexpected faults, cyber-attacks with malicious intentions and safety threats from the environment that may cause undesired consequences [175], posing new challenges on guaranteeing safety, operational efficiency and security properties of these systems.

![Diagram](image.png)

Figure 1.1. General architectures of automated and autonomous systems.

Two fundamental problems have drawn a great amount of research interest in the context of CPSs, namely the verification and the synthesis problems. Facing a CPS and a certain performance specification, we need first to justify whether or not the specification can be satisfied by the system in various environments. When the answer to the verification problem is negative, a next step is how to synthesize an appropriate controller that enforces the fulfillment of the specification despite the uncertainties from the environment. To pursue a “scalable” and “provably correct” design of complex CPSs, model-based formal approaches are widely applied in this context [10, 120]. In general, model-based approaches start with a mathematical abstraction of the cyber part of the underlying CPS, based on which decision-making procedures are performed and then the control policies are implemented as hybrid controllers on the physical part. Despite the great success in
steering automated CPSs to meet formal specifications, the effectiveness of model-based approaches highly rely on the accessibility of precise abstract models of both the system and the given specifications. In other words, when dealing with a dynamic environment with potential uncertainties, obtaining a precise model that accounts for the system’s behaviors \textit{a priori} can be difficult, which severely limits the applicability of model-based design approaches.

The eagerness to achieve complex formal specifications of large-scale CPSs motivates us to go beyond automated systems and study the \textit{autonomous} CPSs (ACPSs) whose architectures are illustrated in Fig. 1.1(b), in which the CPSs under consideration need to appropriately interact with various types of environments. In this dissertation, we aim at tackling the research challenges in the coordination and control of distributed ACPSs by developing novel methodologies of synthesizing both local control policies and coordination strategies among different subsystems so as to satisfy a formal specification in the presence of either environmental uncertainties or unexpected system faults.

![Figure 1.2. The hierarchical control architecture for each subsystem.](image)

Our study of distributed ACPSs starts with associating each subsystem of a given ACPS
with a hierarchical control architecture, which is depicted in Fig. 1.2. The presented architecture consists of a coordination layer that communicates with other subsystems to obtain local control objectives, a supervision layer that generates sequences of control actions (events) to be executed so that the local specification can be achieved, a planning layer that translates the control actions into desired physical state trajectories, and a regulation layer that interconnects directly steers the physical part of the system to track the trajectory.

Figure 1.3. A general system architecture of autonomous cyber-physical systems.

Fig. 1.3 illustrates the working procedure of the hierarchical control architecture associated with each subsystem as follows. At the regulation layer, sensor readings that monitor the status of both the physical dynamics and the environment are processed to generate control signals for actuators to regulate the subsystem’s local status around known set points with respect to the physical dynamics (usually represented in form of continuous-time d-
ifferential equations or discrete-time difference equations) and the constraints posed by the environment. The set points are given from the planning layer, at which optimization-based algorithms are executed to determine feasible or optimal trajectories consisting of such set points, while the cost functions of the optimization processes are obtained from the supervision layer above. The supervision layer basically solves discrete-event supervisory control problems to generate a coarse (usually discrete) sequence of control symbols such that a given logic specification is satisfied, while the logic specification is assigned by the top coordination layer based on the global control objectives and each subsystem’s individual sensing and actuation capabilities.

1.1.2 Overview of the Dissertation

In this dissertation, we model the cyber part (i.e., the coordination and the supervision layer) of each subsystem as a Discrete Event Systems (DES), which represents a class of dynamic systems that evolve according to the occurrence of certain discrete qualitative changes, called events [26, 91]. The event-driven nature of a DES makes it a good tool for analysis and control for ACPSs with complex architectures, since finite abstraction and approximate abstraction techniques [87, 173] have been widely applied in practice for extracting finite transition models of dynamic systems from the continuous-state and continuous-time domains. Furthermore, to compensate for the drawbacks of conventional model-based design approaches and to address uncertainties of both the model of the (sub)system and the environment, the idea of formal inductive synthesis (FIS) [3, 76], which is an algorithmic machine learning process of synthesizing a system with formal guarantees from either examples or counterexamples, has also been leveraged in the context of formal synthesis. In the FIS procedure, a Learner can iteratively accomplish the synthesis objectives by presenting different queries and conjectures to a Teacher. By integrating the FIS procedure with existed formal verification and synthesis techniques, we are able to develop a coordination and control framework for distributed ACPSs that can
achieve a formal specification with resilience and security guarantees.

Four major problems of distributed ACPSs are investigated in this dissertation. First, we consider the formal mission and motion planning and coordination problem of multi-agent systems, in which we propose an automatic synthesis framework to synthesize appropriate local mission supervisors for each agent so that a global specification can be accomplished. Furthermore, we associate synthesized mission plans with appropriate motion plans for each agent in order to address potential uncertainties arisen in the real environment.

Next, we study the resilience of the proposed framework in the presence of unexpected actuator and sensor faults. We model actuator faults as local controllability loss of certain actuator events and sensor faults as observability failure of certain sensor readings, respectively. The fault tolerance property requires that the nominal and fault-pruned subsystems coordinate so as to fulfill the control specification after occurrences of the faults. We propose an active approach for the FTC problem of the distributed DESs subject to possible loss of actuating and sensing capabilities.

Thirdly, we are also interested in the path planning problem of mobile robots and/or vehicles from the perspective of regulating the physical part of an ACPS. Based on the \( \mu \)-tangency properties, we establish geometric conditions under which sub-optimal continuous curvature paths may exist to connect two configurations. Computationally efficient continuous curvature path planning algorithms can then be extracted from the presented existence conditions.

Finally, we investigate the cyber-security of the underlying CPS by studying opacity-enforcement problems in the presence of multiple intruders, each of which is assumed to have full prior knowledge of the system model but can only partially observe the behaviors of the system. An enforcement mechanism is developed based on insertion functions to assure decentralized opacity when no coordination exists among the intruders, and a centralized coordination and refinement procedure is proposed to construct local insertion functions associated with each intruder’s observation capabilities such that joint opacity
can be guaranteed when the intruders may coordinate via an intersection-based protocol.

1.2 Organization and Main Contributions

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

Chapter 2 introduces necessary preliminaries and notations for formal verification and synthesis methods in distributed ACPSs. First, we review the basic concepts in formal languages and automata theory, which are frequently used as the tool to specify the behaviors performed by distributed ACPSs in the following chapters. Next, a brief review of the $L^*$ algorithm [2], which aims to learn an unknown regular language over a given event set via examples and counterexamples, is presented. Afterwards, the Ramadge-Wonham supervisory control theory for discrete event systems (DESs) is reviewed. Finally, the compositional verification and the assume-guarantee reasoning paradigms are briefly reviewed.

Chapter 3 deals with the automatic supervisor synthesis problems for discrete event systems. By modifying the $L^*$ learning algorithm [2], we develop learning-based online supervisor synthesis algorithms [42, 43], and prove that the proposed algorithms converge to the supremal controllable sublanguage of the given prefix-closed specification language. Furthermore, automatic synthesis of supervisors for non-terminating specifications is also studied in this chapter. A necessary and sufficient condition for the existence of the supervisors is presented, and a learning-based algorithm for computing the supremal $\omega$-controllable sublanguage of the given $\omega$-regular specification language is proposed. The correctness and finite convergence of the proposed algorithm are proved, and their effectiveness is validated through illustrative examples.

We leverage the idea of learning-based synthesis [41, 47, 190] and propose a formal design framework for synthesizing integrated mission and motion plans for cooperative multi-agent systems to accomplish global missions in Chapter 4. On the one hand, by modeling the mission execution capabilities of each agent as a discrete event processes, a hierarchical planning framework is carried out to accomplish global missions that are spec-
ified as declarative regular languages. More specifically, we first synthesize a local mission supervisor for each agent based on the agent’s mission execution capabilities; afterwards, the supervised agents are coordinated via an assume-guarantee approach so that the global mission can be fulfilled. On the other hand, based on a formal model of the environment shared by the agents, motion plans are automatically computed for each agent corresponding to the intended execution of missions. We also leverage the idea of inductive learning and present three modifications of the \( L^* \) learning algorithm so that the local mission and motion plans can be successfully synthesized in the presence of uncertainties of the global mission specification and/or the environment. A multi-robot coordination example is presented in this paper to illustrate the proposed planning framework.

Chapter 5 aims at investigating the coordination and control of distributed discrete event systems that are composed of subsystems subject to actuator and/or sensor faults \([44, 45]\). We model actuator faults as local controllability loss of certain actuator events and sensor faults as observability failure of certain sensor readings, respectively. Starting from automata-theoretic models that characterize behaviors of the subsystems in the presence of faulty actuators and/or sensors, we establish necessary and sufficient conditions for the existence of actuator and sensor fault tolerant supervisors, respectively, and synthesize appropriate local post-fault supervisors to prevent the post-fault subsystems from jeopardizing local safety requirements. Furthermore, we apply an assume-guarantee coordination scheme to the controlled subsystems for both the nominal and faulty subsystems so as to achieve the desired specifications of the system. A multi-robot coordination example is used to illustrate the proposed coordination and control architecture.

In addition to the synthesis of formal control policies, a successful design of an autonomous CPS also requires appropriate controllers that deals with the physical part of the system. To this end, Chapter 6 takes motion planning problem of mobile robots as an example and studies the continuous-curvature (CC) steering problem \([46, 48]\) for robots subject to nonholonomic dynamics and constraints on velocity, curvature along the path.
and derivative of the curvature. Based on the $\mu$-tangency property, we establish existence conditions for a class of CC paths which admit the same driving patterns as the Reeds-Shepp paths. These conditions allow efficient implementation of the CC steering, which enables real-time CC path planning. The feasibility and computation efficiency of the resultant CC steering are validated by numerical simulations.

Chapter 7 studies the cyber-security of the CPSs in terms of the opacity property [191, 196]. Opacity is a confidentiality property that characterizes the non-disclosure of specified secret information of a system to an outside observer (termed as an intruder). In this chapter, we study the opacity enforcement problems within the discrete event system formalism in the presence of multiple intruders. We study two cases, one without coordination among the intruders and the other with coordination. We propose appropriate notions of opacity corresponding to the two cases, respectively, and propose enforcement mechanisms for these opacity properties based on the implementation of insertion functions, which manipulates the output of the system by inserting fictitious observable events whenever necessary. The insertion mechanism is adapted to the decentralized framework to enforce opacity when no coordination exists. Furthermore, we present a coordination and refinement procedure to synthesize appropriate insertion functions to enforce opacity when intruders may coordinate with each other by following an intersection-based coordination protocol. The effectiveness of the proposed opacity-enforcement approaches is validated through illustrative examples.

Chapter 8 summarizes the main contributions of this dissertation and discusses several potential research directions for future work.
CHAPTER 2

PRELIMINARIES

2.1 Introduction

The prerequisite definitions and notations that are related to the study of distributed ACPSs are briefly reviewed in this chapter. First, we review the basic concepts in formal languages and automata theory, which are widely used to characterize the logical behaviors of ACPSs at the cyber level. Next, a brief review of the $L^*$ learning algorithm [2], an inductive inference algorithm that learns an unknown regular language over a given event set via examples and counterexamples, is reviewed. Afterwards, we briefly introduce the compositional verification and the assume-guarantee reasoning paradigm. Finally, the Ramadge-Wonham supervisory control theory for discrete event systems (DESs) is reviewed.

The remainder of this chapter is organized as follows. In Section 2.2, we introduce the fundamentals of formal languages and automata theory. Section 2.3 introduces the $L^*$ learning algorithm that is presented in [2] with an illustrative example. In Section 2.4, an essential introduction of the compositional verification procedures in formal verification [7, 96] is presented and the assume-guarantee verification paradigms are reviewed. Interested readers are referred to [2, 7, 35, 36, 73, 122, 132] for more details.
2.2 Basics of Automata Theory and Formal Languages

2.2.1 Finite Automata and Regular Languages

Let \( \Sigma \) be a finite set of event symbols, and let \( 2^\Sigma \) and \( |\Sigma| \) denote the power set and cardinality of \( \Sigma \), respectively. For two event sets \( \Sigma_1 \) and \( \Sigma_2 \), \( \Sigma_1 - \Sigma_2 \) denotes the set-theoretic difference of \( \Sigma_1 \) and \( \Sigma_2 \); whereas \( \Sigma_1 \Delta \Sigma_2 := (\Sigma_1 - \Sigma_2) \cup (\Sigma_2 - \Sigma_1) \). \( \Sigma_1 \Sigma_2 = \{ \sigma_1 \sigma_2 | \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2 \} \), where \( \sigma_1 \sigma_2 \) represents the concatenation of two elements \( \sigma_1 \) and \( \sigma_2 \). A finite sequence \( w = \sigma_1 \sigma_2 \ldots \sigma_m \) composing of elements in \( \Sigma \) is called a string over \( \Sigma \). We denote \( w[i] \) (\( i \in \mathbb{N} \)) as the \( i \)-th event symbol of the string \( w \). The length of a string \( w \) over \( \Sigma \), written as \( |w| \), is the number of event symbols appeared in \( w \). Let \( \Sigma^* \) denote the set of all finite-length strings over \( \Sigma \) plus the empty string \( \epsilon \). A subset \( L \) of \( \Sigma^* \) is called a language over \( \Sigma \). The prefix-closure \( \overline{L} \) of \( L \subseteq \Sigma^* \) is defined as \( \overline{L} = \{ s \in \Sigma^* | (\exists t \in \Sigma^*)[st \in L] \} \). A language \( L \) is said to be prefix-closed if \( \overline{L} = L \). \( L \) is said to be deadlock-free (live) [91] if for all \( s \in \overline{L} \), there exists \( \sigma \in \Sigma \) such that \( s\sigma \in \overline{L} \). For two languages \( L_1, L_2 \subseteq \Sigma^* \), the left quotient (or quotient for short) is the collection of prefixes of strings in \( L_1 \) with a suffix that belongs to \( L_2 \), i.e., \( L_1 / L_2 = \{ s \in \Sigma^* | (\exists t \in L_2)[st \in L_1] \} \). The right quotient of \( L_1 \) and \( L_2 \), written as \( L_1 \setminus L_2 = \{ w \in \Sigma^* | (\exists s \in L_2)[sw \in L_1] \} \), collects the strings in \( L_1 \) that admit a prefix in \( L_2 \). Specifically, if \( L_2 = \{ s \} \), then we abbreviate \( L_1 \setminus L_2 \) as \( L_1 \setminus s \), which denotes the post-language of \( L_1 \) after a string \( s \in \Sigma^* \) takes place.

The definition of finite automata is recalled to recognize languages.

**Definition 2.1** (Non-deterministic Finite Automaton). [7] A non-deterministic finite automaton (NFA), denoted by \( G \), is a 5-tuple

\[
G = (Q, \Sigma, \delta, Q_0, Q_m),
\]

where \( Q \) is a finite set of states, \( \Sigma \) is the set of events, \( \delta : Q \times \Sigma \cup \{ \epsilon \} \rightarrow 2^Q \) is a partial transition function (it is a partial function since it is generally defined on a subset of \( Q \times \Sigma \cup \{ \epsilon \} \)), \( Q_0 \subseteq Q \) is a set of initial states, and \( Q_m \subseteq Q \) is a set of the marked states.
The finite automaton $G$ evolves as follows. It starts from one initial state $q_0 \in Q_0$ and upon the occurrence of a feasible event $\sigma \in \Sigma \cup \{\epsilon\}$ at $q_0$ (i.e., $\delta(q_0, \sigma) \neq \emptyset$), it makes a transition to one of the states in $\delta(q_0, \sigma)$. The above process continues based on the transitions defined by $\delta$.

The transition function $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$ of the NFA $G$ can be extended to $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ inductively: $\forall q \in Q, s \in \Sigma^*, \sigma \in \Sigma,

\delta^*(q, \epsilon) := \epsilon_G^*(q),

\delta^*(q, s\sigma) := \bigcup_{q' \in \delta(\delta^*(q, s), \sigma)} \epsilon_G^*(q'),

where $\epsilon_G^*(q) \subseteq Q$ denotes the “$\epsilon$-closure” of the state $q \in Q$, which is the set of states that are reachable from $q$ on zero or more $\epsilon$-moves [73]. For the sake of simplicity, from now on, we use the notations $\delta$ and $\delta^*$ interchangeably.

$G$ is said to be a deterministic finite automaton (DFA) if: (i) $|Q_0| \leq 1$; (ii) $(\forall q \in Q, \sigma \in \Sigma)[|\delta(q, \sigma)| \leq 1]$. When $G$ is deterministic, the transition function $\delta$ can be rewritten as $\delta : Q \times \Sigma \rightarrow Q$ and can be extended to $\delta : Q \times \Sigma^* \rightarrow Q$ in the usual manner. The DFA $G$ is said to be complete (total) if: (i) $|Q_0| = 1$; (ii) $(\forall q \in Q, \sigma \in \Sigma)[|\delta(q, \sigma)| = 1]$. We omit $Q_m$ if $Q_m = Q$. For each state $q \in Q$, we denote by $\text{Act}_G(q) := \{\sigma \in \Sigma | \delta(q, \sigma) \neq \emptyset\}$ the set of active events defined at $q$. The DFA $G$ is said to be accessible if $(\forall q \in Q)[\exists s \in \Sigma^* : \delta(q, s) = q]$; $G$ is said to be coaccessible if $(\forall q \in Q)[\exists q_m \in Q_m, s \in \Sigma^* : \delta(q, s) = q_m]$; $G$ is said to be trim if it is both accessible and coaccessible [26].

A run of a DFA $G$ on a finite string $w = \sigma_1\sigma_2\ldots\sigma_m \in \Sigma^*$ is a sequence of states $\text{Run}(w) = q_0q_1\ldots q_m \in Q^*$ satisfying $q_0 \in Q_0$ and $q_i = \delta(q_{i-1}, \sigma_i)$ ($i = 1, 2, \ldots, m$). The language generated by a finite automaton $G$ is given by

$L(G) = \{s \in \Sigma^* | \exists q_0 \in Q_0, \delta(q_0, s) \neq \emptyset\}, \quad (2.2)$
where by \( \delta(q_0, s) \) we mean that the transition \( \delta(q_0, s) \) is defined. The language accepted (recognized) by \( G \) is denoted as

\[
L_m(G) = \{ s \in \Sigma^* | \exists q_0 \in Q_0, \delta(q_0, s) \cap Q_m \neq \emptyset \}.
\]

For a string \( t \in \Sigma^* \), we denote by DFA(\( t \)) the string DFA corresponding to \( t \), which is a trim DFA that satisfies: (i) \( L(DFA(t)) = \overline{t} \); (ii) \( L_m(DFA(t)) = t \).

A finite automaton \( G \) is said to be non-blocking if \( \overline{L_m(G)} = L(G) \). Two finite automata \( G_1 \) and \( G_2 \) with the same event set are said to be equivalent if \( L_m(G_1) = L_m(G_2) \). The following theorem states that the classes of NFA and DFA are equivalently expressive in terms of recognizing languages.

**Theorem 2.1.** [7] For a non-deterministic finite automaton \( G \), there always exists a deterministic finite automaton \( G_d \) such that \( G \) and \( G_d \) are equivalent.

A language \( L \subseteq \Sigma^* \) is said to be regular if there exists a DFA \( G \) such that \( L(G) = \overline{L} \) and \( L_m(G) = L \) [26]. Throughout the rest of this dissertation, we focus our study on regular languages.

We then proceed to the operations on finite automata and regular languages. We first introduce the following parallel composition that captures the joint operations of two synchronously interconnected automata.

**Definition 2.2** (Parallel Composition). [7] Given two finite automata \( G_i = (Q_i, \Sigma_i, \delta_i, Q_{i,0}, Q_{i,m}) \), \( i = 1, 2 \), the parallel composition of \( G_1 \) and \( G_2 \), denoted as \( G_1 \parallel G_2 \), is defined as the finite automaton \( G_1 \parallel G_2 = (Q, \Sigma, \delta, Q_0, Q_m) \) where

- \( Q = Q_1 \times Q_2 \) is the set of states;
- \( \Sigma = \Sigma_1 \cup \Sigma_2 \) is the set of events;
- \( \delta : Q \times \Sigma \cup \{\epsilon\} \to 2^Q \) is a partial transition function defined as: \( \forall q = (q_1, q_2) \in \)
$Q, \sigma \in \Sigma$

$$\delta(q, \sigma) := \begin{cases} \delta_1(q_1, \sigma) \times \delta_2(q_2, \sigma), & \text{if } \delta_1(q_1, \sigma) \neq \emptyset, \delta_2(q_2, \sigma) \neq \emptyset, \\ \delta_1(q_1, \sigma) \times \{q_2\}, & \text{if } \delta_1(q_1, \sigma) \neq \emptyset, \sigma \notin \Sigma_2, \\ \{q_1\} \times \delta_2(q_2, \sigma), & \text{if } \delta_2(q_2, \sigma) \neq \emptyset, \sigma \notin \Sigma_1, \\ \emptyset, & \text{otherwise}, \end{cases}$$

$$\delta(q, \varepsilon) = [\delta_1(q_1, \varepsilon) \times \{q_2\}] \times [\{q_1\} \times \delta_2(q_2, \varepsilon)];$$

- $Q_0 = Q_{1,0} \times Q_{2,0}$ is the set of initial states;
- $Q_m = Q_{1,m} \times Q_{2,m}$ is the set of marked states.

In a parallel composition of two automata, the two automata are “synchronized” on the common events in $\Sigma_1 \cap \Sigma_2$. The other events in $\Sigma_1 \cup \Sigma_2$ are not subject to such a constraint and can be executed whenever possible. Note that this composition of more than two finite automata can be defined recursively based on the associativity of parallel composition.

In addition to parallel composition, we also introduce a unitary operation called completion in order to make a given DFA complete.

**Definition 2.3 (Completion).** Given $G = (Q, \Sigma, \delta, q_0, Q_m)$ with an “error” state $q_e \notin Q$, the completion of $G$ is defined as a DFA $\tilde{G} = (\tilde{Q}, \Sigma, \tilde{\delta}, q_0, Q_m)$ with $\tilde{Q} = Q \cup \{q_e\}$, and

$$\forall \tilde{q} \in \tilde{Q}, \sigma \in \Sigma, \tilde{\delta}(\tilde{q}, \sigma) = \begin{cases} \delta(\tilde{q}, \sigma), & \text{if } \tilde{q} \in Q \land \delta(\tilde{q}, \sigma)!, \\ q_e, & \text{otherwise}. \end{cases}$$

It is clear from Definition 2.3 that the completion $\tilde{G}$ of $G$ is complete, i.e., $L(\tilde{G}) = \Sigma^*$ and $L_m(\tilde{G}) = L_m(G)$. The complement of a DFA $G$ over $\Sigma$, written as $\text{co}G$, is a DFA such that $L_m(\text{co}G) := \Sigma^* - L_m(G)$ and can be constructed by swapping the marked states of $\tilde{G}$ with its non-marked states and vice versa, i.e., $\text{co}G = (\tilde{Q}, \Sigma, \tilde{\delta}, q_0, \tilde{Q} - Q_m)$.

Given a subset $\Sigma' \subseteq \Sigma$, the operation of “natural projection” from strings over $\Sigma$ to strings over $\Sigma'$ is formally defined as follows.
Definition 2.4 (Natural Projection). [143] For any subset $\Sigma' \subseteq \Sigma$, the natural projection from $\Sigma^*$ to $\Sigma'^*$ is a mapping $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ such that: (i) $P_{\Sigma'}(\epsilon) = \epsilon$; (ii) $P_{\Sigma'}(\sigma) = \sigma$ if $\sigma \in \Sigma'$ and $P_{\Sigma'}(\sigma) = \epsilon$ otherwise; (iii) $P_{\Sigma'}(s\sigma) = P_{\Sigma'}(s)P_{\Sigma'}(\sigma)$, $\forall s \in \Sigma^*, \sigma \in \Sigma$.

The inverse projection $P_{\Sigma'}^{-1} : \Sigma'^* \to 2^{\Sigma^*}$ corresponding to $P_{\Sigma'}$ is defined as $P_{\Sigma'}^{-1}(t) = \{ s \in \Sigma^* | P_{\Sigma'}(s) = t \}$. The projection $P_{\Sigma'}$ and its inverse $P_{\Sigma'}^{-1}$ can be extended to a language by applying them to all the strings that belong to the language.

The operation of natural projection is further used to define the following “property satisfaction” relation.

Definition 2.5 (Property Satisfaction). [132] Given a system that is modeled by a DFA $M = (Q, \Sigma_M, \delta, q_0, M, Q_m, M)$ and a property $P$ that can be represented by the marked language of a DFA $P = (Q_P, \Sigma_P, \delta_P, q_0, P, Q_m, P)$ satisfying $\Sigma_P \subseteq \Sigma_M$, the system $M$ is said to satisfy the property $P$, written as $M \models P$, if and only if $\forall t \in L_m(M) : P_P(t) \in L_m(P)$, where $P_P$ is the natural projection from $\Sigma_M^*$ to $\Sigma_P^*$. In case that $P$ is prefix-closed, the satisfaction relation holds if and only if $\forall t \in L(M) : P_P(t) \in L(P)$.

Remark 2.1. Note that when $\Sigma_M \subseteq \Sigma_P$, Definition 2.5 reduces to the language inclusion $L_m(M) \subseteq L_m(P)$.

Let $I = \{1, 2, \ldots, n\}$ be an index set. For each $i \in I$, we consider $\{\Sigma_i\}_{i \in I}$ as a set of local event sets. The global event set is written as $\Sigma = \bigcup_{i \in I} \Sigma_i$. Let $P_i(i \in I)$ be the natural projection from $\Sigma^*$ to $\Sigma_i^*$ for each $i \in I$. The synchronous product of a finite set of regular languages $L_i \subseteq \Sigma_i^* (i \in I)$, denoted as $\|_{i=1}^n L_i$, is defined as follows

Definition 2.6 (Synchronous Product). [187] For a finite set of regular languages $L_i \subseteq \Sigma_i^* (i \in I)$, the synchronous product $\|_{i=1}^n L_i$ is given by

$$\|_{i \in I} L_i = \{ t \in \Sigma^* | (\forall i \in I) [P_i(t) \in L_i] \}.$$ (2.4)
Clearly, (2.4) is equivalent to
\[
\|_{i=1}^n L_i = \bigcap_{i=1}^n P_i^{-1}(L_i). \tag{2.5}
\]

From Definition 2.2 and Definition 2.6, for a composed automaton \( G = \|_{i \in I} G_i \) with the local event sets \( \Sigma_i (i \in I) \), it holds that
\[
L(G) = L(\|_{i \in I} G_i) = \|_{i \in I} L(G_i) = \bigcap_{i \in I} P_i^{-1}(L(G_i)). \tag{2.6}
\]

The notion of language separability plays an essential role in the following chapters. Intuitively, a language is separable if it can be retrieved by the synchronous product of a set of “component languages”. Formally, the definition of separability is stated as follows:

**Definition 2.7 (Separable Languages).** \[187\] For the local event sets \( \Sigma_i (i \in I) \) and the global event set \( \Sigma = \bigcup_{i \in I} \Sigma_i \), a language \( L \subseteq \Sigma^* \) is said to be separable with respect to \( \{\Sigma_i\}_{i \in I} \) if there exists a set of local languages \( L_i \subseteq \Sigma_i^* (i \in I) \) such that \( L = \|_{i \in I} L_i \).

If \( L \) is separable with respect to \( \{\Sigma_i\}_{i \in I} \), then the component languages \( L_i (i \in I) \) in Definition 2.7 form a generating set of \( L \). For a separable language, there may exist several generating sets. Let \( B(L) \) be the set of all generating sets of a separable language \( L \subseteq \Sigma^* \), i.e.,
\[
B(L) := \{ (L_{1\alpha}, L_{2\alpha}, \ldots, L_{n\alpha}) | (\forall \alpha, \forall i \in I) [L_{i\alpha} \subseteq \Sigma_i^* \land L = \|_{i \in I} L_{i\alpha}] \}. \tag{2.7}
\]

The following proposition states that the elements of \( B(L) \) is closed under intersection.

**Proposition 2.1.** \[187\] For each \( i \in I \), the non-empty family \( \{ L_{i\alpha} \} \) is closed under arbitrary intersection and
\[
L_{i, \text{inf}} := \bigcap_{\alpha} L_{i\alpha} = P_i(L). \tag{2.8}
\]
An immediate corollary can then be drawn from Proposition 2.1 as follows, stating that the separability of the language $L$ can be checked through the synchronous product of its local projections.

**Corollary 2.1.** [187] A language $L \subseteq \Sigma^*$ is separable with respect to $\{\Sigma_i\}_{i \in I}$ if and only if $L = \|_{i \in I} P_i(L) = \cap_{i \in I} P_i^{-1}[P_i(L)]$.

It can be inferred from Corollary 2.1 that if $(L_1, L_2, \ldots, L_n) \in B(L)$, then for each $i \in I$, $(L_1, L_2, \ldots, L_{i-1}, L_{i,\inf}, L_{i+1}, \ldots, L_n) \in B(L)$.

The next proposition states that for each $i \in I$, the family $\{L_{i,\alpha}\}$ is closed under union and hence admits a unique supremal element.

**Proposition 2.2.** [187] For each $i \in I$, the non-empty family $\{L_{i,\alpha}\}$ is closed under arbitrary union and

$$L_{i,\sup} := \bigcup_\alpha L_{i,\alpha} = P_i(L) \cup [\Sigma_i^* - P_i(\hat{L}_i)], \quad (2.9)$$

where $\hat{L}_i = \bigcap_{j \in I - \{i\}} P_j^{-1}[P_j(L)]$.

Similar to Corollary 2.1, if $(L_1, L_2, \ldots, L_n) \in B(L)$, then for each $i \in I$,

$$(L_1, L_2, \ldots, L_{i-1}, L_{i,\inf}, L_{i+1}, \ldots, L_n) \in B(L).$$

We conclude this section with the following proposition.

**Proposition 2.3.** [187] The set of all separable languages (with respect to $\{\Sigma_i\}_{i \in I}$), over $\Sigma$, is closed under arbitrary intersection.

**Remark 2.2.** In general, it is undecidable whether or not there exists a non-empty separable sublanguage of $L$. For the local event sets $\Sigma_i$ ($i \in I$) and the global event set $\Sigma = \bigcup_{i \in I} \Sigma_i$, let $D(\Sigma) = \{(\sigma_1, \sigma_2) \in \Sigma \times \Sigma | \exists i \in I, \sigma_1, \sigma_2 \in \Sigma_i\}$. The independence relation is defined as $I(\Sigma) = \Sigma \times \Sigma - D(\Sigma)$ [112]. It follows from [112] that the existence of a non-empty separable sublanguage of $L$ is decidable if and only if $I(\Sigma)$ is transitive.
2.2.2 Büchi Automata and \(\omega\)-regular Languages

In addition to the set \(\Sigma^*\) of the finite-length strings over \(\Sigma\), we denote by \(\Sigma^\omega\) the set of infinite-length strings, namely \(\omega\)-strings, over an event alphabet \(\Sigma\), i.e.,

\[
\Sigma^\omega = \{ w | w = \sigma_1\sigma_2\sigma_3 \cdots , \forall m \in \mathbb{N}, \sigma_m \in \Sigma \}.
\]

An \(\omega\)-language over \(\Sigma\) is a subset \(L \subseteq \Sigma^\omega\). Given a regular language \(L \subseteq \Sigma^*\), its limit language, written as \(\lim L\), is an \(\omega\)-language that admits infinitely many finite prefixes in \(L\).

**Definition 2.8** (Limit Language). *For a regular language \(L \subseteq \Sigma^*\),

\[
\lim L = \{ w \in \Sigma^\omega | (\exists \text{ infinitely many } n \in \mathbb{N})[w^n \in L] \},
\]

(2.10)

where \(w^n = w_1w_2\ldots w_n\) represents a prefix of the \(\omega\)-string \(w\) with \(|w^n| = n\).

We define the following metric to compare the similarity of two \(\omega\)-strings over \(\Sigma\).

**Definition 2.9** (Distance between \(\omega\)-strings). [141] Given two infinite strings \(e_1, e_2 \in \Sigma^\omega\), the distance \(\rho(e_1, e_2)\) between the two infinite strings is defined to be

\[
\rho(e_1, e_2) = \begin{cases} 
\frac{1}{n}, & \text{if } e_1^{n-1} = e_2^{n-1} \land e_1^n \neq e_2^n, \\
0, & \text{if } e_1 = e_2,
\end{cases}
\]

(2.11)

where \(e_1^n\) and \(e_2^n\) represent the prefix of size \(n\) of \(e_1\) and \(e_2\), respectively.

Note that we can define a metric space \((\Sigma^\omega, \rho)\) based on the metric presented in Definition 2.9.

Let \(\Sigma^\infty = \Sigma^* \cup \Sigma^\omega\). For two strings \(w \in \Sigma^\infty, r \in \Sigma^*\), \(r\) is said to be a strict prefix of \(w\), written as \(r < w\), if there exists \(v \in \Sigma^\infty\) such that \(w = rv\). Given an monotone and infinite sequence of strings \(s_1 < s_2 < \ldots < s_n < \ldots\) with \(s_n \in \Sigma^*\) for each \(n \in \mathbb{N}\), there exists a
unique $\omega$-string $e \in \Sigma^\omega$ such that $s_n < e$ holds for each $n$. In this case, the $\omega$-string $e$ can also be written as $e = \lim_{n \to \infty} s_n$. With slightly abusing the notations, we still denote $\bar{\mathcal{L}}$ as the prefix-closure of a given $\omega$-language $\mathcal{L} \subseteq \Sigma^\omega$, which is defined as

$$\bar{\mathcal{L}} = \{ s \in \Sigma^* | (\exists w \in \mathcal{L}) [s < w] \}. \quad (2.12)$$

The following proposition provides the necessary and sufficient condition for a regular language to agree on the prefix closure with its limit $\omega$-regular language.

**Proposition 2.4.** [91] Consider a regular language $L \subseteq \Sigma^*$, then $\lim L = \mathcal{L}$ if and only if $L$ is deadlock-free; moreover, $\lim L = L$ if and only if $L$ is deadlock-free and prefix-closed.

The topological closure (shortly closure) [56] of an $\omega$-language $\mathcal{L} \subseteq \Sigma^\omega$ can be defined with respect to the aforementioned metric space $(\Sigma^\omega, \rho)$.

$$\text{clo}(\mathcal{L}) = \lim \bar{\mathcal{L}}. \quad (2.13)$$

The $\omega$-language $\mathcal{L}$ is said to be topologically closed if $\mathcal{L} = \text{clo}(\mathcal{L})$ [141]. Given two $\omega$-languages $\mathcal{L}, \mathcal{H} \subseteq \Sigma^\omega$, we say that $\mathcal{L}$ is closed relative to $\mathcal{H}$ if $\mathcal{L} = \text{clo}(\mathcal{L}) \cap \mathcal{H}$. It is clear that when $\mathcal{L} \subseteq \mathcal{H}$, topological closeness of $\mathcal{L}$ implies the closeness of $\mathcal{L}$ relative to $\mathcal{H}$ [92].

An $\omega$-automaton (or stream automaton) is a variation of finite automaton that runs on infinite, rather than finite strings. The difference between $\omega$-automata and conventional finite automata lies on the acceptance conditions. We focus on a special class of $\omega$-automata, called Büchi automata, which is defined as follows.

**Definition 2.10 (Non-deterministic Büchi Automaton).** [7] A non-deterministic Büchi automaton (NBA), denoted by $G$, is a 5-tuple

$$G = (Q, \Sigma, \delta, Q_0, F), \quad (2.14)$$

where $Q$ is a finite set of states, $\Sigma$ is a finite set of events, $\delta : Q \times \Sigma \cup \{\epsilon\} \to 2^Q$ is a partial
transition function, \( Q_0 \subseteq Q \) is a set of initial states, \( F \subseteq Q \) is the set of accept states.

A run over an \( \omega \)-string \( w = \sigma_1 \sigma_2 \sigma_3 \ldots \in \Sigma^\omega \) denotes an infinite sequence of states \( \text{Run}(w) = q_0 q_1 q_2 q_3 \ldots \) such that \( q_0 \in Q_0 \) and \( q_i \in \delta(q_i, \sigma_i) \) holds for all \( i \geq 0 \). Let \( \inf(\text{Run}(w)) \) denote the set of states occurring infinitely many times in the run \( \text{Run}(w) \). An infinite string \( w \) is accepted by the NBA \( G \) if it can visit a state in \( F \) infinitely often; that is, \( w \) is an accepted string of \( G \) if there exists a run \( \text{Run}(w) \) over \( w \) such that \( \inf(\text{Run}(w)) \cap F \neq \emptyset \). Therefore, the \( \omega \)-language accepted by the NBA \( G \), denoted as \( \mathcal{L}(G) \), is defined as the collection of all the \( \omega \)-strings that can be accepted by \( G \).

\[
\mathcal{L}(G) = \{ w \in \Sigma^\omega | \exists \text{ an accepting run } \text{Run}(w) \text{ over } w \text{ in } G \}. \tag{2.15}
\]

A Büchi automaton \( G \) is said to be deterministic (DBA) if: (i) \( |Q_0| \leq 1 \) and (ii) \( |\delta(q, \sigma)| \leq 1 \) for all \( q \in Q \) and \( \sigma \in \Sigma \). Different from NFA and DFAs, who are equally expressive in terms of regular languages, the class of \( \omega \)-strings accepted by DBAs is a strict subset of those who can be accepted by NBAs [7]. An \( \omega \)-language \( \mathcal{L} \) over \( \Sigma \) is said to be \( \omega \)-regular if there exists an NBA \( G \) such that \( \mathcal{L}(G) = \mathcal{L} \). For a DBA \( G \) without terminal states, we abuse the notation slightly and write \( L(G) \) to be the regular languages generated by viewing \( G \) as a DFA (without terminal states) whose marked states are set to be \( Q_m = F \). (2.15) then implies that

\[
\mathcal{L}(G) = \{ w \in \lim(L(G)) | \exists \text{ infinitely many } n \in \mathbb{N} \text{ such that } \delta(q_0, w^n) \in Q_m \} = \lim(L_m(G)). \tag{2.16}
\]

Note that (2.16) does not necessarily hold if \( G \) is a general NBA.
2.3 The $L^*$ Learning Algorithm

2.3.1 Details of the $L^*$ Learning Algorithm

In this dissertation, the formal synthesis and/or verification problems of ACPSs are addressed based on the application of oracle-guided inductive synthesis approaches [76]. In particular, we are interested in the adoption of regular language inference (a.k.a., automata learning) techniques, which are referred to as a class of algorithmic procedures of learning the canonical DFA that recognizes an unknown regular language with the minimal information of whether any finite string belongs to the language. The automata learning algorithms are commonly divided into two categories: passive and active learning algorithms. Passive learning algorithms assume that sufficient negative and positive samples for the target are available [14, 39, 126]; however, it is unrealistic to provide all the execution string of target systems in practice. Active learning algorithms construct models incrementally by actively querying the target system rather than relying passively on given strings [2, 168].

Among the active learning algorithms, the $L^*$ learning algorithm (abbreviated as the $L^*$ algorithm hereafter) is one of the most popular approaches. The $L^*$ algorithm, which was first developed by Angluin [2] and was improved later by Rivest and Shapire [146], learns an unknown regular language $U$ over a known event set $\Sigma$ and produces a minimal and canonical DFA $M$ such that $L_m(M) = U$. The $L^*$ algorithm assumes a Minimally Adequate Teacher, henceforth referred to as the Teacher, that answers two types of queries from the Learner: the first type is the membership query, in which the Teacher determines whether or not a string $s \in \Sigma^*$ belongs to $U$; the second type is the equivalence query, in which a conjecture DFA $M$ is presented to the Teacher and the Teacher answers whether $L_m(M) = U$ or not. If $L_m(M) \neq U$ the oracle returns a counterexample string $c \in L_m(M) \Delta U$.

At any given time, the $L^*$ algorithm has, in order to construct the conjectured DFA

---

$^1$By “minimal” we mean that the obtained DFA contains the least number of states.
$M$, information about a finite collection of strings over $\Sigma$, classified either as members or non-members of $U$. Such information is incrementally organized into an observation table, which is a 3-tuple $(S, E, T)$ that consists of: a prefix-closed set $S$ of strings, a suffix-closed set $E$ of strings (both of which are over $\Sigma$) and a membership function $T : (S \cup S \Sigma)E \rightarrow \{0, 1\}$. For $s \in S \cup S \Sigma$ and $e \in E$, $T(se)$ maps the string $se$ onto 1 if $se \in U$, otherwise it returns 0. The observation table can be viewed as a 2-dimensional array whose rows are labeled by strings $s \in S \cup S \Sigma$ and whose columns are labeled by symbols $e \in E$, with the entries in the labeled rows and columns equal to $T(se)$. For a string $s \in S \cup S \Sigma$, the row function $\text{row}(s)$ denotes the function $f$ from $E$ to $\{0, 1\}$ defined by $f(e) = T(se)$ [2]. The following two properties are essential in the $L^*$ algorithm.

**Definition 2.11** (Closed and Consistent Observation Tables). [2] An observation table is said to be

- **Closed** if $(\forall s \in S)(\forall \sigma \in \Sigma) [\exists s' \in S : \text{row}(s\sigma) = \text{row}(s')]$,
- **Consistent** if $(\forall s_1, s_2 \in S : \text{row}(s_1) = \text{row}(s_2))(\forall \sigma \in \Sigma)[\text{row}(s_1 \sigma) = \text{row}(s_2 \sigma)]$.

The working procedure of the $L^*$ algorithm is summarized as Algorithm 1. We define the $i$-th observation table constructed by the $L^*$ algorithm as $T_i$. For some $i \in \mathbb{N}$, if the observation table $T_i$ is not closed, then $s\sigma$ is added to $S$ and $T$ is updated to make it closed where $s \in S$ and $\sigma \in \Sigma$ are the elements for which no $s' \in S$ exists such that $\text{row}(s\sigma) = \text{row}(s')$. In the case that $T_i$ is not consistent, then there exist $s_1, s_2 \in S$, $\sigma \in \Sigma$ and $e \in E$ such that $\text{row}(s_1) = \text{row}(s_2)$ but $T(s_1\sigma e) \neq T(s_2\sigma e)$; to make it consistent, the $L^*$ algorithm adds $\sigma e$ to $E$ to refine $T$. After refining an observation table $T_i$ to be both closed and consistent, the $L^*$ algorithm constructs a conjecture DFA $M(S, E, T) := (Q, \Sigma, \delta, q_0, Q_m)$ as follows:

$$Q = \{\text{row}(s) : s \in S\}, \quad \delta(\text{row}(s), \sigma) = \text{row}(s\sigma)$$

$$q_0 = \text{row}(\epsilon), \quad Q_m = \{\text{row}(s) : (s \in S) \land (T(s) = 1)\}.$$ (2.17)
The Teacher performs the equivalence queries on the conjecture $M$ to justify whether or not $L_m(M) = U$. If $L_m(M) = U$, the Teacher terminates the $L^*$ algorithm and returns $M$; otherwise, it generates a counterexample $c \in L_m(M) \Delta U$. The $L^*$ algorithm updates $S$ to be $S \cup \{c\}$ and iterates the entire procedure to produce a new observation table.

Algorithm 1: The $L^*$ Algorithm [2]

**Input:** The event alphabet $\Sigma$

**Output:** The language $U$ and a minimal DFA $M$ such that $L_m(M) = U$

1: $S \leftarrow \{\epsilon\}$, $E \leftarrow \{\epsilon\}$, $i \leftarrow 0$

2: Construct the observation table $T_0(S, E, T)$

3: repeat

4: while $T_i(S, E, T)$ is not closed or consistent do

5: if $T_i$ is not closed then

6: find $s \in S$, $\sigma \in \Sigma_i$ such that $\forall s' \in S : row(s') \neq row(s\sigma)$

7: $S \leftarrow S \cup \{s\sigma\}$

8: extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries

9: end if

10: if $T_i$ is not consistent then

11: find $s_1, s_2 \in S$, $\sigma \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $T(s_1\sigma e) \neq T(s_2\sigma e)$

12: $E \leftarrow E \cup \{\sigma e\}$

13: extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries

14: end if

15: end while

16: $M_i \leftarrow M(T_i)$

17: if the Teacher presents a counterexample $c \in \Sigma^*$ then

18: $S \leftarrow S \cup \bar{c}$

19: $i \leftarrow i + 1$

20: extend $T_i$ to $(S \cup S\Sigma_i)E$ using membership queries

21: end if

22: until The Teacher generates no more counterexamples against $M_j$

23: $M \leftarrow M_i$ return $M$

In the following-up work by Rivest and Schapire [146], the $L^*$ algorithm only requires the observation table to be closed and analyzes the counterexample $c$ by finding the longest suffix $c_s$ of $c$ that witnesses the difference between $L_m(M)$ and $U$. Adding $c_s$ to $E$ reflects
the difference in next conjecture by splitting states in $M$. Once $c_s$ is added to $E$, the $L^*$ algorithm also iterates the entire process to update $M$ with respect to $c_s$.

2.3.2 Characteristics of the $L^*$ Algorithm

For each closed and consistent observation table $(S, E, T)$, the following theorem guarantees that the candidate DFA $M(S, E, T)$ is the “smallest” in terms of the number of states.

**Theorem 2.2.** [2] If $(S, E, T)$ is a closed and consistent observation table, then the DFA $M(S, E, T)$ constructed from $(S, E, T)$ is consistent with the finite function $T^2$; moreover, any other DFA consistent with $T$ but inequivalent to $M(S, E, T)$ must have more states.

Theorem 2.2 indicates that for each closed and consistent observation table, the $L^*$ algorithm constructs the uniquely smallest DFA $M$ that is consistent with $T$, which makes it particularly attractive for our framework introduced in next chapters. Angluin [2] proved that the sequence of DFAs constructed by the $L^*$ algorithm was strictly increasing in the number of states.

**Theorem 2.3.** [2] If $(S, E, T)$ is closed and consistent, and $M(S, E, T)$ is a DFA constructed from $(S, E, T)$. Let $(S', E', T')$ be an updated closed and consistent observation table if the counterexample $c$ is returned by the Teacher. If $M(S, E, T)$ has $N$ states, then the DFA $M(S', E', T')$ constructed from $(S', E', T')$ has at least $(N + 1)$ states.

Note that $M(S', E', T')$ has more states than $M(S, E, T)$ does not necessarily imply that $L_m(M(S, E, T)) \subseteq L_m(M(S', E', T'))$ in general cases. Finally, Angluin [2] also showed that the $L^*$ algorithm terminates with the correct answer by executing finitely many iterations.

---

2 A DFA $M$ is said to be consistent with the function $T$ if for every $s \in S \cup S \Sigma$ and $e \in E$, $\delta(q_0, se) \in Q_m$ if and only if $T(se) = 1$. 


Theorem 2.4. [2] Given any minimally adequate Teacher presenting an unknown regular language \( U \subseteq \Sigma^* \), the \( L^* \) algorithm eventually terminates and outputs an DFA isomorphic to the minimal one accepting \( U \). Moreover, if \( N \) is the number of states of the minimal DFA accepting \( U \) and \( M \) is an upper-bound on the length of any counterexample provided by the Teacher, then the total running time of the \( L^* \) algorithm is bounded by a polynomial of \( M \) and \( N \).

Remark 2.3. In fact, it can be proved in [2] that at most \((K + 1)(N + M(N − 1))N\) membership queries and at most \((N − 1)\) conjecture tests are sufficient for the \( L^* \) algorithm to construct the correct DFA, where \( K = |\Sigma| \). In the follow-up work [146], the bound of membership queries is improved to be \( O(KN^2 + N \log M) \).

2.3.3 An Example of the \( L^* \) Algorithm

This applies the \( L^* \) algorithm for learning an unknown regular language.

Example 2.1. Suppose that the unknown language \( U \) is the set of strings over the alphabet \( \Sigma = \{a, b\} \) with an event number of \( a \)'s and an event number of \( b \)'s. We illustrate how Algorithm 1 constructs a minimal DFA \( M \) such that \( \mathcal{L}(M) = U \). Initially, the \( L^* \) algorithm sets \( S = E = \{\epsilon\} \) (Line 1), and asks membership status of the strings \( \epsilon \), \( a \) and \( b \). This leads to the initial observation table \( T_1 \); however, \( T_1 \) is consistent but not closed since \( \text{row}(a) \) is distinct from \( \text{row}(\epsilon) \). The \( L^* \) algorithm then adds \( a \) to \( S \) and updates \( T_1 \) to \( T'_1 \) by querying the strings \( aa \) and \( ab \) (Lines 4-7).

![Figure 2.1. Observation tables and conjectured DFA (Part 1 of 3).](image-url)
The observation table $T_1'$ is closed and consistent and a conjectured DFA $M_1$ is constructed as depicted in Fig. 2.1 (Line 15). $M_1$ is not a correct DFA for $U$, since the Teacher can provide a counterexample (Line 17) $bb \in U - L_m(M_1)$ in this case.

To process the counterexample $bb$, the $L^*$ algorithm adds $\bar{bb} = \{\epsilon, b, bb\}$ to $S$ for the construction of a new observation table $T_2$, which is built up by querying the strings $ba$, $bba$ and $bbb$. $T_2$ is a closed observation table but it is not consistent, since $row(aa) \neq row(ba)$; to make $T_2$ consistent, the $L^*$ algorithm follows Lines 9-12 of Algorithm 1 by adding $a$ to $E$ and querying the strings $aaa$, $aba$, $bba$ and $bbba$ to construct the observation table $T_2'$, which is both closed and consistent. The DFA constructed based on $T_2'$, $M_2$, is depicted in Fig. 2.2.

Note that $M_2$ is also not correct for $U$, therefore the Teacher returns false and presents a counterexample $abb \in L_m(M_2) - U$. $L^*$ adds $\bar{abb} = \{\epsilon, a, ab, abb\}$ to $S$ for the construction of a new observation table $T_3$ by querying abba, abaa, abbaa, abbb and abbaa, which is
This table turns out to be closed but not consistent, since \( \text{row}(bb) \neq \text{row}(abb) \). To make \( T_3 \) consistent, the \( L^* \) algorithm adds \( b \) to \( E \) and updates the table to be \( T'_3 \), which is both closed and consistent. A DFA \( M_3 \) is then constructed based on \( T'_3 \) and is presented as a conjecture to the Teacher; this time it is a correct DFA that accepts \( U \) and the Teacher returns true.

2.4 Essentials of Compositional Verification Methods

2.4.1 Assume-guarantee Reasoning in Compositional Verification

In this section, we briefly review the compositional verification procedures presented in [36]. An assume-guarantee formula in compositional verification is a triple \( \langle A \rangle M \langle P \rangle \),
where \( M \) is the system model, \( P \) is the property to be verified and \( A \) is an assumption about \( M \)’s environment, each of which can be represented by an associated DFA. The formula holds if whenever \( M \) is part of a system satisfying \( A \), the system must also guarantee the property \( P \), i.e., \( \forall E, E \parallel M \models A \) implies that \( E \parallel M \models P \) [132]. Recall Definition 2.5, it has been verified in [36] that a system \( M \) violates a property \( P \) if and only if the error state \( q_e \) is reachable in \( M \parallel \tilde{P} \), where \( \tilde{P} \) is the completion of \( P \) (cf. Definition 2.3). Therefore, we can have the following theorem that presents the necessary and sufficient condition for an assume-guarantee formula to be satisfied.

**Theorem 2.5.** [132] When \( \Sigma_P \subseteq \Sigma_A \cup \Sigma_M \), the assume-guarantee formula \( \langle A \rangle M \langle P \rangle \) holds if and only if \( q_e \) is unreachable in \( A \parallel M \parallel \tilde{P} \), where \( \tilde{P} \) is the completion of \( P \) and \( q_e \) is the error state stated in Definition 2.3.

Under the assumption that \( \Sigma_P \subseteq \Sigma_A \cup \Sigma_M \) is satisfied, Theorem 2.5 is in fact equivalent to \( M \parallel A \models P \).

A series of symmetric and asymmetric assume-guarantee proof rules are incorporated in compositional verification. The simplest proof rule is the following asymmetric rule (ASYM) [132] for determining whether or not a property \( P \) can be satisfied by a system \( M \) that consists of two components, namely \( M_1 \) and \( M_2 \),

\[
\begin{align*}
1 & \quad \langle A \rangle M_1 \langle P \rangle \\
2 & \quad \langle \text{true} \rangle M_2 \langle A \rangle \\
& \quad \langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle
\end{align*}
\]

where \( A \) denotes an assumption about the environment in which \( M_1 \) is placed. The soundness and completeness of the ASYM rule is proved in the literature [132]. Note that the rule is not symmetric in its use of the two components, and does not support circularity; thus we call ASYM to be an asymmetric and non-circular assume-guarantee proof rule.

The *weakest assumption* is a key ingredient of compositional verification, which is defined formally as follows.
Definition 2.12 (Weakest Assumption). [132] Let $M_1$ and $M_2$ be two system components defined over $\Sigma_1$ and $\Sigma_2$, respectively, and $P$ be a property defined over $\Sigma_P$. Let $\Sigma_A := (\Sigma_1 \cup \Sigma_P) \cap \Sigma_2$ be an interface alphabet, the weakest assumption for $M_1$ is a DFA $A_w$ defined over $\Sigma_A$ such that for any component $M_2$, $\langle \text{true} \rangle M_1 \parallel P_A(M_2) \langle P \rangle$ if and only if $\langle \text{true} \rangle M_2 \langle A_w \rangle$, where $P_A$ denotes the natural projection from $\Sigma_2^*$ to $\Sigma_A^*$.

In general, the number of states of the assumption DFA $A_w$ is less than the number of states of the system component $M_2$; therefore, deployment of an assume-guarantee reasoning paradigm can efficiently justify $M_1 \parallel M_2 \models P$ since computation of the parallel composition $M_1 \parallel M_2$ is avoided. The following lemma can be immediately derived from Definition 2.12

Lemma 2.1. [132] Given the weakest assumption $A_w$ as above, $\langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle$ if and only if $\langle \text{true} \rangle M_2 \langle A_w \rangle$.

2.4.2 Assume-guarantee Reasoning for Multiple Subsystems

In practice, a distributed ACPS typically comprises many individual subsystems that work synchronously and jointly. To apply directly the ASYM proof rule for assume-guarantee reasoning for a large-scale system that consists of multiple subsystems, the subsystems have to be partitioned into two groups, namely $M_1$ and $M_2$, to fit the ASYM proof rule; however, it has been shown in [37] that a good subsystem partition is very important to the compositional verification with the ASYM proof rule, and if the partition is improper, traditional monolithic verification procedure may even outperform the assume-guarantee reasoning approach, which requires new proof rules to deal with the general multi-agent case.

In this subsection, we consider the assume-guarantee proof rules for the system that is composed of arbitrarily finite number of subsystems. For such a purpose, we review a variety of assume-guarantee proof rules that are sound, whereas completeness of these
rules is required to guarantee termination. We first investigate the following proof rule from [132], which is similar to the ASYM proof rule but involves some form of circular reasoning, and is named the CIRC-N proof rule for its asymmetric nature.

1 \langle A_1 \rangle M_1 \langle P \rangle \\
2 \langle A_2 \rangle M_1 \langle A_1 \rangle \\
\ldots \ldots \\
n \langle A_n \rangle M_n \langle A_{n-1} \rangle \\
n + 1 \langle \text{true} \rangle M_1 \langle A_n \rangle \\
\frac{(\text{true})(M_1 || M_2 || \cdots || M_n) \langle P \rangle}{\langle \text{true} \rangle(M_1 || M_2 || \cdots || M_n) \langle P \rangle}

Soundness and completeness properties of this proof rule follow from [68]. Although sound and complete, the CIRC-N proof rule is not always satisfactory in practice since it is not symmetric in the use of the component modules of the subsystems. In contrast, we prefer to use the following circular and symmetric proof rule, namely SYM-N, for the assume-guarantee reasoning in compositional verification. Note that the number of premises to be proved in the SYM-N proof rule and the number of assumptions required increase linearly with the number of the agents that are involved in the system.

1 \langle A_1 \rangle M_1 \langle P \rangle \\
2 \langle A_2 \rangle M_2 \langle P \rangle \\
\ldots \ldots \\
n \langle A_n \rangle M_n \langle P \rangle \\
n + 1 \frac{L_m(coA_1||coA_2||\cdots||coA_n) \subseteq P}{(\text{true})(M_1 || M_2 || \cdots || M_n) \langle P \rangle}

To pursue a successful application of the SYM-N proof rule, We require that \( \Sigma(P) \subseteq \Sigma = \bigcup_{i \in I} \Sigma_i \), and that for \( i \in I \), \( \Sigma(A_i) \subseteq (\bigcap_{i \in I} \Sigma_i) \cup \Sigma(P) \) [132]. Informally, each \( A_i \) is a postulated environment assumption for the component \( M_i \) to achieve property \( P \). Recall that \( coA_i \) is the complement of \( A_i \).
The soundness and completeness of the SYM-N proof rule are summarized in the following theorem.

**Theorem 2.6.** [132] The SYM-N proof rule is sound and complete.

For each $i \in I := \{1, 2, \ldots, n\}$, let $\Sigma_{-i} = \bigcup_{j \in I \setminus \{i\}} \Sigma_j$ denote the set of events that belong to all the subsystems except $G_i$. By setting $\Sigma_{A_i} = \Sigma_{-i}$, the weakest assumption $A_i$ with respect to $M_i$ and $P$ in the SYM-N proof rule can be computed accordingly.

2.5 Conclusion

This chapter introduces the basic concepts and notions in automata theory, formal languages, the $L^*$ learning algorithm and compositional verification techniques. These definitions and theorems provide the tools of solving the various formal synthesis problems for large-scale CPSs with potential model and/or environment uncertainties in the following chapters.
CHAPTER 3

LEARNING-BASED SUPERVISOR SYNTHESIS OF DISCRETE EVENT SYSTEMS FOR FORMAL SPECIFICATIONS

3.1 Introduction

When the logical behaviors and decision-making processes in the cyber part of an ACP-S can be modeled as a finite automaton (or a Büchi automaton for non-terminating behaviors), the computation of corresponding high-level control policies can be carried out by means of supervisory control techniques. In this chapter, we propose a formal approach for computing a permissive supervisor (in the sense of language inclusion) for an uncontrolled DES in order to fulfill various types of formal specifications. Our study starts with the supervisory control of DESs under complete observation, in which we leverage the idea of the formal inductive synthesis methods [76] and develop a counterexample-guided supervisor synthesis framework based on the modification of the $L^*$ learning algorithm [2]. Due to the polynomial complexity of the $L^*$ algorithm (ref. Theorem 2.4), the proposed algorithm turns to serve as a computationally efficient tool to synthesize correct-by-construction supervisors that steers the uncontrolled DES to satisfy a regular language specification. More specifically, the proposed learning-based algorithm correctly converges to the supremal controllable sublanguage of the given prefix-closed specification language within a finite number of iterations under complete observation. Furthermore, compared to conventional supervisor synthesis algorithms (see, e.g., [26, 91, 142, 143]), which require precise DFA models of both the uncontrolled DES and the formal specification, one advantage of the learning-based synthesis algorithm developed in this chapter is that the algorithm can
also be applied for DESs whose complete model knowledge may not be known *a priori*. The supervisor synthesis algorithm is further extended to partially-observed DESs. In this case, the learning-based synthesis algorithm can compute a satisfactory supervisor that result in more permissive closed-loop behaviors than the supremal controllable and normal sublanguage of the given specification in the literature (see, e.g., [26, 91]).

In addition to specifications that are represented in regular languages, we are also interested in the supervisory control problems in which the control specification is given in the form of an \( \omega \)-regular language. Such consideration is motivated from the real application in which regular languages are lack of expressiveness when the control specifications are given by not only safety properties, but also other properties such as liveness\(^1\) properties. In this chapter, we consider a special class of the supervisor synthesis problems for non-terminating specifications in which the uncontrolled system is modeled as a deterministic Büchi automaton. By incorporating the \( L^* \) algorithm with language properties such as controllability, observability and deadlock-freedom, we manage to derive a series of learning-based supervisor synthesis algorithms to satisfy non-terminating specifications under both complete and partial observation of the plant. We expect the proposed algorithms to provide more insights on controlling a DES plant for other types of non-terminating behaviors, such as temporal logic formulas.

The rest of this chapter is organized as follows. We introduce the Ramadge-Wonham framework of supervisory control under complete observation and present a learning-based supervisor synthesis algorithm for conventional DES in Section 3.2 with formal proofs of their correctness and finite convergence. The learning-based approaches are then extended in Section 3.3 to explore synthesis methods of computing a permissive supervisor for a partially-observed plant. In Section 3.4, we further extend the learning-based synthesis algorithms so that they can compute a deterministic supervisor that steers an uncontrolled deterministic Büchi automaton to satisfy a class of \( \omega \)-regular language specifications. We

\(^1\) A property \( P \) over \( \Sigma \) is a liveness property if \( P \subseteq \Sigma^\omega \) and \( \overline{P} = \Sigma^* \).
conclude this chapter in Section 3.5.

3.2 Learning-based Supervisor Synthesis of Discrete Event Systems

3.2.1 The Ramadge-Wonham Supervisory Control Theory

Initiated by Ramadge and Wonham [142, 143], the supervisory control theory provides a formal framework for properly controlling behaviors of DESs that are modeled as finite automata.

In the Ramadge-Wonham framework, the control is exercised by a feedback controller, called a supervisor, whose action is to enable and/or disable events so that the controlled system generates some pre-specified specification languages.

According to Theorem 2.1, we assume without loss of generality that the uncontrolled DES plant under consideration is modeled by a trim DFA $G = (Q, \Sigma, \delta, q_0, Q_m)$, whose ingredients are defined according to Definition 2.1. We wish to adjoin a supervisor, denoted as $S$, to interact with $G$ in a “string feedback” manner, as depicted in Fig. 3.1.

Figure 3.1. The feedback loop of supervisory control with complete observation

For the purpose of control, the event set $\Sigma$ is partitioned into two disjoint subsets $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where $\Sigma_c$ is the set of controllable events, whose occurrence can be disabled by the supervisor $S$; whereas $\Sigma_{uc}$ is the set of uncontrollable events, whose occurrence cannot be disabled by the supervisor $S$. Note that it is reasonable to set a certain event to be
uncontrollable in practical engineering systems; for instance, an uncontrollable event may model the reception of a sensor reading in a control loop, whose occurrence cannot be prohibited due to hardware or actuation limitations.

We define $\Gamma = \{ \gamma \in 2^\Sigma | \Sigma_{uc} \subseteq \gamma \}$ as the set of control patterns [26], i.e., uncontrollable events are always enabled in any control patterns. Formally, a supervisor $S$ is a mapping from the generated language of $G$ to the set of control patterns, $S : L(G) \rightarrow \Gamma$, and $S(s)$ stands for all the events that are enabled by the supervisor $S$ after the plant $G$ generates the string $s$. The closed-loop system of $G$ controlled by $S$, denoted as $S/G$, is another DES whose generated and marked languages are simply subsets of $L(G)$ and $L_m(G)$ containing the strings that remain feasible in the presence of $S$, respectively. This is formalized as the following definition.

**Definition 3.1.** The language $L(S/G)$ generated by $S/G$ is defined recursively as follows:

1. $\epsilon \in L(S/G)$;
2. $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)[s \in L(S/G) \land s\sigma \in L(G) \land \sigma \in S(s) \Leftrightarrow s\sigma \in L(S/G)]$.

The language marked by $S/G$ is defined as $L_m(S/G) = L(S/G) \cap L_m(G)$.

The aforementioned supervisor $S$ in Fig. 3.1 can equivalently be realized as a DFA that operates in parallel with $G$. With slightly abusing the notations, we assume that the supervisor is realized as a DFA $S = (Z, \Sigma, \xi, z_0, Z_m)$ over $\Sigma$. By introducing the DFA realization of $S$, the closed-loop behaviors $L(S/G)$ and $L_m(S/G)$ can be re-written as $L(S\|G)$ and $L_m(S\|G)$, respectively. According to Definition 2.2, $L(S\|G) = L(S) \cap L(G)$ and $L_m(S\|G) = L_m(S) \cap L_m(G)$, thus the parallel composition based supervisor restricts the behaviors of the plant as well.

In order for the control achieved by a DFA-based supervisor to be equivalent to the one achieved by a mapping-based supervisor as defined in Definition 3.1, the following two conditions must hold.

(i) An uncontrollable event should never be disabled;
(ii) $L_m(S\|G) = L(S\|G) \cap L_m(G)$.

A DFA-based supervisor $S$ meets the first requirement if it is $\Sigma_{uc}$-enabling, which is formally defined as follows.

**Definition 3.2 ($\Sigma_{uc}$-enabling Supervisors).** [91] The supervisor $S$ is said to be $\Sigma_{uc}$-enabling if

\[
(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma_{uc})[s \in L(S\|G) \land s\sigma \in L(G) \iff s\sigma \in L(S\|G)].
\]  

(3.1)

In other words, if the DFA realization $S$ is $\Sigma_{uc}$-enabling, then it holds that for each $s \in L(S)$, $\text{Act}_S(\xi(s)) = S(s)$, where the $S$ on the right side of the equation stands for the supervisor mapping.

Moreover, since $L(S\|G) \cap L_m(G) = [L(G) \cap L(S)] \cap L_m(G) = L(S) \cap L_m(G)$, a DFA-based supervisor $S$ meets the second requirement if it is non-marking, as explained in the following definition.

**Definition 3.3 (Non-marking Supervisors).** [91] The supervisor $S$ is said to be non-marking if

$L_m(G) \cap L(S) = L_m(S\|G)$.

Note that a sufficient condition for $S$ to be non-marking is that $L(S) = L_m(S)$.

We introduce the following properties of a regular language that help characterize the class of solvable supervisory control problems.

**Definition 3.4 (Closed and Controllable Languages).** A regular language $L \subseteq \Sigma^*$ is said to be

- $L_m(G)$-closed (a.k.a., relative-closed) with respect to $G$ [91] if $\overline{L} \cap L_m(G) = L$;
- Controllable with respect to $G$ and $\Sigma_{uc}$ [142] if $\overline{L\Sigma_{uc}} \cap L(G) \subseteq \overline{T}$;

It is shown in [142, 143] that given any plant $G$ modeled as a DFA with complete observation, there exists a $\Sigma_{uc}$-enabling, non-marking DFA-based supervisor $S$ that can restrict $L(S\|G)$ to any non-empty and prefix-closed language $L \subseteq L(G)$ if and only if $L$ is
controllable with respect to $G$ and $\Sigma_{uc}$. Furthermore, a non-blocking supervisor $S$ exists such that the marked language $L_m(S \parallel G) = L$ for any non-empty language $L \subseteq L_m(G)$ if and only if $L$ is controllable and $L_m(G)$-closed with respect to $G$. If the given specification language $L$ is not controllable, we define the class of controllable sublanguage of $L$ and the class of prefix-closed and controllable superlanguage of $L$ as follows, respectively:

\[
C_{in}(L) := \{ K \subseteq L | K \Sigma_{uc} \cap L(G) \subseteq K \};
\]
\[
C_{out}(L) := \{ K \subseteq \Sigma^* | (L \subseteq K \subseteq L(G)) \wedge (K = K) \wedge (K \Sigma_{uc} \cap L(G) \subseteq K) \}.
\]

These two classes are certainly not empty since $\emptyset \in C_{in}(L)$ and $L(G) \in C_{out}(L)$. On the one hand, since controllability of languages are closed under arbitrary unions, there exists a unique supremal element $\sup C(L) = \bigcup_{K \in C_{in}(L)} K$ in $C_{in}(L)$ and is denoted as the \textit{supremal controllable sublanguage} of $L$. Furthermore, if $L$ is prefix-closed, then so is $\sup C(L)$. On the other hand, the set $C_{out}(L)$ is also closed under arbitrary intersections and therefore, the \textit{infimal prefix-closed and controllable superlanguage} of $L$, written as $\inf C(L)$, also uniquely exists in $C_{out}(L)$. More specifically, given $G$, $\Sigma_{uc}$ and $L$, $\inf C(L)$ can be computed as [26]

\[
\inf C(L) = \overline{\Sigma_{uc}^*} \cap L(G).
\]

3.2.2 Problem Formulation

The uncontrolled DES (a.k.a., a plant) under consideration in this chapter is modeled as the following DFA

\[
G = (Q, \Sigma, \delta, q_0),
\]

where $Q$, $\Sigma$ and $\delta$ are defined as Definition 2.1, and $q_0 \in Q_0$ is the unique initial state of $G$; note that the set $Q_m$ of marked states are not involved in this chapter since we are more interested in the generated behaviors of the plant under (possible) control. To apply

\footnote{A DFA-based supervisor $S$ is said to be non-blocking if $L_m(S \parallel G) = L(S \parallel G)$}
the supervisory control framework shown in Fig. 3.1 for $G$, the event set $\Sigma$ is partitioned into the set $\Sigma_c$ of controllable events and the set $\Sigma_{uc}$ of uncontrollable events. Based on these notations, we can then formulate the basic supervisory control problem (BSCP) of the plant $G$ under complete observation as follows.

**Problem 3.1 (BSCP).** Consider a DES plant $G$ (3.4) over $\Sigma$ and a prefix-closed specification language $L \subseteq \Sigma^*$. Let $\Sigma_{uc} \subseteq \Sigma$ be the set of uncontrollable events, find a $\Sigma_{uc}$-enabling and non-marking supervisor $S$ for $G$ that satisfies:

(i) **Correctness:** $L(S \parallel G) \subseteq L$;

(ii) **Maximal permissiveness:** for any other supervisor $S'$ satisfying (i), $L(S \parallel G) \not\subseteq L(S' \parallel G)$.

From Definition 3.1, the closed-loop behaviors of the plant $G$ under the control of the supervisor $S$ should satisfy $L(S/G) = L(S \parallel G) \subseteq L(G)$. Therefore, it is reasonable to assume that the specification language $L = \mathcal{T} \subseteq L(G)$ in Problem 3.1.

### 3.2.3 The $L^*_C$ Algorithm: Supervisor Synthesis via Automata Learning

Whenever $L$ is controllable with respect to $G$ and $\Sigma_{uc}$, the Ramadge-Wonham supervisory control theory suggests that the solution of Problem 3.1 is a supervisor $S$ whose DFA realization satisfies $L(S) = L_m(S) = L$ [26, 91]. Otherwise, the maximally permissive solution of Problem 3.1 is the supervisor $S$ that restricts the plant’s behavior to the supremal controllable sublanguage $\text{sup } C(L)$ of $L$. To compute the maximally permissive supervisor whenever $L$ fails to be controllable, we modify the $L^*$ algorithm and propose a learning-based algorithm, namely the $L^*_C$ algorithm, to compute a maximally permissive supervisor $S$ via automata learning techniques. The algorithm is illustrated in Algorithm 2 and the ingredients of the $L^*_C$ algorithm are explained as follows in details.

---

3The subscript “C” stands for “controllability”.
**Membership Queries** Similar to the conventional $L^*$ algorithm, the $L^*_C$ algorithm learns the desired supervisor by building up a family of observation tables. The construction of the observation table starts with the *membership queries* employed by the $L^*_C$ algorithm. Different from the $L^*$ algorithm, the membership queries of the $L^*_C$ algorithm are formed on the basis of *illegal* strings. Under the assumption that $L = \overline{L} \subseteq L(G)$, a string $s \in L(G)$ is said to be *illegal* (with respect to $L$) if $s \notin L$. Furthermore, a string $st \in L(G)$ is said to be *uncontrollably illegal* if $st$ becomes illegal due to a suffix that is composed of only uncontrollable event; in other words, $st$ is uncontrollably illegal with respect to the prefix-closed specification $L$ if and only if $s \in L$, $t \in \Sigma_{uc}^*-\{\epsilon\}$ but $st \notin L$.

We denote by $C$ the collection of uncontrollably illegal behaviors of $G$. Thus $C$ can be viewed as as a subset of illegal behaviors that may emerge in $G$, i.e., $C \subseteq L(G) - L$. In case that the plant model $G$ is explicitly given before, $C$ can be obtained precisely by means of complete exploration approaches such as breadth-first and/or depth-first search techniques [2, 204]. The $L^*_C$ algorithm initializes $C$ by setting $C^0 = \emptyset$ and for each $j \in \mathbb{N}$, we denote by $C^j$ the set of uncontrollably illegal strings after the $j$-th uncontrollably illegal string has been added to $C$. The Teacher hence updates $C$ by $C^{j+1} := C^j \cup \{s_{j+1}\}$ before constructing the $j$-th observation table.

Let $D_u(\cdot)$ be the operator that computes the “legal” prefixes of an uncontrollably illegal string by removing the uncontrollable suffix; that is, for a string $st \in C$ with $s \in L$ and $t \in \Sigma_{ic}^*$, $D_u(st) = s$. Formally, we define that

$$D_u(C) = \{s \in L(G) | (\exists t \in \Sigma_{uc}^*)[st \in C]\}. \quad (3.5)$$

The $L^*_C$ algorithm differs from that of the $L^*$ algorithm in the sense that the Teacher in the $L^*_C$ algorithm sequentially answers a family of dynamical and conditional membership queries during each iteration step rather than the static membership queries. Formally, let $T^j (j \in \mathbb{N})$ denote the membership function in the observation tables constructed by the
\(L^*_c\) algorithm in the \(j\)-th iteration. When \(j = 0\), the Teacher initializes the membership queries by following the conventional \(L^*\) algorithm, i.e., for \(t \in \Sigma^*\),

\[
T^0(t) = \begin{cases} 
0, & \text{if } t \notin L, \\
1, & \text{otherwise.} 
\end{cases} 
\]  
(3.6)

On the other hand, when \(j \geq 1\), the answers to the membership queries are determined based on the set \(C^j\):

\[
T^j(t) = \begin{cases} 
0, & \text{if } T^{j-1}(t) = 0 \text{ or } t \in D_u(C^j)\Sigma^*, \\
1, & \text{otherwise.} 
\end{cases} 
\]  
(3.7)

Note that a DFA \(M\) that is consistent with the membership function \(T^j\) for \(j \geq 1\) shall satisfy that \(L_m(M) = L - D - u(C^j)\Sigma^*\).

Thanks to the membership function \(T^j\), the \(L^*_c\) algorithm can then evaluate and build up a closed and consistent observation table in an exactly same way as the conventional \(L^*\) algorithm does.

**Equivalence Queries and Counterexamples** By correctly answering the membership queries via (3.6) and (3.7), the Teacher is able to build up a closed and consistent observation table \(S, E, T^j\) in the \(j\)-th iteration and a conjecture DFA \(M^j = M(S, E, T^j)\) \((j \in \mathbb{N})\) can hence be constructed with respect to \((S, E, T^j)\) by applying the procedure given in (2.17). Note that the \(L^*_c\) algorithm always deals with prefix-closed languages, and therefore all the states in \(M^j\) are marked by following (2.17). The \(L^*_c\) algorithm then takes \(S = M^j\) as the supervisor and justifies whether or not the correctness and maximal permissiveness of \(M^j\) can be satisfied. Since the target language to be learned by the \(L^*_c\) algorithm need not be \(L\), the equivalence queries of the traditional \(L^*\) algorithm cannot be utilized by the \(L^*_c\) algorithm. To implement the equivalence queries and to facilitate the generation of counterexamples, the \(L^*_c\) algorithm takes advantage of the following se-
quence \( \{K^j\} \) \((j \in \mathbb{N})\) of languages.

\[
K^0 := L, \quad K^{j+1} := K^j - [(L(G) - K^j)/\Sigma_{uc}]\Sigma^*, \quad j \geq 1.
\]

(3.8)

It can be readily verified from (3.8) that \( \{K^j\}_{j \in \mathbb{N}} \) is a monotone decreasing (in the sense of language inclusion) sequence of prefix-closed languages. The Teacher hence answers the equivalence queries with respect to the conjecture DFA \( M^j = M(S, E, T^j) \) \((j \in \mathbb{N})\) by evaluating the following function \( \xi^j : \Sigma^* \rightarrow \{0, 1\} \), where \( uc \) stands for "uncontrollably illegal strings".

\[
\xi^j(t) = \begin{cases} 
0, & \text{if } t \in L(M^j) \Delta K^j, \\
1, & \text{otherwise}.
\end{cases}
\]

(3.9)

A string \( t \in \Sigma^* \) that results in \( \xi^j(t) = 0 \) will be categorized as a counterexample, and the \( L^*_C \) algorithm processes \( t \) in exactly the same way as the \( L^*_C \) algorithm does by adding all the strings in \( \tilde{t} \) to \( S \) so that a new observation table can be constructed. Furthermore, if \( t \in L(M^j) - K^j \) and \( t \) is an uncontrollably illegal string (with respect to \( L \)), then the \( L^*_C \) algorithm adds \( t \) to \( C^j \) to update \( C^{j+1} \) and the membership function (3.7).

The \( L^*_C \) Synthesis Algorithm The supervisor synthesis procedure of the \( L^*_C \) algorithm is presented in Algorithm 2, in which the notation \( L^*_C(G, \Sigma_{uc}, L) \) indicates that the algorithm computes the desired supervisor \( S \) based on the information of \( L(G) \), \( \Sigma_{uc} \) and \( L \).

Algorithm 2: The \( L^*_C(G, \Sigma_{uc}, L) \) learning algorithm

\textbf{Input}: The plant \( G \), the specification \( L \), the event set \( \Sigma \), and the set \( \Sigma_{uc} \) of uncontrollable events.

\textbf{Output}: A correct and maximally permissive supervisor \( S \).

1. \( S \leftarrow \{\epsilon\}, E \leftarrow \{\epsilon\}, j \leftarrow 0, C^0 \leftarrow \emptyset \)
2. Construct \( (S, E, T^j) \) with membership function \( T^j \) in (3.6) and (3.7)
3: repeat
4:   while \((S, E, T^j)\) is not closed or consistent do
5:     if \((S, E, T^j)\) is not closed then
6:       Find \(s \in S, \sigma \in \Sigma\) such that \(\forall s' \in S: \text{row}(s') \neq \text{row}(s\sigma)\)
7:       \(S \leftarrow S \cup \{s\sigma\}\)
8:     end if
9:     if \((S, E, T^j)\) is not consistent then
10:    Find \(s_1, s_2 \in S, \sigma \in \Sigma\) and \(e \in E\) such that \(\text{row}(s_1) = \text{row}(s_2)\) but \(T^j(s_1\sigma e) \neq T^j(s_2\sigma e)\)
11:   \(E \leftarrow E \cup \{\sigma e\}\)
12: end if
13: Extend \(T^j\) to \((S \cup \Sigma E)\) using membership queries (3.6) and (3.7)
14: end while
15: Construct \(M^j = M(S, E, T^j)\) based on the procedure in (2.17)
16: Perform the equivalence queries (3.9) among \(L, L(M^j)\) and \(K^j\)
17: if \(t \neq \epsilon\) and \(\xi^j(t) = 0\) then
18:   \(S \leftarrow S \cup t\)
19:   \(j \leftarrow j + 1\)
20: if \(t\) is uncontrollably illegal then
21:   \(C^{j+1} \leftarrow C^j \cup \{t\}\)
22: end if
23: Extend \(T^j\) to \((S \cup \Sigma E)\) using membership queries (3.6) and (3.7)
24: end if
25: until \(t = \epsilon\)
26: \(S \leftarrow M^j\) return \(S\)

The working procedure of the \(L_C^*\) algorithm is stated as follows. In the initialization part (lines 1-2), the \(L_C^*\) algorithm sets \(S = E = \{\epsilon\}\) and builds up an initial observation table by answering the membership function \(T^0(t)\) (3.6) and evaluating the entries in the observation table. In the \(j\)-th iteration step, from lines 4 to 14, the \(L_C^*\) algorithm makes the current observation table closed and consistent based on Definition 2.11. When the observation table is closed and consistent, a conjecture DFA \(M^j\) is constructed by following (2.17) at line 15. The equivalence query function \(\xi^j(t)\) is then performed over \(M^j\) to check whether or not \(M^j\) satisfies the correctness at line 16. Once a string \(t \in \Sigma^*\) is generated such that \(\xi^j(t) = 0\), then \(t\) is a counterexample \(t\) and is further analyzed in lines 17 to 22. Specifically, if \(t\) is a common counterexample, then all the prefixes of \(t\) are added.
to \( S \) to update the observation table; besides, \( t \) will also be added to \( C^j \) to update \( C^{j+1} \) if it is an uncontrollably illegal behavior. The aforementioned loop iterates until no more counterexamples are reported by the Teacher; and the algorithm terminates with the current conjecture \( M^j \).

3.2.4 Characteristics of the \( L_C^* \) Algorithm

In this subsection, we investigate the correctness, maximal permissiveness and finite convergence properties of the \( L_C^* \) algorithm developed in Algorithm 2.

**Correctness** Our study starts with the following lemma that provides an iterative procedure to compute the supremal controllable sublanguage of a given regular language.

**Lemma 3.1.** [91] The supremal controllable sublanguage \( \text{sup} \ C(L) \) of a non-empty language \( L \subseteq L(G) \) with respect to \( G \) and \( \Sigma_{uc} \) can be computed iteratively by following (3.8). If there exists \( N \in \mathbb{N} \) such that \( K^{N+1} = K^N \), then \( \text{sup} \ C(L) = K^N \). Furthermore, if \( L \) is prefix-closed, then \( \text{sup} \ C(L) \) is also prefix-closed and can be computed directly as

\[
\text{sup} \ C(L) = L - [(L(G) - L)/\Sigma_{uc}^*]\Sigma^*.
\]  

Since the supremal controllable sublanguage \( \text{sup} \ C(L) \) of the given specification \( L \subseteq L(G) \subseteq \Sigma^* \) uniquely exists, then a supervisor \( S \) that steers the plant \( G \) to achieve \( \text{sup} \ C(L) \) will be the unique maximally permissive solution of Problem 3.1. Therefore, the correctness and maximal permissiveness of the \( L_C^* \) algorithm can be attained if \( S = L_C^*(G, \Sigma_{uc}, L) \) is the supervisor that achieves \( \text{sup} \ C(L) \). Based on Lemma 3.1, we first establish the correctness and maximal permissiveness of the \( L_C^* \) algorithm presented in Algorithm 2 in the following theorem.

**Theorem 3.1.** Given the plant \( G \), the set \( \Sigma_{uc} \subseteq \Sigma \) of uncontrollable events, and a non-empty and prefix-closed specification \( L \subseteq L(G) \), the \( L_C^* \) algorithm computes a DFA \( S = L_C^*(G, \Sigma_{uc}, L) \) such that \( L(S) = L(S\|G) = \text{sup} \ C(L) \) if it terminates.
Proof. By comparing the iteration (3.8) to the dynamical membership queries presented in (3.6), it is clear that initially the DFA consistent with $T^0(t)$ is a DFA $M$ such that $L(M) = L_m(M) = L$, which coincides with $K^0$ in (3.8). When $j \geq 1$, the $L_C^*$ algorithm keeps performing equivalence queries and generating counterexamples to update not only the observation table but the set $C^j$ as well (whenever either $t \in L(G) - K^j$ or $t \in L(G) - L(M^j)$ is satisfied). On the one hand, a DFA $M$ that is consistent with the membership function (3.7) should satisfy that

$$L(M) = L_m(M) = L - D_u(C^j)\Sigma^*.$$  

(3.11)

On the other hand, under the assumption that the $L_C^*$ algorithm does terminate. Then there exists $N \in \mathbb{N}$ such that the set $C^N$ eventually collects all the uncontrollably illegal strings with respect to $L$ when no more counterexamples are returned by the Teacher. In this case, the set $C^N$ is given by $C^N = L - [D_u(L(G) - L)]\Sigma^*$. Thus, from (3.11), we can write that at the $N$-th iteration step, the DFA $M$ that is consistent with (3.7) satisfies that

$$L(M) = L_m(M) = L - D_u(C^N)\Sigma^* = L - [D_u(L(G) - L)]\Sigma^* = L - [(L(G) - L)/\Sigma_{uc}^*]\Sigma^*.$$  

(3.12)

In other words, $L(M)$ in fact coincides with the results obtained in (3.10), which implies that if the $L_C^*$ algorithm can terminate within a finite number of iterations, it should correctly returns a DFA that recognizes the language $\text{sup} \ C(L)$. Since the supremal controllable sublanguage of a given language uniquely exists, the correctness and maximal permissiveness of the $L_C^*$ algorithm can be proved.

Next, we show that the $L_C^*$ algorithm computes the satisfactory supervisor within a finite number of iterations, which is formally stated in the following theorem.

**Theorem 3.2.** Given the plant $G$, the set $\Sigma_{uc} \subseteq \Sigma$ of uncontrollable events, and a non-empty and prefix-closed specification $L \subseteq L(G)$, the $L_C^*$ algorithm correctly returns a...
correct supervisor $S = L^*_C(G, \Sigma_{uc}, L)$ within a finite number of iterations.

Proof. We assume that in the $j$-th iteration step, the DFA $M^j = M(S, E, T^j)$ constructed by the $L^*_C$ algorithm has $N_j$ states, where $(S, E, T^j)$ is the $j$-th closed and consistent observation table. Furthermore, we assume that the number of the states of the minimal DFA accepting $\sup C(L)$ is given by $N_{\sup}$. On the one hand, the $L^*_C$ algorithm can be viewed as the procedure of the $L^*$ algorithm with the membership functions $T^j(t)$ and equivalence queries $\xi^j(t)$ for $t \in \Sigma^*$. Therefore, according to Theorem 2.2, $n_j$ is the smallest number of states of all the DFAs that are isomorphic to $M^j$. Furthermore, it can be inferred from Theorem 2.3 that after the $(j + 1)$-th counterexample $t \in \Sigma^*$ is generated and is used to update the membership function $T^{j+1}$, the DFA $M^{j+1}$ that is consistent with $T^{j+1}$ must have at least $(N_j + 1)$. Thus, we have $N_{j+1} \geq N_j + 1$ and hence $\{N_j\}_{j \in \mathbb{N}}$ is a monotone increasing sequence. On the other hand, since for each $j \in \mathbb{N}$, $N_j$ is always upper-bounded by $0 < N_{\sup} < +\infty$. Therefore, $\{N_j\}_{j \in \mathbb{N}}$ is a monotone increasing sequence with a finite upper-bound. It follows immediately that $\{N_j\}_{j \in \mathbb{N}}$ is convergent. Since all the elements in $\{N_j\}_{j \in \mathbb{N}}$ are integers, it must converge to $N_{\sup}$ within a finite number of steps. \hfill \qed

Computational Complexity Next we provide the analysis of the computational complexity of the $L^*_C$ algorithm from both time and space perspectives.

Let $M$ be the number of states of the minimal DFA that accepts an unknown language $U \subseteq \Sigma^*$, $N$ be the maximal length of the counterexample strings and $K = |\Sigma|$, it has been proved in Theorem 2.4 that at most $(K + 1)(N + M(N - 1))N$ membership queries and at most $(N - 1)$ conjecture tests are sufficient for the $L^*$ algorithm to construct the correct DFA. Since the $L^*_C$ algorithm performs membership and equivalence tests in the same way as the $L^*$ algorithm does, we can also conclude that $(K + 1)(N + M(N - 1))N$ is a sufficient upper-bound of the time complexity of the $L^*_C$ algorithm, where in this algorithm $M$ denotes the number of states of the desired supervisor $S$. Compared to the conventional supervisor synthesis algorithms in [26, 91, 143], which admit a time complexity of
$O(N_G M)$ (where $N_G$ denotes the number of states of the plant $G$), the $L^*_C$ algorithm possesses a computational time complexity with the same order of $N$ ($O(N^2)$). Nevertheless, the above time complexity analysis is rather conservative, since the computation of the set $C$ can effectively eliminate the illegal behaviors and update the observation table in one iteration step. In addition to computational time complexity, compared to the algorithm presented in [91], whose computational space complexity if $O(N_G M)$, the $L^*_C$ algorithm only requires an $O(M)$ computational space, which is a significant improvement.

**Discussion: Supervisor Synthesis with Unknown Plant** In practice, control specifications are often defined by structured natural languages, unified modeling languages (UML) and logical formulas [203], rather than finite automata; and the construction of an appropriate automaton model requires experience in DES modeling [26] and turns out to be non-trivial. In addition to specifications, the uncontrolled DES plant in applications turns to be represented by a simulation input-output model or a “black-box”, whose source codes and/or formal model is not precisely accessible or becomes outdated as the systems evolves over time, rather than a precise DFA model. Therefore, the conventional supervisor synthesis methods cannot be applied directly to ensure the achievement of the specifications [33] in the absence of the formal model of either the plant or the specification. To apply the $L^*_C$ algorithm for a possibly unknown plant, the following assumption is made for the plant $G$ and specification language $L$ to solve Problem 3.1 in the absence of prior model knowledge of $G$.

**Assumption 3.1.** The strings generated by $G$ are “ergodic”, i.e., all the strings that could be generated by $G$ can be observed with an equal and nonzero probability.

Compared to the supervisor synthesis algorithms in the traditional Ramadge-Wonham supervisory control theory, the $L^*_C$ algorithm presented in this chapter can be applied for the case in which complete model knowledge of the plant $G$ may not be accessible a priori. In this case, the essential ingredients of the algorithm, including the set $C^j$ of uncontrollably illegal strings, and the sequence $\{K_j\}_{j \in \mathbb{N}}$ of languages that are used by the equivalence
queries $\xi^j$, should be approximated via various sampling-based methods, such as random walk and the $W$-method [32], instead of a complete search a priori. Since we assume that $G$ can be represented by a finite automaton, the approximation will eventually result in the same supervisor as the sampling procedure proceeds. In this case, the supervisor can be written as $S = L_C^\ast(\Sigma_{uc}, L)$.

### 3.2.5 An Illustrative Example

We use a supervisor synthesis example from [91] to validate the effectiveness of the $L_C^\ast$ algorithm.

**Example 3.1.** We consider the event set $\Sigma = \{\epsilon, a, b\}$, and the set of uncontrollable events is given by $\Sigma_{uc} = \{b\}$. The specification language is given by $L = a^n + a^kba^*$, where $k \leq 2$ and $n > 2$. The plant language is $L(G) = a^nba^*$. Note that $L$ is not controllable in this example. The DFA models corresponding to $L$ and $L(G)$ are depicted as follows, respectively.

![DFA Diagrams](image)

We aim at applying the $L_C^\ast$ algorithm to synthesize a correct and maximally permissive supervisor $S$ such that $L(S\parallel G) = \sup C(L)$. To demonstrate the execution of the $L_C^\ast$ algorithm step by step, we initialize $S = E = \{\epsilon\}$ and $C^0 = \emptyset$. By using the specification $L$ (which is prefix-closed) as the first membership function $T^0$, a closed and consistent observation table, namely $(S, E, T^0)$, is formed as Table 3.1, along with its corresponding conjecture DFA $M(S, E, T^0)$. Note that since we are interested in prefix-closed languages, we only list the elements $s \in S$ such that $T^0(s) = 1$, which leads the $L_C^\ast$ algorithm to
construct the conjecture DFA with all its states marked.

TABLE 3.1

\[ T^0 \text{ IN EXAMPLE 3.2.1} \]

\[
\begin{array}{c|cc}
T^0 & \epsilon & \\
\hline
S & \epsilon & 1 \\
S \Sigma - S & a & 1 \\
& b & 1 \\
\end{array}
\]

\[ M(S, E, T^0) : \quad \]

The DFA \( M^0 = M(S, E, T^0) \) is then presented to the Teacher as the conjecture supervisor. This conjecture is denied when a string \( t_0 = aaab \in \Sigma^* \) is generated by the Teacher. It is clear that \( t_0 \in L(M^0) - L = L(M^0) - K^0 \); therefore we have \( \tau_1^0(t_0) = 0 \) and according to (3.9), \( t_0 \) is a counterexample. In this case, we add \( \overline{T}_0 = \{\epsilon, a, aa, aaa, aaab\} \) to \( S \) to build up a new closed and consistent observation table. Furthermore, since \( aaa \in L \), \( b \in \Sigma_{uc} \) but \( aaab \notin L \), \( t_0 = aaab \) is uncontrollably illegal and hence we obtain \( C^1 = \{t_0\} \). Based on \( C^1 \) and membership function \( T^1 \), the updated observation table \((S, E, T^1)\) is constructed as shown in Table 3.2. Note that we also keep the elements \( s \in S \) such that \( T^1(s) = 1 \) in Table 3.2 so that \( L(M^1) = L_m(M^1) \).
TABLE 3.2

*T1 IN EXAMPLE 3.2.1*

<table>
<thead>
<tr>
<th>$T^1$</th>
<th>$\epsilon$</th>
<th>a</th>
<th>aa</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$\epsilon$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S\Sigma - S$</th>
<th>b</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>aab</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The $L^*_C$ algorithm then follows (2.17) to construct an updated DFA $M^1 = M(S, E, T^1)$ as shown below. $M^1$ is then used as the conjecture supervisor. Consider a string $t_1 = \text{baaa} \in \Sigma^*$ at this point. Note that $t_1 \in L - L(M^1) = K^0 - L(M^1)$, hence $t_1 \in K^1 - L(M^1)$ must be satisfied as $K^1 \subseteq K^0$. Therefore $\xi^1(t) = 0$ and $t_1$ is a counterexample (which in fact implies that the conjecture supervisor $M^1$ is not maximally permissive). Since $t_1$ is not uncontrollably illegal, we do not update $C^1$ at this iteration step. The strings in $\overline{T}_1 = \{b, ba, baa, baaa\}$ is added to $S$ in order to update the observation table, which is denoted as $(S, E, T^2)$.
The $L_C^*$ algorithm makes $(S, E, T^2)$ closed and consistent, and the observation table is shown in Table 3.3, along with the corresponding DFA $M(T^2)$. This time no more counterexamples are detected and we can conclude that $M(T^2)$ recognizes the supremal controllable sublanguage $\sup C(L)$ of $L$, which coincides with the result obtained in [91].
3.3 Learning-based Supervisor Synthesis under Partial Observation

In this section, we go beyond the completely-observed DES plants and consider the situation where the supervisor may be constrained to observe only events in a specified set of observable events that can be generated by the plant. Such situation may correspond to limited deployment of sensors in practice, in which only crucial information of the uncontrolled system’s status are available to the controller. In this section, we start our study with an introduction of the Ramadge-Wonham’ supervisory control framework under partial observation, followed by the learning-based supervisor synthesis approach that deals with a partially-observed plant.

3.3.1 Supervisory Control under Partial Observation

When the DES plant can only be observed partially, the event set $\Sigma$ is partitioned into two disjoint subsets $\Sigma = \Sigma_o \cup \Sigma_{uo}$ where $\Sigma_o$ is the set of observable events, whose occurrence can be detected by the supervisor, and $\Sigma_{uo}$ is the set of unobservable events, which cannot be seen by the supervisor. The presence of partial observation can be captured by the natural projection $P_o$ from $\Sigma^*$ to $\Sigma_o^*$ (ref. Definition 2.4). In this case, the feedback control loop under partial observation is depicted in Fig. 3.2.

![Figure 3.2. The feedback loop of supervisory control under partial observation](image)
The supervisor $S_P : P_o[L(G)] \rightarrow \Gamma$ cannot distinguish between two strings $s_1$ and $s_2$ as long as they are observationally indistinguishable, i.e., $P_o(s_1) = P_o(s_2)$. Since the supervisor $S_P$ takes its control actions based on observed strings, it must take identical control action following all strings that have identical projected values. The closed-loop system of $G$ under the control issued by $S_P$, denoted as $S_P/G$, is defined analogously to the case of complete observation.

**Definition 3.5.** The language generated by $S_P/G$ is defined recursively as follows:

(i) $\epsilon \in L(S_P/G)$;

(ii) $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)[s \in L(S_P/G) \land s\sigma \in L(G) \land \sigma \in S_P(P_o(s)) \iff s\sigma \in L(S_P/G)]$.

The language marked by $S_P/G$ is defined as $L_m(S_P/G) = L(S_P/G) \cap L_m(G)$.

In addition to being $\Sigma_{uc}$-enabling and non-marking, a DFA-based supervisor $S_P$ can equivalently represent the mapping based partial observation supervisor $S_P$ in Fig. 3.2 if it is also $P_o$-compatible.

**Definition 3.6 ($P_o$-compatible Supervisors).** [91] The DFA-based supervisor $S_P$ is said to be $P_o$-compatible with respect to $G$ and $P_o$ if

$$(\forall s, t \in L(S_P||G))(\forall \sigma \in \Sigma)[P_o(s) = P_o(t) \land \sigma \in L(S_P||G) \land 
\text{t} \sigma \in L(G) \Rightarrow \text{t} \sigma L(S_P||G)].$$ (3.13)

In the sequel, we use $L(S_P||G)$ and $L_m(S_P||G) = L(S_P||G) \cap L_m(G)$ to replace $L(S_P/G)$ and $L_m(S_P/G)$, respectively, provided that $S_P$ is $\Sigma_{uc}$-enabling, non-marking and $P_o$-compatible.

Given a specification language $L \subseteq L(G)$, the supervisory control problem aims at designing a supervisor $S$ such that $L(S||G) = L$. Towards this end, we introduce the following properties of regular languages to solve the supervisory control problem under partial observation.
Definition 3.7 (Observable and Normal Languages). A language $L \subseteq \Sigma^*$ is said to be

- **Observable** with respect to $G$ and $P_0$ [34, 110] if $(\forall s, t \in \overline{L}, \sigma \in \Sigma) [P_0(s) = P_0(t) \land s\sigma \in L \land t\sigma \in L(G) \Rightarrow t\sigma \in \overline{L}]$;

- **Normal** [34, 110] if $P_0^{-1}(P_0(\overline{L})) \cap L(G) = \overline{L}$;

Intuitively, $L \subseteq \Sigma^*$ is said to be normal also if $(\forall s, t \in L(G)) [s \in \overline{L} \land t \in L(G) \land P_0(s) = P_0(t) \Rightarrow t \in \overline{L}]$. Hence normality is a stronger property than observability [26].

If the plant behavior is partially observed, a $\Sigma_{uc}$-enabling, non-marking and $P_0$-compatible supervisor $S_P$ can be constructed to restrict $L(S_P\parallel G)$ to the desired non-empty and prefix-closed specification $L \subseteq L(G)$ if and only if $L$ is controllable and observable; and a supervisor $S_P$ exists such that the marked language $L_m(S\parallel G) = L$ for any non-empty language $L \subseteq L_m(G)$ if and only if $L$ is controllable, observable and $L_m(G)$-closed.

3.3.2 Learning-based Synthesis of DES Supervisors under Partial Observation

With the development of the $L^*_C$ algorithm for supervisor synthesis under complete observation, a natural next step is to check the availability of applying the $L^*_C$ algorithm for the synthesis of supervisors that steers a partially-observed DES plant to meet a regular specification. For such a pursuit, we now formulate the following supervisory control of a DES plant under partial observation (SCPO) problem.

**Problem 3.2 (SCPO).** Consider a DES plant $G$ (3.4) over $\Sigma$ and a prefix-closed specification language $L \subseteq L(G)$. Let $\Sigma_{uc} \subseteq \Sigma$ be the set of uncontrollable events, $\Sigma_o \subseteq \Sigma$ be the set of observable events and $P_0 : \Sigma^* \rightarrow \Sigma_o^*$ be the associated natural projection, find a $P_0$-compatible, $\Sigma_{uc}$-enabling and non-marking supervisor $S_P$ for $G$ such that

(i) **Correctness:** $L(S_P\parallel G) \subseteq L$;

(ii) **Maximal permissiveness:** there does not exist a supervisor $S'_P$ satisfying (i) such that $L(S_P\parallel G) \subset L(S'_P\parallel G)$. 

53
Similar to Problem 3.1, the solution of Problem 3.2 is reduced to the computation of the prefix-closed, controllable and observable sublanguages of the given specification language $L$ with respect to the plant $G$. Nevertheless, observability of languages does not necessarily preserve under union and thus, the class of prefix-closed and observable sublanguages of a given language need not possess a supremal element\[26\] in general, which implies that unlike Problem 3.1, there does not exist a supremally permissive supervisor for Problem 3.2. Despite this discouraging result, various approaches have been proposed in the literature to compute correct supervisors that may not be maximally permissive. One of these alternative approaches relies on the property of normality (cf. Definition 3.7). Due to the fact that language normality is preserved under arbitrary union, the supremal normal sublanguage of a given language does uniquely exist. Therefore, a correct solution of Problem 3.2 is a partial-observation supervisor $S_P$ that steers the plant $G$ to meet the supremal controllable and normal sublanguage of $L$, denoted as $\text{sup} \ CN(L)$. Under the assumption that $\Sigma_c \subseteq \Sigma_o$, the controllability and observability properties together imply normality and in this case, $\text{sup} \ CN(L)$ is the supremal solution of Problem 3.2. Nevertheless, normality is stronger notion than observability (a normal language is always observable) in general, and deployment of the aforementioned supervisor hence may lead to more conservative closed-loop behaviors of the plant.

In this subsection, we develop another alternative approach that solves Problem 3.2 in a more permissive manner than $\text{sup} \ CN(L)$. The synthesis procedure consists of the following two steps.

(i) **Offline synthesis:** we first assume that the plant is completely observed, i.e., $\Sigma_o = \Sigma$ and $\Sigma_{uo} = \emptyset$. Next, we employ the $L^*_C$ algorithm to synthesize a supremally permissive supervisor $S = L^*_C(G, \Sigma_{uc}, L)$ offline such that $L(S||G) = \text{sup} \ C(L)$ under complete observation.

(ii) **Online implementation:** Once the complete-observation supervisor $S$ is computed by the $L^*_C$ algorithm, we incorporate $S$ with the observable events $\Sigma_o$ to implement a $P_\sigma$-compatible and correct supervisor $S_P$ online based on the partially-observed behaviors generated by the plant.
For convenience of presentation, we assume that the complete-observation supervisor has already been computed a priori to achieve $\sup C(L)$ by assuming all the events in $\Sigma_{uo}$ are observable. To this regard, we further assume that $S = L^*_C(G, \Sigma_{uc}, L)$ can be modeled as by the following DFA

$$S = (X, \Sigma, \zeta, x_0).$$  \hfill (3.14)

Note that the marked states are investigated in neither Problem 3.1 nor Problem 3.2, the obtained supervisor $S$ need not admit a set $X_m$ of marked states.

For any subset $X' \subseteq X$, the set of $\sigma$-reachable states from $X'$ with an observable event $\sigma \in \Sigma_o$ is defined as

$$OR_\sigma(X') = \{ x \in X | (\exists x' \in X') [x = \zeta(x', \sigma)] \}. \hfill (3.15)$$

On the other hand, the unobservable reach of the subset of states $X' \subseteq X$ and the subset of events $\Sigma' \subseteq \Sigma$ is defined as

$$UR_{\Sigma'}(X') = \{ x \in X | (\exists x' \in X', w \in (\Sigma' \cap \Sigma_{uo})^*) [x = \zeta(x', w)] \}, \hfill (3.16)$$

and we write $UR(X')$ if $\Sigma' = \Sigma$.

The partial-observation supervisor $S_P$ that solves Problem 3.2 can then be extracted from $S$ by using the aforementioned notations. Specifically, $S_P$ is expressed by an accessible DFA

$$S = \text{trim}(X_P, \Sigma_o, \zeta_P, x_{P,0}), \hfill (3.17)$$

where $\text{Ac}$ denotes the operator of computing the trim part of a given DFA, and

$$X_P = 2^X,$$

$$x_{P,0} = UR(x_0) = UR(\{x_0\}), \hfill (3.18)$$

$$\forall x_P \in X_P, \sigma \in \Sigma_o : \zeta_P(x_P, \sigma) = UR(OR(x_P, \sigma)).$$
Let $\Omega(L)$ denote the closed-loop language of the plant $G$ under the supervision of $S + P$, i.e., $\Omega(L) = L(S_P\|G)$, the following theorem states that $\Omega(L)$ is in general a more permissive solution of Problem 3.2 than $\sup CN(L)$.

**Theorem 3.3.** [72] If $\sup C(L) \neq \emptyset$, then

$$\sup CN(L) \subseteq \Omega(L) \subseteq \sup C(L).$$

(3.19)

Note that the aforementioned supervisor synthesis approach does not pose additional requirements on the set of controllable events $\Sigma_c$ or the set of observable events $\Sigma_o$.

**Remark 3.1.** The aforementioned extraction procedure to synthesize $S_P$ admits a computational complexity of $O(|X||X_P|) = O(|X||2^X|)$ [72].

3.3.3 An Illustrative Example

We use a simple example to illustrate the computation procedure of the proposed supervisor synthesis method and demonstrate its permissiveness.

**Example 3.2.** We consider the event set $\Sigma = \{\alpha, \beta, \gamma\}$, and the plant $G$ is given such that $L(G) = \alpha(\beta\gamma + \gamma\beta)$. The specification language is given by $L = \alpha\beta + \alpha\gamma$. The plant model and the DFA representation of the specification are given in Fig. 3.3 (a) and (b), respectively.

![Fig. 3.3. The DFAs of $G$ and $L$.](image-url)
In this example, the set of controllable events is given by $\Sigma_c = \Sigma = \{\alpha, \beta, \gamma\}$, while the set of observable events is the singleton $\Sigma_o = \{\alpha\}$. We first assume that $\Sigma_o = \Sigma$ and synthesize the complete-observation supervisor $S$ to fulfill $\sup C(L)$. Since all the events are assumed to be controllable, $\sup C(L) = L$ and $S$ can be realized as shown in Fig. 3.4.

![Diagram](a) The supervisor $S$ ![Diagram](b) The supervisor $S_P$

Figure 3.4. The DFAs of $S$ and $S_P$.

Next, we involve the unobservable events $\Sigma_{uo} = \{\beta, \gamma\}$ and follow the computation procedure (3.18) to extract $S_P$ from $S$. The partial-observation supervisor $S_P$ is hence obtained such that $L(S_P \parallel G) = \overline{a}$. On the other hand, one can compute that $\sup CN(L) = \{\epsilon\}$. Therefore $\sup CN(L) \subseteq \Omega(L) = L(S_P \parallel G)$ and $S_P$ is a correct supervisor that is strictly more permissive than $\sup CN(L)$.

3.4 Learning-based Synthesis of DES Supervisors for Non-terminating Specifications

3.4.1 Research Motivation

So far we have restricted the discussion to behaviors consisting of finite length strings. However, many properties to be ensured for an ACPS may not be described by a formal language of finite length. As the following example implies, liveness properties of a system, which require that some desired conditions should be satisfied eventually by the system,
cannot be expressed in the form of regular languages in general.

**Example 3.3.** Consider a communication system that is composed of a pair of transmitter and receiver. The event set of $G$ is given by $\Sigma = \{\text{trans}, \text{rec}\}$.

![Figure 3.5. The communication system $G$.](image)

We require that every transmitted message must be eventually received. Therefore, the desired specification of the communication system can be captured by the Büchi automaton shown in Fig. 3.5, which accepts all the infinite strings over $\Sigma$ such that every occurrence of the event trans must be followed by an occurrence of the event rec. Since each finite length string of the communication system can be extended in a manner such that the above constraint is satisfied, this constraint hence imposes no restriction on finite length behaviors and cannot be specified by a finite automaton.

Example 3.1.1 hence highlights the need of a formal supervisor synthesis framework to deal with non-terminating control objectives, which will be discussed in detail in the rest of this section.

### 3.4.2 Supervisory Control of Non-terminating Behaviors

The supervisor synthesis problem for controlling sequential or non-terminating behaviors was first studied by Ramadge [141], Thistle and Wonham [176] for DES plants that could be modeled by $\omega$-automata. In this dissertation, we follow the framework developed in [141] which characterized the sequential behavior of a plant and the closed-loop system.
as deterministic Büchi automata, respectively. We start from the supervisory control under complete observation and assume that the uncontrolled plant $G$ is modeled by a DBA. The control paradigm that drives a completely-observed DBA to meet an $\omega$-regular language specification is similar as shown in Fig. 3.1, in which we also assume that the supervisor $S$ is a DBA. With slightly abusing the notations, we also denote $S\|G$ as the closed-loop system of the plant $G$ under the supervision of $S$.

The following lemma characterizes $\mathcal{L}(S\|G)$, which stands for the closed-loop sequential behaviors of $S\|G$ when both $S$ and $G$ are given as DBAs.

**Lemma 3.2.** [92] The $\omega$-strings generated by the closed-loop system $S\|G$ is given by

$$\mathcal{L}(S\|G) = \lim(\mathcal{L}(S\|G)) \cap \mathcal{L}(G),$$

(3.20)

where $\mathcal{L}(G)$ is defined according to (2.15) and $\mathcal{L}(S\|G)$ represents the regular languages generated by the parallel composition of $S$ and $G$ by viewing both of them as DFAs.

Since both $S$ and $G$ are deterministic, the conclusion presented in Lemma 3.2 can be inferred from (2.16) immediately. The following example is adopted to illustrate the application of Lemma 3.2.

**Example 3.4.** Consider a plant $G$ whose event set $\Sigma$ of $G$ is given by $\Sigma = \{a, b, c, u\}$.

![Diagram](image)

**Figure 3.6.** The plant $G$ with the specification $S$. 

59
The transition diagram of the plant $G$ is shown in Fig. 3.6 (a). In this example, by viewing $G$ as a DFA, we have $L(G) = (a+b)^*ca^*(b+u)a^*$ and $L_m(G) = (a+b)^*ca^*(b+u)a^*$. Now we consider another automaton $S$ over $\Sigma$ whose transition relation is illustrated in Fig. 3.6 (b). If $S$ is a DFA, then it can be verified that $H \subseteq G$, along with $L(S) = (ca + (ab)^* acu)a^*a^*$ and $L_m(S) = (ca + (ab)^* acu)a^*a^*$. Therefore, we can write that $L(S \parallel G) = L(S) = (ca + (ab)^* acu)a^*a^*$.

On the other hand, by viewing both $G$ and $S$ as DBAs, the $\omega$-regular languages accepted by $G$ and $H$ are given by $L(G) = (a+b)^*ca^*(b+u)a^\omega$ and $L(S) = (ca + (ab)^* acu)a^\omega$, respectively. Thus, according to Lemma 3.2, we have

$$L(S \parallel G) = \lim(L(S \parallel G)) \cap L(G) = \lim(L(S)) \cap L(G)$$

$$= \lim((ca + (ab)^* acu)a^*) \cap ((a+b)^*ca^*(b+u)a^\omega)$$

$$= (ab)^*(ca + (ab)^* acu)a^\omega \cap ((a+b)^*ca^*(b+u)a^\omega)$$

$$= (ca + (ab)^* acu)a^\omega.$$
desired specifications, i.e., \( \mathcal{L}(S \parallel G) = \mathcal{L} \) if and only if \( \mathcal{L} \) is \( \omega \)-controllable with respect to \( G \) and \( \Sigma_{uc} \) and \( \omega \)-closed relative to \( \mathcal{L}(G) \). If \( \mathcal{L} \) is not \( \omega \)-controllable but \( \omega \)-closed relative to \( \mathcal{L}(G) \), then [141] further pointed out that the unique supremal \( \omega \)-controllable sublanguage of \( \mathcal{L} \), denoted as \( \sup C(\mathcal{L}) \), always exists. Thus synthesis of a maximally permissive supervisor is possible.

To consider the supervisory control of non-terminating behaviors under partial observation, we inherit the control loop shown in Fig. 3.2 and define the following observation projection \( P_\omega : \Sigma^\omega \rightarrow \Sigma_\infty^\omega \) for infinite string.

**Definition 3.9 (\( \omega \)-projection).** For an \( \omega \)-string \( w \in \Sigma^\omega \), let \( \{s_n\}_{n \in \mathbb{N}} \subseteq \Sigma^* \) be a strictly monotone sequence of finite prefixes of \( w \), i.e., \( s_n < s_{n+1} < w \) for all \( n \in \mathbb{N} \) and \( \lim_{n \rightarrow \infty} s_n = w \). Then the observation of the \( \omega \)-string \( w \) through the projection \( P_\omega(w) \) is defined to be

\[
v := P_\omega(w) = \lim_{n \rightarrow \infty} P_o(s_n) \in \Sigma_\infty^\omega,
\]

where \( P_o \) is the standard natural projection from \( \Sigma^* \) to \( \Sigma_\infty^* \).

The inverse projection \( P_\omega^{-1} \) is defined by \( P_\omega^{-1}(v) = \{ w \in \Sigma^\omega | P_\omega(w) = v \} \). The notions of \( \omega \)-observability and \( \omega \)-normality are then defined as follows, respectively.

**Definition 3.10 (\( \omega \)-observable and \( \omega \)-normal Languages).** An \( \omega \)-language \( \mathcal{L} \subset \Sigma^\omega \) is said to be

- \( \omega \)-Observable with respect to \( G \) and \( P_\omega \) if \( \overline{\mathcal{L}} \) is observable with respect to \( G \) (viewed as a DFA) and \( P_o \);
- \( \omega \)-Normal with respect to \( G \) and \( P_\omega \) if \( \overline{\mathcal{L}} \) is normal with respect to \( G \) (viewed as a DFA) and \( P_o \).

For a non-empty and non-terminating control specification \( \mathcal{L} \subseteq \mathcal{L}(G) \), there exists a nonblocking partial-observation supervisor \( S_P \) such that \( \mathcal{L}(S_P \parallel G) = \mathcal{L} \) if and only if \( \mathcal{L} \) is \( \omega \)-controllable, \( \omega \)-observable and \( \omega \)-closed relative to \( \mathcal{L}(G) \) [141]; if the desired behavior is
not \( \omega \)-observable, a supervisor \( S_P \) can be synthesized so that the closed-loop system generates the supremal \( \omega \)-controllable and \( \omega \)-normal sublanguage of the specification language \( L \) [92].

### 3.4.3 Learning-based Supervisor Synthesis Algorithms for \( \omega \)-Regular Specifications

In this section, we assume that the uncontrolled DES plant is modeled by a deterministic Büchi automaton

\[
G = (Q, \Sigma, \delta, q_0, F),
\]  

(3.23)

where the event set is partitioned as \( \Sigma = \Sigma_c \cup \Sigma_{uc} = \Sigma_o \cup \Sigma_{uo} \). We are interested in synthesizing appropriate supervisors that can also be represented by DBA. Therefore, the supervisory control loop for the plant \( G \) (3.23) is similar as shown in Fig. 3.1 under complete observation and Fig. 3.2 under partial observation, respectively. The infinite behaviors generated by the controlled plant \( S || G \), which is denoted as \( L_\omega(S||G) \), is clarified in Lemma 3.2 and is re-stated as follows.

\[
L(S||G) = \lim(L(S||G)) \cap L(G),
\]

(3.24)

where \( L(G) \) is the \( \omega \)-language accepted by \( G \) and the notation \( L(S||G) \) indicates the regular languages generated by the parallel composition of \( S \) and \( G \) by viewing both of them as DFAs (with \( Q_m = F \)).

Based on the aforementioned plant models and preliminary notations, we now formulate the basic supervisor synthesis problems for a DBA plant in order to achieve \( \omega \)-regular language specifications (BSCP\( \omega \)) under complete observation as follows.

**Problem 3.3 (BSCP\( \omega \)).** Consider a plant \( G \) (3.23) that can be modeled by a DBA (3.23) over \( \Sigma \) and a topologically closed specification language \( L \subseteq L(G) \). Let \( \Sigma_{uc} \subseteq \Sigma \) be the set of uncontrollable events, synthesize a \( \Sigma_{uc} \)-enabling, non-marking and \( \omega \)-non-blocking supervisor \( S \) such that \( L(S||G) \) satisfies \( L \) in a maximally permissive manner.
Remark 3.2. The requirement that $\mathcal{L} \subseteq \mathcal{L}(G)$ is topologically closed in Problem 3.3 indeed suggests that $\mathcal{L}$ is $\omega$-relative-closed with respect to $\mathcal{L}(G)$.

We first consider the case in which $G$ can be completely observed and aim at solving Problem 3.3. It is shown in [92, 141] that a $\Sigma_{uc}$-enabling, non-marking and $\omega$-non-blocking supervisor $S$ exists to solve Problem 3.3 if and only if $\mathcal{L}$ is $\omega$-controllable with respect to $G$ and $\Sigma_{uc}$ and $\omega$-closed relative to $\mathcal{L}(G)$. The following proposition provides us with the characterization of the maximally permissive solution of the BSCP$^\omega$ problem.

Proposition 3.1. [141] For the DES plant $G$ (3.23) and a non-empty $\omega$-regular language $\mathcal{L} \subseteq \mathcal{L}(G)$, if $\mathcal{L}$ is $\omega$-relative-closed with respect to $\mathcal{L}(G)$, then there exists a unique supremal $\omega$-controllable language, written as $\text{sup } C(\mathcal{L})$, contained in $\mathcal{L}$.

Proposition 3.1 implies that whenever $\mathcal{L}$ is $\omega$-relative-closed with respect to $\mathcal{L}(G)$, there exists a unique maximally permissive solution of Problem 3.3; and according to Remark 3.2, this requirement can be fulfilled. Now recall Definition 3.8, the finite prefix-closure of an $\omega$-controllable language forms a controllable regular language. Therefore, for the topologically closed specification $\mathcal{L} \subseteq \mathcal{L}(G)$, the computation of $\text{sup } C(\mathcal{L})$ can be performed based only on the prefix-closed regular language $\overline{\mathcal{L}}$. To this regard, we introduce the following theorem which establishes the connection between $\omega$-controllable $\omega$-regular languages and controllable and deadlock-free regular languages.

Theorem 3.4. [92] Let $\mathcal{L} \subseteq \mathcal{L}(G)$ be topologically closed, then

$$\text{sup } C(\mathcal{L}) = \lim(\text{sup } CD(\overline{\mathcal{L}})),$$  \hspace{1cm} (3.25)

where $\text{sup } CD(\cdot)$ is the operator for computing the supremal controllable and deadlock-free (cf. Section 2.2.1) sublanguage of a given regular language.

Theorem 3.4 indicates that, in order to synthesize a supervisor that drives the closed-loop behaviors of the plant to accomplish the specification $\mathcal{L}$ in a maximally permissive
manner, we may first compute the supremal controllable and deadlock-free sublanguage of the prefix-closed regular language $\overline{L}$ and then the desired DBA-based supervisor $S$ can be obtained by taking the limit language of $\sup CD(\overline{L})$. Note that the language $\sup CD(\overline{L})$ can be proved to exist uniquely [91] and can be computed in a modular fashion [121]:

$$\sup CD(\overline{L}) = \sup C(\sup D(\overline{L})).$$

(3.26) provides us with insights of solving Problem 3.3 by synthesizing a DBA-based supervisor $S$ that accepts the supremal topologically closed and $\omega$-controllable sublanguage $\sup C(\mathcal{L})$. Based on (3.26), computation of $\sup D(\overline{L})$ can be accomplished by modifying the $L^*$ learning algorithm.

For a regular language $L \subseteq \Sigma^*$, it can be inferred immediately from the definition of deadlock-free languages that $L$ is deadlock-free if and only if the following inequality holds:

$$L \subseteq \overline{L}/\Sigma,$$

(3.27)

which implies that the supremal deadlock-free sublanguage $\sup D(L)$ of $L$ uniquely exists. Furthermore, it follows from Corollary 2.2 in [91] that $\sup D(L)$ can be obtained via the following iterative scheme:

$$K^0_D := \overline{L},$$

$$K^{j+1}_D := K^j_D - (\Sigma^* - K^j_D/\Sigma)\Sigma^*, \quad j \geq 1.$$

Next, we develop a second modification of the $L^*$ algorithm, written as the $L^*_D$ algorithm. Similar to the $L^*_C$ algorithm, the $L^*_D$ algorithm gains the inspiration from the iterative computation procedure in (3.28) and utilizes the following dynamical membership queries in order to compute $\sup D(L)$ via automata learning: initially, for $t \in \Sigma^*$ and

\[5\]

Here the subscript “D” indicates the pursuit of deadlock-free sublanguages.
Furthermore, as an analogy to the set $C$ that is deployed by the $L_C^*$ algorithm, we define the set
\[
C_D = \{ t \in \Sigma^* | (\forall \sigma \in \Sigma)(\exists s \in \mathcal{L})[(t = s\sigma) \land (t \notin \mathcal{L})] \}. \tag{3.30}
\]
In other words, $C_D$ is the collection of illegal (with respect to $L$) strings over $\Sigma$ which become illegal due to the last event. For $j \in \mathbb{N}$, we denote by $C_D^j$ the set of the strings after the $j$-th string with an illegal last event is added. Therefore, the Teacher of the $L_D^*$ algorithm updates $C_D^{j+1}$ by setting $C_D^{j+1} = C_D^j \cup \{t^{j+1}\}$. We also define $D_D(\cdot)$ as an operator that returns the prefix of a string in $C_D$ by discarding its last event. Hence, we can write that
\[
D_D(C_D) = \{ s \in \mathcal{L} | (\forall \sigma \in \Sigma)[s\sigma \in C_D]\}. \tag{3.31}
\]
With the help of the set $C_D$ and the operator $D_D(\cdot)$, we can now define the membership queries when the iteration step $j \geq 1$, as follows.
\[
T_D^j(t) = \begin{cases} 
0, & \text{if } T_{D}^{j-1}(t) = 0 \text{ or } t \in D_D(C_D)^* \\
1, & \text{otherwise.}
\end{cases} \tag{3.32}
\]
It is clear that an observation table that is consistent with the membership function $T_D^j$ will be $\mathcal{L} - D_D(C_D)^*\Sigma^*$, which is identical to the iterative procedure in (3.28) and can be used to compute $\sup D(L)$. The $L_D^*$ learning algorithm constructs an observation table $(S, E, T_D^j)$ by answering the membership queries (3.29) and (3.32). Once the $L_D^*$ algorithm makes the observation table closed and consistent, a conjecture DFA $S_D^j = M(S, E, T_D^j)$ is constructed by following the procedure in (2.17). For convenience of presentation, we assume without loss
of generality that \( S_j^D \) can be represented in the form of the following DFA:

\[
S_j^D = (X_D, \Sigma, \zeta_D, x_D, X_{D,m}).
\]

The equivalence queries are then performed with respect to \( S_d \). More specifically, the equivalence queries are determined by the function \( \xi_j^D : \Sigma^* \rightarrow \{0, 1\} \) \((j \in \mathbb{N})\), which are specified as follows.

\[
\xi_j^D(t) = \begin{cases} 
0, & \text{if } t \in L_m(S_j^D) \Delta L, \\
1, & \text{otherwise}. 
\end{cases}
\]

Once \( \xi_j^D(t) = 0 \) for \( t \in \Sigma^* \), then \( t \) is returned by the Teacher as a counterexample and the \( L_*^D \) algorithm processes \( t \) in the same way as the \( L^* \) algorithm. In particular, if a string \( t \) satisfies: (i) \( t \in L_m(S_j^D) \); (ii) \( \exists x \in X_D \) \([\zeta_D(x_D, t) = x] \); (iii) \( \forall \sigma \in \Sigma \) \([\zeta_D(x, \sigma) = \emptyset] \), then \( t \) will be added to \( C_j^D \) so that the updated \( C_j^D \) can be obtained.

The above computation procedure is summarized in Algorithm 3, in which the notation \( L_*^D(L) \) indicates that a minimal DFA that recognizes \( \sup D(L) \) is obtained.

---

**Algorithm 3: The \( L_*^D(L) \) learning algorithm**

**Input:** The language \( L \) and the event set \( \Sigma \).

**Output:** A minimal DFA \( S_D \) such that \( L_m(S_D) = \sup D(L) \).

1: \( S \leftarrow \{\epsilon\}, E \leftarrow \{\epsilon\}, j \leftarrow 0, C_0^D \leftarrow \emptyset \)

2: Construct \((S, E, T_j^D)\) with membership function \( T_j^D \) in (3.29) and (3.32)

3: repeat

4: \[\text{while } (S, E, T_j^D) \text{ is not closed or consistent do}\]

5: \[\text{if } (S, E, T_j^D) \text{ is not closed then}\]

6: \[\text{Find } s \in S, \sigma \in \Sigma \text{ such that } \forall s' \in S : \text{row}(s') \neq \text{row}(ss)\]

7: \[S \leftarrow S \cup \{s\sigma\}\]

8: \[\text{end if}\]

9: \[\text{end while}\]

10: \[\text{end repeat}\]
if \((S, E, T^j_D)\) is not consistent then

\[
\text{Find } s_1, s_2 \in S, \sigma \in \Sigma \text{ and } e \in E \text{ such that } \text{row}(s_1) = \text{row}(s_2) \text{ but } T^j_D(s_1\sigma e) \neq T^j_D(s_2\sigma e)
\]

\[
E \leftarrow E \cup \{\sigma e\}
\]

end if

end while

Construct \(S^j_D = M(S, E, T^j_D)\) based on the procedure in (2.17)

Perform the equivalence queries (3.34) among \(L\) and \(L_m(S^j_D)\)

if \(t \neq \epsilon\) and \(\xi^j_D(t) = 0\) then

\[
S \leftarrow S \cup \{t\}
\]

\[
j \leftarrow j + 1
\]

if \(t \in L_m(S^j_D)\) and \((\exists x \in X_D)[\zeta^j_D(x_D, t) = x]\) and \((\forall \sigma \in \Sigma)[\zeta^j_D(x, \sigma) = \varnothing]\)

then

\[
C^{j+1}_D \leftarrow C^j_D \cup \{t\}
\]

end if

end if

end if

end while

\(S_D \leftarrow S^j_D\), return \(S_D\)

Inspired by Theorem 3.4 and the the conclusion presented in (3.26), we can now proceed to the computation of \(\text{sup} \ C(L)\) of the \(\omega\)-regular specification \(L \subseteq L(G)\) based on the application of the \(L_C^*\) and the \(L_D^*\) algorithms. The computation scheme, denoted as \(L_C^\omega\)

1. Compute the finite prefix-closure \(\overline{L}\) of the given \(\omega\)-regular specification \(L \subseteq L(G)\).

2. For the prefix-closed and regular language \(\overline{L}\), construct a DFA \(G_D\) that recognizes \(\overline{L}\), i.e., \(L_m(G_d) = L(G_d) = \overline{L}\).

3. Given \(L_m(G_d) = L(G_d) = \overline{L}\), we apply the \(L_D^*\) algorithm (Algorithm 3) with the membership queries (3.29) and (3.32) as well as the counterexample queries (3.34) to construct a minimal DFA \(S_d\) such that \(L_m(S_d) = L(S_d) = L_D^*(\overline{L}) = \text{sup} \ D(\overline{L})\).

4. Facing the prefix-closed language \(\text{sup} \ D(\overline{L}) \subseteq L(G_d)\), we take \(\Sigma_{uc} \subseteq \Sigma\) into consideration and apply the \(L_C^*\) algorithm (Algorithm 2) in order to obtain a supervisor \(S_c = L_C^*(G_d, \Sigma_{uc}, L(S_d))\).

5. According to (3.26), the parallel composition of \(S_c\) and \(S_d\) should result in

\[
L(S_d || S_c) = \text{sup} \ C(L(S_d)) = \text{sup} \ C(\text{sup} \ D(\overline{L})) = \text{sup} \ CD(\overline{L}).
\]
6. Construct a DBA-based supervisor $S$ such that $\mathcal{L}(S) = \lim L(S_c \| S_d)$.

7. It then follows from Theorem 3.4 that $S$ is the desired supervisor.

![Diagram](image)

Figure 3.7. The $L^\omega_C$ supervisor synthesis procedure.

Note that the correctness and finite convergence of Algorithm 3 can be guaranteed in the same way as the $L^\ast_C$ algorithm. Therefore, the overall computation procedure $L^\omega_C$ can be guaranteed to be correct and finitely convergent by Theorem 3.1 and Theorem 3.4.

We use the following example to demonstrate the effectiveness of the $L^\omega_C$ procedure.

**Example 3.5.** We now consider the DBA plant $G$ and the DBA $S$ as given in Example 3.4 with the set of uncontrollable events being given by the singleton $\Sigma_{uc} = \{u\}$. We make an additional assumption that all the states in both $G$ and $S$ are marked, as shown in Fig. 3.8 (a) and (b), respectively. Therefore, $L(G) = L_m(G) = (a + b)^*ca^*(b + u)a^*$ and $L(S) = L_m(S) = (ca + (ab)^*ac)a^*$. Furthermore, $L(G) = (a + b)^\omega + (a + b)^*ca^\omega + (a + b)^*ca^*(b + u)a^\omega$ and $\mathcal{L}(S) = (ab)^\omega + (ca + (ab)^*ac)a^\omega$.

68
Let the non-terminating control specification $\mathcal{L}$ be given by $\mathcal{L} = \mathcal{L}(S)\mathcal{L}(G)$. From Fig. 3.8 (b), it can be verified that $\overline{\mathcal{L}} = L(S)$ in this example. Therefore, we can write that

$$\lim(\overline{\mathcal{L}}) = \lim(L(S)) = \mathcal{L}(S) = \mathcal{L},$$

which implies that $\mathcal{L}$ is topologically closed. It hence follows from Remark 3.3.1 that $\mathcal{L}$ is $\omega$-relative-closed with respect to $\mathcal{L}(G)$. Therefore, the existence of the maximally permissive supervisor is guaranteed and from Theorem 3.4 and (3.26), the desired supervisor should steer $G$ to achieve

$$\sup C(\mathcal{L}) = \lim \sup C(\sup D(\overline{\mathcal{L}})) = \lim \sup C(\sup D(L(S))).$$

Hence the supervisor can be computed via the $L^\omega_C$ procedure. We first apply the $L^*_D$ algorithm for the computation of $\sup D(L(S))$. Note that for each string $s \in L(S) = \overline{L}(S)$, the inclusion $\overline{s} \subseteq s/\Sigma$ is satisfied, since it can be observed from Fig. 3.8 that for each state of $S$, there always exists a transition starting from this state and is labeled by an event in $\Sigma$. Therefore, we can conclude that $L(S)$ is a deadlock-free regular language and hence $\sup D(L(S)) = L(S)$.

Next, we apply the $L^*_C$ algorithm for the computation of $\sup C(\sup D(\overline{\mathcal{L}})) = \sup C(L(S))$. 
The first conjecture, denoted as $M^0 = M(S, E, T^0)$, that constructed by the $L^*_C$ algorithm is depicted in Fig. 3.9 (a).

![Figure 3.9. The conjecture supervisors $M^0$ and $M^1$ for $\text{sup } C(L(S))$.](image)

The $L^*_C$ algorithm then performs equivalence queries with respect to $M^0$, and a counterexample $acu \in L(S) - L(M^0)$ is detected. Since the string $acu$ is a common counterexample, we add all the strings in $\overline{acu}$ to construct an updated observation table, and the updated conjecture DFA is given in Fig. 3.9 (b), in which a string $abacuab$ is a counterexample. We hence update the observation table again and the corresponding conjecture DFA is given in Fig. 3.10, where no more counterexamples are returned by the $L^*_C$ algorithm.

![Figure 3.10. The conjecture DFA $M^1$ for $\text{sup } C(L(S))$.](image)

Finally, the maximally permissive closed-loop behavior for the controlled plant $G$ is given by $\text{sup } C(\mathcal{L}) = \lim \sup C(L(S))$. Therefore, the DBA representation of the desired
supervisor is exactly the same as Fig. 3.10, where the marked states shall be viewed as accepting states of a Büchi automaton. In this example, we can write that \( \sup C(\mathcal{L}) = (ab)^\omega + (ab)^*acua^\omega \).

It is worth pointing out at this point that the assumption that the specification \( \mathcal{L} \) is topologically closed in Problem 3.3 may become restrictive in some cases. We alter this assumption by assuming that \( \mathcal{L} \) is the limit language of a regular language; that is, there exists a regular language \( L \subseteq \Sigma^* \) such that \( \mathcal{L} = \lim L \). Under this assumption, we can also obtain a closed form expression of \( \sup C(\mathcal{L}) \).

**Theorem 3.5.** [92] If there exists a regular language \( L \subseteq \Sigma^* \) such that \( \mathcal{L} = \lim L \), then

\[
\sup C(\mathcal{L}) = \lim \sup CD(\overline{L}). \tag{3.35}
\]

When the condition of Theorem 3.5 is satisfied, we can still apply the \( L^\omega \) synthesis procedure for solving Problem 3.3, except for replacing \( \mathcal{L} \) by \( \overline{L} \).

3.4.4 Discussion: Learning-based Supervisor Synthesis under Partial Observation

In addition to the BSCP\( ^\omega \) problem, we are also interested in controlling a DES plant to fulfill \( \omega \)-regular specifications under partial observation. For such a purpose, we now formulate the counterpart of the SCPO problem for non-terminating specifications as the following SCPO\( ^\omega \) problem.

**Problem 3.4 (SCPO\( ^\omega \)).** Consider a plant \( G \) (3.23) that can be modeled by a DBA (3.23) over \( \Sigma \) and a topologically closed specification language \( \mathcal{L} \subseteq L(G) \). Let \( \Sigma_{uc} \subseteq \Sigma \) be the set of uncontrollable events, \( \Sigma_o \subseteq \Sigma \) be the set of observable events and \( P_\omega : \Sigma^\omega \to \Sigma_o^\infty \) be the associated natural projection (cf. Definition 3.9), synthesize a \( P_\omega \)-compatible \( \Sigma_{uc}\)-enabling and non-marking supervisor \( S_P \) such that \( \mathcal{L}(S\|G) \subseteq \mathcal{L} \).

\(^6\)The \( P_\omega \)-compatible supervisors can be defined similarly to the \( P_o \)-compatible supervisors for regular language specifications (cf. Definition 3.6).
Before proceeding to the solution of Problem 3.4, we need to make the following assumption.

**Assumption 3.2.** \( P_\omega(\mathcal{L}(G)) \subseteq \Sigma_\omega^o \); in other words, the plant \( G \) persistently generates observable events and cannot keep issuing \( \omega \)-strings that consist of only unobservable events.

Similar to the BSCP\( \omega \) problem, solving SCPO\( \omega \) can be reduced to the computation of the supremal topologically closed, \( \omega \)-controllable and \( \omega \)-normal sublanguage of the given \( \omega \)-regular specification \( \mathcal{L} \). It is shown that such language, which is denoted as \( \text{sup} \text{CN}(\mathcal{L}) \), indeed exists uniquely and a closed form expression for this sublanguage is given in the following theorem.

**Theorem 3.6.** [92] Let \( \mathcal{L} \subseteq \mathcal{L}(G) \) be topologically closed, then

\[
\text{sup} \text{CN}(\mathcal{L}) = \lim \text{sup} \text{CND}(\overline{\mathcal{L}}),
\]

where \( \text{sup} \text{CND}(\cdot) \) is the operator for computing the supremal controllable, normal and deadlock-free sublanguage.

Moreover, if we assume that \( \mathcal{L} \) is the limit of a regular language \( L \subseteq \Sigma^* \) rather than a topologically closed language, we have the following conclusion.

**Theorem 3.7.** [92] If \( \mathcal{L} = \lim L \), where \( L \in \Sigma^* \), then \( \text{sup} \text{CN}(\mathcal{L}) = \lim \text{sup} \text{CND}(\overline{\mathcal{L}}) \).

Since the computation of deadlock-free, normal and controllable sublanguages can be proceeded in a modular manner [121], we can then apply the \( L_D^* \) and the \( L_C^* \) algorithms for the computation of \( \text{sup} \text{CN}(\mathcal{L}) \) of the given specification \( \mathcal{L} \). The details are omitted here.

### 3.5 Conclusion

This chapter investigates the supervisor synthesis problems for both finite and infinite behavior specifications. To this regard, a series of modifications of the \( L^* \) algorithm is
proposed so that they can be adapted to various system architectures, including control under both complete observation and partial observation. The correctness, permissiveness, and finite convergence of the modified algorithms are illustrated. Furthermore, examples are presented to validate the effectiveness of the proposed algorithms.
CHAPTER 4

FORMAL MISSION AND MOTION PLANNING OF COOPERATIVE
MULTI-AGENT SYSTEMS VIA AUTOMATA LEARNING

4.1 Introduction

In Chapter 3, we propose a series of learning-based supervisor synthesis algorithms in order to design appropriate logical control and decision-making policies for an ACPS that can be modeled as a finite automaton. In this chapter, we go beyond the centralized system and study the application of the proposed synthesis algorithms for ACPSs with distributed architectures. In particular, we aim at integrating the proposed supervisor synthesis algorithms with formal verification techniques to solve the formal mission and motion planning problems of cooperative multi-agent systems, in which by “cooperative” we mean that the system of interest consists of a number of heterogeneous agents collaborating via physical interaction and wireless communication in a shared environment so as to accomplish various performance objectives. Representing a typical class of distributed ACPSs, cooperative multi-agent systems admit higher flexibility to various specifications and adaptiveness to environmental uncertainties, and hence outperform a monolithic ACPS that is capable of accomplishing multiple missions. Over the last two decades, cooperative multi-agent systems have emerged rapidly as a hot research topic in applications ranging from intelligent transportation networks, flexible manufacturing systems, smart grids to networked mobile robots in recent years, see e.g. [22, 28, 88, 123] and the references therein.

One of the essential issues in the design of cooperative multi-agent systems is how to design individual control policies and inter-agent coordination strategies such that the
desired performance can be attained. Although it is well-known that sophisticated collective behaviors could emerge from a group of agents via simple coordination rules, how to guarantee the fulfillment of complex coordination objectives in the presence of possible environment uncertainties still remains elusive. We are therefore motivated to derive a scalable and provably correct approach to assure the accomplishment of high-level missions for a team of cooperative agents in this chapter. More specifically, we consider a team of heterogeneous agents collaborating in a shared environment with potential uncertainties, based on which the formal mission and motion planning problem is investigated from a DES perspective. We assume that sequential executions of local missions of each agent are modeled by a local mission automaton, whereas a local motion automaton is associated with the agent to characterize its motion capabilities within the environment. From the agent-planning viewpoint, the coordination among the agents can be captured by the parallel composition of all the agents’ mission automata [18, 133, 163], and an appropriate mission plan for each agent can be carried out by means of supervisory synthesis techniques. A formal synthesis approach is developed to coordinate the cooperative multi-agent systems to accomplish the complex team missions in the presence of uncertain environments via divide-and-conquer. First, the global mission is decomposed into local missions for individual agents. Next, a provably-correct mission supervisor that fulfills the local mission is synthesized. Afterwards, a learning-based coordination scheme is proposed to justify whether or not the collective behaviors of all agents satisfy the global mission. Finally, a motion plan corresponding to the local missions of each agent is computed based on a nominal environment model. In case the motion plans fail to be feasible due to the uncertainties of the real environment, a counterexample-guided motion re-planning procedure is employed so that the accomplishment of the global mission will still be assured. The contributions of this chapter can be summarized as:

(i) The proposed formal planning framework differs from the previous works on multi-agent planning (see, e.g., [28, 80, 88]) in the sense that we account for the local controllability of certain mission events. Towards this end, a local mission plan is
realized by applying a local mission supervisor for the agent. In particular, a modified $L^*$ learning algorithm is developed to synthesize a permissive mission supervisor via inductive learning techniques [76].

(ii) The formal planning framework developed in this chapter is also different from the coordination planning of multi-agent systems [133, 163], in which the event sets associated with each agent were pairwise disjoint and the agents were coordinated through “coordination modules”. Instead, we adopt a counterexample-guided assume-guarantee approach to coordinate the mission executions of all the agents [36, 132] under local mission supervisors, and a modification of the $L^*$ learning algorithm is presented to automatically generate the appropriate assumptions for individual agents.

(iii) In addition to our previous work on mission planning of cooperative multi-agent systems [41, 43], we compute an appropriate motion plan corresponding to each agent’s mission plan by applying a second modification of the $L^*$ algorithm for the prior knowledge of the environment. Moreover, when a motion plan fails to be feasible due to the uncertainties of the real environment, a counterexample-guided motion re-planning procedure is employed so as to accomplish the global mission.

The remainder of this chapter is organized as follows. A brief review of prior works on formal planning of multi-agent/robotic systems is presented in Section 4.2. We formulate the formal mission and motion planning problems for cooperative multi-agent systems for a declarative regular language specification in Section 4.3. In Section 4.4, we propose an automatic synthesis framework to synthesize appropriate local missions and motion plans for each agent in the presence of environment uncertainties, along with an illustrative example in multi-robot coordination. In order to accomplish a successful multi-agent mission planning, the $L^*_c$ synthesis algorithm is first applied in Section 4.5 to synthesize local mission supervisor synthesis, followed by another modification of the $L^*$ algorithm that is developed to achieve the assume-guarantee coordination among the agents afterwards. The prior knowledge of the nominal environment is incorporated with the obtained mission plans in Section 4.6 in order to generate feasible and correct motion plans for individual agents; furthermore, a counterexample-guided algorithm is also proposed to re-compute the motion plans in case that the original ones fail to be feasible in the real environment. We conclude this chapter in Section 4.7.
4.2 Related Work

In this section, we briefly review the prior works on the formal motion and/or mission planning problems of both single and multiple intelligent agents, along with their applications on autonomous robotic systems.

4.2.1 Formal Planning of Autonomous Robots

Intelligent mobile robots with localization, sensing and motion capabilities represent a typical class of ACPSs. The motion planning problem for mobile robots has received a considerable amount of research interest in recent years. Informally speaking, given a mobile robot with its dynamics, a description of the environment, an initial configuration of the robot, and a set of goal configurations, the motion planning problem is solved by obtaining a sequence of control inputs that steers the robot to move from the initial configuration to the goal configurations while avoiding the collision with the surrounding obstacles in the environment [31, 103]. In addition to the classical motion planning problems that aim at achieving a reach-avoid maneuver, a growing number of diverse robotic applications such as mobile navigation, emergency response, autonomous manipulation and surgical operations require both continuous motions and discrete behaviors, which makes integrated mission and motion planning of robots an increasingly important problem. On the one hand, the motion planner should solve the motion planning problem at the physical level by generating dynamically-feasible motion trajectories that can be tracked by the robot. The dynamics of the robot express physical constraints on the feasibility of motions, such as bound of the velocity of motions, bound on the turning radius, and keeping the wheels from sliding sideways [79, 81, 105]. Moreover, the presence of obstacles makes motion planning more challenging since the planned motions must also be collision-free, which requires the robot to pass through narrow spaces and/or unstructured terrains [184]. On the other hand, research challenges also arise in the cyber level, whose design objective
focuses on computing coarse, which is typically discrete, control actions that are executed sequentially by the robot [10, 135].

Several methodologies have been developed to solve the formal planning problem of robotic systems. By utilizing a behavior-based approach [5, 181], the mission and/or motion planning objectives can be accomplished by sequentially composing pre-defined behaviors. Despite the capability of adapting to various team tasks, the behavior-based design method show empirical features, resulting in a trial-and-error design process and therefore lacking guarantees of the performance. Although recent studies in behavior-based planning [114] have taken guarantee of performance into consideration, it mainly focuses on the mission planning problem of one single agent and motion planning problem was not discussed therein.

The desire to achieve complex planning objectives leads to the research interest in formal specifications such as regular languages, linear temporal logic (LTL) and/or computation tree logic (CTL) formulas [133, 134, 136, 169], which admit powerful expressiveness of safety, liveness and fairness properties. A recent trend of “symbolic motion planning” has enabled the model-based formal integration of high-level discrete mission executions with low-level motion controllers in a unified framework so that accomplishment of complex specifications can be guaranteed [4, 10]. The standard procedure of solving the symbolic motion planning problem states as follows: first, a discrete model is obtained to abstract the continuous dynamics of the robot, is obtained accordingly [138]; secondly, discrete control and decision-making processes are carried out by leveraging ideas from formal verification [7, 35, 182], reactive synthesis [95, 137] and supervisory control [26, 91, 143] techniques; finally, the obtained sequences of symbolic motion actions that fulfill the specifications are used to generate the desired motion trajectories and the feedback controllers in the physical level so that the robot is able to precisely track them. In [58], the motion planning problem for mobile robots with second order dynamics was addressed in order to achieve LTL specifications. A unified framework was proposed
in [89] to automatically translate high-level tasks in the form of a specific class of LTL formulas into correct-by-construction hybrid controllers; the results therein were further extended in [90] in such a way that task planning was also considered in the presence of the environment. However, the aforementioned earlier work of robotic motion planning usually assumed the external environment to be known and static. Recently, methodologies have also been developed to address the uncertainties of the environment. In [29], the authors designed a control policy to achieve a surveillance mission by combining formal synthesis methods with automata learning algorithms of the uncertain environments, whereas Lahijanian et al. [100] employed a multilayered synergistic planner to generate trajectories that partially satisfy an LTL specification in uncertain environments. On the other hand, by introducing stochastic features into the abstract model for the robotic motion planning problem, the uncertainty arising in the environments was dealt with in [99, 115] by using stochastic model checking techniques [99] so that a partial satisfaction of the specification can be achieved. The integrated mission and motion planning of robotic systems was also considered in [40] where the integrated mission and motion plans can be computed as a composition of different control primitives. Nevertheless, it is worth noticing that the assumption embedded in the symbolic planning approach that a discrete abstraction of the robot’s behaviors (motion and actions) can be obtained is non-trivial to be satisfied when complex nonlinear dynamics are considered.

4.2.2 Performance-guaranteed Coordination of Multi-agent Systems

Multi-agent systems also represent a typical class of ACPSs, since each agent directly interacts with the physical environments with non-trivial continuous dynamics and physical constraints, while the decision-making process of each agent and the communication network between agents jointly form the cyber part. Furthermore, the perception and situation-awareness functionalities of each agent bridge the gap between the cyber and the physical parts. Compared to a centralized monolithic system that is capable of executing
multiple missions, a team of intelligent agents offer greater performance due to high flexibility to various specifications, ability to re-configuration in the presence of environmental uncertainties and reliability against possible loss of functionalities. Due to their wide applications, ranging from power grids, transportation networks, manufacturing systems to robotic teams, multi-agent systems have emerged rapidly as a hot research topic in the context of ACPSs and numerous distributed coordination and control problems have been extensively studied in recent years, see, e.g., [9, 22, 61, 88, 119, 123–125, 145, 158, 180, 200] and the references therein.

The past two decades have witnessed the development of performance-guaranteed coordination and control schemes of multi-agent systems, and many contributions have been made to address various coordination purposes, such as consensus [125], flocking [124, 200], rendezvous [53], formation control [52, 60, 145], collision avoidance [127] and cooperative learning [97] of multi-agent systems. The achievement of the global coordination goals in the above-cited works is assured by means of stabilization techniques, such as Lyapunov stability [124, 125], barrier certificates [127] and game theory [116, 162]. Despite fulfillment of steady-state performance objectives, satisfaction of more complicated and temporal specifications has not been investigated in these previous works.

Despite the success for a single agent to attain formal mission and motion specifications, planning of multi-agent case are computationally non-trivial. Filippidis et al. [62] proposed a decentralized control scheme to satisfy local LTL specifications with communication constraints among the agents; and this idea was developed by Guo and Dimarogonas [69] to derive a partially decentralized coordination scheme that decomposed the team into clusters of dependent agents to fulfill local LTL specifications. To cope with computational complexity issues, the results therein were further extended by Tumova et al. [178] by employing a receding horizon planning approach [188, 189]. Nevertheless, specifying each agent’s individual task is difficult to scale up in practice and usually becomes burdensome for human operators in application.
On the other hand, given a global LTL specification, Kloetzer and Belta [88] derived a monolithic solution by assigning individual tasks to each agent accordingly. Nonetheless, deployment of such approaches often suffered from the “state explosion” due to their monolithic design pattern. The results were further extended in [179] where robustness and optimality of the synthesized plans were taken into consideration. “Trace-closed” regular language specifications were studied in [28] for automatic deployment of robotic teams. To pursue a top-down coordination architecture, Karimadini and Lin [83] presented necessary and sufficient conditions under which the global tasks can be retrieved by local ones, while Partovi and Lin [131] investigated this problem for CTL specifications. Brandin et al. [18] studied the coordinated planning of multi-agent systems by using an incremental verification approach.

In addition to the aforementioned multi-agent planning approaches that were based on formal verification techniques, we also get inspirations from the decentralized/distributed supervisory control theory of discrete event systems. Willner and Heymann [187] extended the study of decentralized supervisory control theory [111] and investigated the supervisory control of concurrent discrete-event systems by proposing a necessary and sufficient condition for the existence of the decentralized supervisor based on the separability of the specification, and their results were extended in [77] by relaxing the assumptions upon the controllability status of the shared events. This problem was also considered in [64] in a different way and a centralized solution was performed on approximations of the plant by using the notion of mutual controllability. Coordination and control theory of distributed discrete event systems were also applied for multi-agent systems via a supervisory control approach. The multi-agent coordination problem was tackled in [133, 163], in which all the agents were assumed to possess only local events and were coordinated through “coordination modules”. To deal with the constraints on inter-agent communication, Kantaros and Zavlanos [80] developed distributed controllers for a team of mobile robots that determined meeting times at the nodes of the mobility graph so that connectivity of the
communication network can be ensured over time, infinitely often.

It is worth pointing out that the prior work mainly focused on task allocation and/or design of control policies among multiple agents with known external environment, while in our work \([41, 42]\), the formal planning and control of multi-agent systems in the presence of both system and environment uncertainties was investigated. Moreover, we consider the case in which the global mission is given in a declarative manner, and the mission supervisor is synthesized via inductive learning techniques \([76]\). For such a purpose, a series of modified \(L^*\) learning algorithms \([2]\) are developed.

4.3 Problem Formulation

The basic setup of the formal mission and motion planning problems of cooperative multi-agent systems is presented in this section. We first introduce the automata models of the cooperative multi-agent system under consideration, followed by the statement of the formal planning problem.

4.3.1 Formal Models of Cooperative Multi-agent Systems

The cooperative multi-agent system under consideration is composed of \(n\) heterogeneous agents moving in a shared environment. From an ACPS point of view, a formal model of the cyber part of an individual agent should provide description of the sequential executions of local missions of the agent. For such a pursuit, we model the mission capabilities of the \(i\)-th agent \((i \in I)\) as the following mission DFA

\[
G_i = (Q_i, \Sigma_i, \delta_i, q_i, 0),
\]

where \(\Sigma_i\) is a set of mission events, or actions that are to be performed by the agent within an environment. A transition \(q_i' = \delta_i(\sigma_i, q_i)\) is defined if the agent \(G_i\) starts a mission \(\sigma_i \in \Sigma_i\) at state \(q_i\) and finishes it at state \(q_i'\). For the application of SCT, the set of local
mission events $\Sigma_i$ is partitioned into local controllable missions $\Sigma_{i,c}$ and local uncontrollable missions $\Sigma_{i,uc}$. In practice, the controllable mission may refer to as certain control commands over the actuators of the agent, whereas uncontrollable missions may include reception of either sensor readings or information from other agents. Without loss of generality, we assume that all the mission events in $\Sigma_i$ are (locally) observable, that is, $G_i$ can monitor the occurrences of any local missions. The sets of the global missions, globally controllable and uncontrollable missions are given by $\Sigma = \bigcup_{i \in I} \Sigma_i$, $\Sigma_c = \bigcup_{i \in I} \Sigma_{i,c}$ and $\Sigma_{uc} = \Sigma - \Sigma_c$, respectively. For each mission $\sigma \in \Sigma$, let $I_\sigma = \{i \in I|\sigma \in \Sigma_i\}$; in other words, $|I_\sigma| = 1$ suggests that $\sigma$ be executed by one single agent, while $|I_\sigma| > 1$ indicates that all the agents in $I_\sigma$ should coordinate to accomplish the shared mission $\sigma$ synchronously. From the multi-agent coordination point of view, it is reasonable to assume that all the shared missions should agree on the local controllability status, i.e.,

\[(\forall i, j \in I)((i \neq j) \Rightarrow (\Sigma_{i,uc} \cap \Sigma_{j,c} = \emptyset))]. \tag{4.2}\]

The collective executions of all the missions of the cooperative multi-agent system can be modeled as $G = \|_{i \in I} G_i$.

It is also necessary to provide a formal description of the environment in which all the agents perform their mission executions. In this chapter, the environment is assumed a priori to be partitioned into $N$ regions, and two adjacent regions may be connected via one or more doors. This assumption is reasonable since various partitioning methods [58, 87] can be utilized to obtain a finite abstraction of the environment. We assume that this nominal environment is modeled as the following graph:

\[E_0 = (V, D, \rightarrow_{E_0}), \tag{4.3}\]

where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of the partitioned regions with unique identities, $D = \{d_1, d_2, \ldots, d_{N_d}\}$ with $N_d \in \mathbb{N}$ is the collection of all the doors; and the adjacency
relation $\rightarrow_{E_0}$ satisfies that $\rightarrow_{E_0} \subseteq V \times (D \cup \{\epsilon\}) \times V$, i.e., for $v, v' \in V$ and $d \in D$, $(v, d, v') \in \rightarrow_{E_0}$ if and only if an agent may move from $v$ to $v'$ via the door $d$. We assume that $\rightarrow_{E_0}$ is reflexive and $(v, \epsilon, v) \in \rightarrow_{E_0}$ holds for all $v \in V$. Note that the environment graph $E_0$ is directed, i.e., there exist one-way doors in the environment that leads to the adjacency relation $\rightarrow_{E_0}$ to be asymmetric.

All the agents are assumed to be aware of the prior knowledge of the nominal environment $E_0$. Therefore, the motion capabilities of the agent $G_i$ ($i \in I$) within $E_0$ can be extracted as the following motion DFA:

$$G^m_i = (V, D, \delta^m_i, v_{i,0}),$$

(4.4)

where $v_{i,0} \in V$ is the initial region of the $i$-th agent and the motion transition $\delta^m_i$ is defined such that $(\forall v \in V)(\forall d \in D)[\delta^m_i(v, d) = v' \iff (v, d, v') \in \rightarrow_{E_0}]$, i.e., an agent can move from the current region to an adjacent region as long as there is a door connecting them.

4.3.2 Formal Mission and Motion Planning Problem of Cooperative Multi-agent Systems

We now integrate an individual agent’s capacities to service various missions with its motion capabilities within the environment by a mapping

$$\pi_i : \Sigma_i \cup \{\epsilon\} \rightarrow 2^V$$

(4.5)

defined that specifies a set of regions in which the agent $G_i$ ($i \in I$) can perform a mission. In particular, we assume that $\pi_i(\epsilon) = V$, i.e., an agent can pass through a region without executing any mission. Note that $\pi_i$ ($i \in I$) is only related to the regions in the environment and is independent of the adjacency relationship $\rightarrow_{E_0}$ of the environment graph $E_0$.

An integrated mission-motion plan for the agent $G_i$ can then be defined thanks to $\pi_i$.

**Definition 4.1 (Integrated Plans).** An integrated mission-motion plan for the $i$-th agent with
mission DFA $G_i$ and motion DFA $G_{i,m}^m$ ($i \in I$) is a string $LP_i \in [V \cup (\Sigma_i \cup \{\epsilon\})]^*$ which satisfies the following properties.

(i) The agent must stay in its initial region, i.e., $LP_i(0) = v_{i,0}$ for all $i \in I$.

(ii) The motion of an agent should be restricted by the transition relation of the environment. In other words, for any $i \in I$, $P_V(LP_i) \subseteq \text{Run}[L(G_{i,m}^m)]$, where $\text{Run}[L(G_{i,m}^m)] = \bigcup_{s \in L(G_{i,m}^m)} \text{Run}(s)$ is the collection of all runs of $G_{i,m}^m$ and $P_V$ represents the natural projection from $[V \cup (\Sigma_i \cup \{\epsilon\})]^*$ to $V^*$.

(iii) An agent may either pass through a region without servicing any mission request or execute a mission only in a region that it can be executed. Formally, if $LP_i(k - 1), LP_i(k) \in V$, then there exists a door $d \in D$ such that $LP_i(k) = \delta^m_i(LP_i(k - 1), d)$; if $LP_i(k) \in \Sigma_i$ and $LP_i(k - 1) \in V$, then $LP_i(k - 1) \in \pi_i(LP_i(k))$; otherwise, if $LP_i(k), LP_i(k + 1) \in \Sigma_i$ and $LP_i(k - 1) \in V$, then $LP_i(k - 1) \in \pi_i(LP_i(k) \cap \pi_i(LP_i(k + 1)))$.

An integrated plan $LP_i$ for the agent $G_i$ ($i \in I$) determines a motion plan $L_{i,m}^m = P_V(LP_i)$ and a mission plan $L_{i,m}^m = P_{\Sigma_i}(LP_i)$, where $P_{\Sigma_i}$ is the natural projection from $[V \cup (\Sigma_i \cup \{\epsilon\})]^*$ to $\Sigma_i^*$. In addition to Definition 4.1, a shared mission $LP_i(k)$ for some $k \in \mathbb{N}$ (i.e., $|I_{LP_i(k)}| > 1$) should be executed via a wait-and-coordinate protocol: the agent $G_i$ should broadcast the mission request $LP_i(k)$ to all the other agents in $I_{LP_i(k)}$ in region $LP_i(k - 1)$. When all the other agents move to the designated region for $\pi_j(LP_i(k))$, $j \in I_{LP_i(k)} - \{i\}$, the mission can be accomplished jointly. Therefore, the coordination of the missions accomplished by all the agents is given by $\|i \in I LP_i$.

In this chapter, we are interested in the successful mission and motion planning of cooperative multi-agent systems in the presence of environment uncertainty. More specifically, the practical environment $\mathcal{E} := (V, D, \rightarrow_{\mathcal{E}})$ in which the agents perform their mission plans shares the same set $V$ of regions with the nominal one $\mathcal{E}_0$; nonetheless, some doors in $D$ might be blocked in $\mathcal{E}$, rendering the adjacency relationship $\rightarrow_{\mathcal{E}}$ to be different from $\rightarrow_{\mathcal{E}_0}$. As a consequence, an integrated plan $LP_i$ developed based on $\mathcal{E}_0$ may fail to be realizable in $\mathcal{E}$.

By defining a global mission as a prefix-closed regular language over $\Sigma$, our main objective is to obtain an integrated plan for each agent such such that the mission can be
fulfilled in the presence of environment uncertainties.

**Problem 4.1.** Consider a cooperative multi-agent system that consists of \(n\) agents with the nominal environment \(E_0\). The \(i\)-th agent is associated with a mission DFA \(G^N_i\) and a motion DFA \(G^m_i\) \((i \in I)\) with respect to \(E_0\). Given a prefix-closed global mission \(L = \overline{L} \subseteq \Sigma^*\), compute an integrated mission and motion plan \(LP_i\) \((i \in I)\) for each agent such that \(\| \bigwedge_{i \in I} LP_i \models L\) holds in both the nominal environment \(E_0\) and the real environment \(E\).

**Remark 4.1.** In practice, it is reasonable to request that all the agents jointly accomplish the global mission in a "step-by-step" manner; thus, we omit the marked states for both \(G_i\) and \(G^m_i\) and the global mission \(L\) is assumed to be prefix-closed. The synthesis of non-blocking mission and motion plans will be investigated in the future.

4.4 Overview of the Approach

In this section, we present our approach to solve Problem 5.1, along with a multi-robot coordination example to illustrate the proposed planning framework.

4.4.1 The Formal Mission and Motion Planning Framework

We now propose a two-layer formal mission and motion planning framework to solve Problem 4.1, whose working procedure is depicted in Fig. 4.1. Since in general \(\Sigma \cap D = \emptyset\), therefore \(\| \bigwedge_{i \in I} LP_i \models L\) is equivalent to \(\| \bigwedge_{i \in I} L^m_i \models L\), which hence enables us to solve Problem 5.1 in a “modular” manner. Our approach to solve Problem 5.1 is to “divide-and-conquer”, and can be summarized as the following steps.

(i) **Mission decomposition:** We first generate a prefix-closed and feasible local mission \(L_i \subseteq \Sigma^*_i\) for the agent \(G_i\) \((i \in I)\), which captures all the missions that should be implemented by \(G_i\) in order to accomplish the global mission.

(ii) **Mission supervisor synthesis:** Facing a local mission specification \(L_i\), we synthesize a local mission supervisor \(S_i\) for the agent \(G_i\) \((i \in I)\) based on \(\Sigma_{i,c}\) and \(\Sigma_{i,uc}\). A local mission plan \(L^m_i\) corresponding to \(L_i\) can then be obtained by setting \(L^m_i = L(S_i|G_i) \subseteq L_i\).
(iii) **Assume-guarantee coordination:** After obtaining the local mission plans, we employ a compositional verification procedure to justify whether or not the collective executions of local missions of each agent can fulfill the global mission jointly. When the joint execution of the mission plans fails to satisfy \( L \), a counterexample \( t \subseteq \Sigma^* \) is generated by the compositional verification that indicates the violation of \( L \). We update the local mission by taking advantage of \( L_{mi}^i \) and \( t \). Afterwards, we return to step (i) and update the local mission supervisors until no more counterexamples are generated. Finally, local mission plans \( L_{mi}^i (i \in I) \) can be obtained.

(iv) **Automatic motion planning:** We then generate a motion plan \( L_{mo}^i \) for the agent \( G_i (i \in I) \) by incorporating the motion DFA \( G_{mi}^i (i \in I) \) in the nominal environment \( E_0 \) with the obtained mission plan \( L_{mi}^i \) so that the combination of \( L_{mi}^i \) and \( L_{mo}^i \) forms a valid integrated local plan \( LP_i \) for \( G_i \).

(v) **Counterexample-guided motion re-planning:** In case that \( LP_i (i \in I) \) cannot be implemented in the practical environment \( E \) due to uncertainties, we apply a counterexample-guided re-planning procedure to automatically refine the motion plan \( L_{mo}^i \) such that a feasible integrated plan can be formed.
It is also worth pointing out that automata learning algorithms are also involved in the framework shown in Fig. 4.1 in order to not only efficiently generate satisfactory mission and motion plans for the individual agents based on the nominal environment, but to assure successful multi-agent planning in the presence of environment uncertainties as well.

4.4.2 A Motivating Example of Multi-Robot Coordination

Throughout the rest of this chapter, we consider a multi-robot coordination and planning problem as an illustrative example of the formal mission and motion planning framework shown in Fig. 4.1. As depicted in Fig. 4.2 (a), the cooperative multi-robot system is assumed to be composed of three robots, whose mission DFAs are given by $G_1$, $G_2$, and $G_3$, respectively. All the robots are supposed to have identical localization sensors and communication devices. Furthermore, $G_2$ is the only robot that is equipped with a fire-extinguisher and is capable of responding to a fire alarm.

![Multi-robot coordination scenario](image1)

(b) The nominal environment $E_0$.

Figure 4.2. The multi-robot coordination example.

The environment of the example is shown in Fig. 4.2 (b), where all the doors connecting the rooms are automatically closed whenever there is no external force to keep them open.
In the nominal case, Room 2 and Room 1 are connected by the one-way door $D_2$ and the
two-way door $D_1$, while Room 3 and Room 1 are connected by $D_1$ and another two-way
door $D_3$. From (4.3), all the elements in the nominal environment $E_0 = (V, D, \rightarrow_{E_0})$ are
given by

- $V = \{R_1, R_2, R_3\}$, where $R_j$ stands for Room $j$ ($j = 1, 2, 3$);
- $D = \{D^l_1, D^r_1, D_2, D_3\}$, where $D^l_1$ and $D^r_1$ represent the left and right half of $D_1$, respectively;
- The adjacency relation $\rightarrow_{E_0} = \{(R_1, D_2, R_2), (R_1, D^r_1, R_2), (R_1, D^l_1, R_3), (R_1, D_3, R_3), (R_2, D^r_1, R_1), (R_3, D_3, R_1), (R_3, D^l_1, R_1)\}$;

For each $i \in I := \{1, 2, 3\}$, the motion DFA $G^m_i = (V, D, \delta^m_i, v^i_0)$ for the $i$-th robot
can be extracted from $E_0$ according to (4.4): $v^i_0 = R_1$, $\delta^m_i(R_1, D^r_1) = \delta^m_i(R_1, D_2) = R_2$, $\delta^m_i(R_1, D_3) = \delta^m_i(R_1, D^l_1) = R_3$, $\delta^m_i(R_3, D_3) = \delta^m_i(R_3, D^l_1) = R_1$. A DFA representation
of $G^m_i$ is depicted in Fig. 4.3

![DFA Diagram]

Figure 4.3. The motion DFA $G^m_i$ for the robot $G_i$ ($i = 1, 2, 3$).

On the other hand, the mission events of the cooperative multi-robot system are listed
in Table 4.1. The local event set $\Sigma_i$ ($i \in I$) associated with $G_i$ is defined as follows,
respectively:

\[ \Sigma_1 = \{h_1, \text{Open}, \text{Close}, G_2 \in R_1, G_1 \in R_3, D_1 \text{close}, D_1 \text{open}, G_1 \in R_1, R\} , \]

\[ \Sigma_2 = \{h_2, F, D_1 \text{open}, G_2 \in R_1, G_2 \in R_2, R\} , \]

\[ \Sigma_3 = \{h_3, G_3 \in R_3, \text{Open}, \text{Close}, D_1 \text{open}, G_2 \in R_1, D_1 \text{close}, G_3 \in R_1, R\} . \]

We assume that the \( i \)-th robot has the full knowledge of the events in \( \Sigma_i \ (i \in I) \) for the purpose of local mission planning. Note that in this example, it is reasonable to assume \( G_2 \in R_1 \in \Sigma_1, G_2 \in R_1 \in \Sigma_3 \) and \( D_1 \text{open} \in \Sigma_2 \) since they can all be viewed as the information that is transmitted among \( G_1, G_2 \) and \( G_3 \) via synchronous communication.

### TABLE 4.1

<table>
<thead>
<tr>
<th>Event</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_i )</td>
<td>Robot ( G_i ) ((i = 1, 2, 3) ) receives the mission request.</td>
</tr>
<tr>
<td>( F )</td>
<td>Robot ( G_2 ) extinguishes the fire.</td>
</tr>
<tr>
<td>( G_i \text{to} D_1 )</td>
<td>Robot ( G_i ) approaches the door ( D_1 ), ( i = 1, 3 ).</td>
</tr>
<tr>
<td>( G_i \text{on} D_1 )</td>
<td>Robot ( G_i ) ((i = 1, 3) ) localizes itself at the door ( D_1 ).</td>
</tr>
<tr>
<td>( G_i \in R_k )</td>
<td>Robot ( G_i ) stays at Room ( R_k ) ((i, k = 1, 2, 3) ).</td>
</tr>
<tr>
<td>( Open )</td>
<td>Control command for moving forward to open ( D_1 ).</td>
</tr>
<tr>
<td>( Close )</td>
<td>Control command for moving backward to close ( D_1 ).</td>
</tr>
<tr>
<td>( D_1 \text{open} )</td>
<td>The status that ( D_1 ) is opened by robots.</td>
</tr>
<tr>
<td>( D_1 \text{closed} )</td>
<td>The status that ( D_1 ) is closed by ( G_1 ) and ( G_3 ).</td>
</tr>
<tr>
<td>( R )</td>
<td>The robot returns to Room ( R_1 ) and waits for the next mission.</td>
</tr>
</tbody>
</table>
The global mission assigned to this multi-robot team requests that the robot $G_2$ enter $R_2$ in response to a fire-extinguishing alarm from $R_2$; meanwhile, $G_1$ and $G_3$ need to open $D_1$ jointly so that $G_2$ can successfully return. After $G_2$ returns from $R_2$, $G_1$ and $G_3$ should close $D_1$ and return to Room 1 to wait for possible alarms in the future. We use the following two prefix-closed languages $L_{1}^{spe}$ and $L_{2}^{spe}$ to capture the two requirements for the multi-robot system, respectively,

\[
L_{1}^{spe} = (h_2 FG_2 in R_1 R)^*,
\]

\[
L_{2}^{spe} = ((h_1 h_3 + h_3 h_1) Open D_1 open G_2 in R_1 Close D_1 close R)^*.
\]

The overall global mission specification is hence given by $L = L_{1}^{spe} \parallel L_{2}^{spe}$.

The mission DFA $G_i$ ($i \in I$) provides a formal description of all the possible executions of local missions of the $i$-th robot. In this example, we are interested in the possible fire-extinguishing request from Room 2 and thus the mission DFA $G_2$ shall characterize robot $G_2$’s mission executions between Room 1 and Room 2 in the nominal environment. In fact, robot $G_2$ can enter $R_2$ from $R_1$ via either $D_1$ or $D_2$, provided that $D_1$ has already been opened by the other two robots, while $G_2$ can only return to $R_1$ via $D_1$ since $D_2$ is a one-way door and $(R_2, D_2, R_1) \notin \rightarrow \epsilon_0$. Thus, the mission DFA $G_2$ can be constructed accordingly, as depicted in Fig. 4.4.

On the other hand, the mission DFAs $G_1$ and $G_3$ describe the mission executions of the other two robots within Room 1 and Room 3 in this example. For such a pursuit, the DFA models that characterizes the behaviors of the robot $G_i$ ($i = 1, 3$) are shown in Fig. 4.5. Furthermore, to synthesize a local mission supervisor for an individual robot, the sets of locally uncontrollable events for each robot are given by $\Sigma_{1,uc} = \{h_1\}$, $\Sigma_{2,uc} = \{h_2\}$ and $\Sigma_{3,uc} = \{h_3\}$, respectively, which is consistent with our assumption (4.2).

In the sequel, we aim to appropriate integrated plans $LP_i = (L_{1i}^{mi}, L_{1i}^{mo})$ for each robot with respect to $G_i$ and $G_i^{m}$ ($i \in I$) such that $\parallel_{i=1}^{3} LP_i \models L$ holds in both nominal and practical environments.
4.5 Counterexample-guided Mission Planning of Cooperative Multi-agent Systems

This section concerns with the formal mission planning of the cooperative multi-agent system. For such a pursuit, we propose a counterexample-guided method to synthesize a series of local mission supervisors for individual agents such that the global mission can be fulfilled. As depicted in Fig. 4.1, the mission planning procedure involves the iterative execution of three steps: (i) a prefix-closed local mission is assigned to the agent $G_i$ ($i \in I$) based on $G_i$ and $L$; (ii) a corresponding mission supervisor $S_i$ is synthesized for $G_i$ so that
is satisfied; (iii) an assume-guarantee coordination scheme is developed such that the collective execution of all the local missions will guarantee the accomplishment of the global mission \( L \).

4.5.1 Learning-based Synthesis of Local Mission Supervisors

We start the formal mission planning with the synthesis of feasible local missions for each agent. In fact, the synthesis of a locally feasible mission plan \( L_{mi}^i \) for the agent \( G_i \) \((i \in I)\) consists of two steps: (i) we assign the agent \( G_i \) with an appropriate local mission \( L_i \) in terms of \( G_i \) and \( L \); (ii) we exploit the controllability status of the local mission events \( \Sigma_{i,uc} \) to develop feasible local missions and an associated mission supervisor is synthesized to steer the agent to accomplish the mission.

Initially, the local mission \( L_{mi}^i \) \((i \in I)\) is assigned to the agent \( G_i \) \((i \in I)\) via the following procedure:

\[
L_i = P_i(L),
\]
\[
L_{mi}^i = \{ t \in L(G_i) | \text{DFA}(t) \models L_i \}.
\]

The first equation of (4.6) guarantees that the obtained language \( L_i \) from the global mission \( L \) shares the same set of mission events of the agent \( G_i \) \((i \in I)\), and the second equation assures that the initial local mission \( L_{mi}^i \) is a proper sublanguage of \( L(G_i) \). Since \( L \) is a prefix-closed language over \( \Sigma \) and prefix-closeness is preserved under natural projection, we can conclude that \( L_i \) is also prefix-closed and hence by construction, \( L_{mi}^i \) is a prefix-closed sublanguage of \( L(G_i) \).

Given the initialized local mission \( L_{mi}^i \) (4.6) for the agent \( G_i \) \((i \in I)\), the local mission supervisor \( S_i \) is synthesized to steer \( G_i \) to fulfill the local mission \( L_{mi}^i \) in a maximally permissive manner. Since \( L_{mi}^i \) is a non-empty and prefix-closed sublanguage of \( L(G_i) \), it is reasonable to apply the \( L^* \) algorithm for the synthesis of the mission supervisor \( S_i \) with respect to \( L_{mi}^i \) and \( \Sigma_{i,uc} \). In particular, by applying the \( L^* \) algorithm for the local mission
\[ L_i^{mi} \quad (i \in I), \] it can be written that

\[ S_i = L_C^*(G_i, \Sigma_{i,uc}, L_i^{mi}) = \sup C_i(L_i^{mi}) \subseteq L_i^{mi}. \quad (4.7) \]

Since the correctness and finite termination of the \( L_C^* \) algorithm are guaranteed by Theorem 3.1 and Theorem 3.2, respectively, the synthesis of \( S_i \) can be accomplished within a finite number of iterations.

**Remark 4.2.** It is also worth pointing out that, as suggested in Section 3.2.4, it is also possible to utilize the \( L_C^* \) algorithm when prior model knowledge of \( G_i \) is not available. Therefore, the mission supervisor synthesized in (4.7) can also be attained even if \( G_i \) is not fully given a priori.

We now revisit the multi-robot coordination example in Section 4.4.2 to see how the (initial) local mission supervisors can be computed.

**Example 4.1.** We follow the framework shown in Fig. 4.1 to synthesize (initial) local mission plans supervisors \( S_i \) for each robot \( G_i \) \((i \in I)\) in Section 4.4.2. The initial local missions \( L_i^{mi} \) for the robot \( G_i \) \((i \in I)\) can be computed according to (4.6), and the DFA representations of the local missions assigned to the three robots are shown in Fig. 4.6, Fig. 4.7 and Fig. 4.8, respectively.

![Diagram](image_url)

Figure 4.6. The initialized mission \( L_1^{mi} \) for the robot \( G_1 \).
Next, we apply the $L^*_C$ algorithm for the synthesis of the local mission supervisor $S_i$ for the $i$-th robot with respect to $G_i$, $\Sigma_{i,uc}$ and $L_{mi}^i$ ($i \in I$). Note that $\Sigma_{i,uc} = \{h_i\}$ for all $i \in I$, it is clear that $L_{mi}^i$ is a locally controllable sublanguage of $L(G_i)$ in the multi-robot example. A candidate mission supervisor $S_2$ is synthesized as shown in Fig. 4.9 to achieve $\sup C_2(L_{mi}^2)$ for the robot $G_2$, which is identical to $L_{mi}^2$. 
The intuition behind the mission supervisors depicted in Fig. 4.9 states that the robot $G_2$ must enter $R_2$ to respond to the fire alarm. In addition, local mission supervisors $S_1$ and $S_3$ can be developed in a similar manner for $G_1$ and $G_3$, as shown in Fig. 4.10 and Fig. 4.11, respectively. It turns out that $\sup C_1(L_{1}^{mi}) = L_{1}^{mi}$ and $\sup C_3(L_{3}^{mi}) = L_{3}^{mi}$.

Figure 4.9. The local mission supervisor $S_2$ for the robot $G_2$.

Figure 4.10. The local mission supervisor $S_1$ for the robot $G_1$.  

96
Figure 4.11. The local mission supervisor $S_3$ for the robot $G_3$.

According to Theorem 3.1, the $L^*_C$ algorithm always synthesizes a local mission supervisor in a maximally permissive manner; therefore, for $i = 1, 3$, the current local mission supervisors $S_i$ allows the robot $G_i$ to either stay in $R_1$ or enter $R_3$ so that the heavy door $D_1$ can be opened jointly.

4.5.2 The $L^*_CV$ Algorithm: Assume-guarantee Coordination of Multiple Agents

After deducing the local mission supervisors, it is necessary to determine whether or not the collective execution of the local missions of all the agents will imply the satisfaction of the global mission $L$. For such a purpose, we first present an assume-guarantee framework to efficiently evaluate the coordination of the controlled agents. If the answer to the assume-guarantee verification is positive, then the coordination is successful and the formal planning problem is solved. Otherwise, in case that the assume-guarantee coordination fails to succeed, a counterexample will be returned by the verification procedure; we then develop a counterexample-guided scheme to update the local mission specifications and to refine the local mission supervisors accordingly.

In order to apply the assume-guarantee (cf. Section 2.4) paradigm of compositional verification, we denote by

$$M_i := S_i || G_i \ (i \in I)$$

(4.8)
the component module of the mission execution of the $i$-th agent under control. By applying the idea of the $L^*_C$ algorithm, the mission supervisor $S_i$ is synthesized in a maximally permissive manner, i.e.,

$$L_m(M_i) = L(M_i) = \sup C(L_i^{m_i}) = \sup C(P_i(L) \cap L(G_i)) \subseteq P_i(L). \quad (4.9)$$

The collective execution of the local missions of the cooperative multi-agent system is then given by $M = \|_{i \in I} M_i$. With slightly abusing the notations, we use $L$ and $coL$ to denote the DFAs that recognizes $L$ and its complement language, respectively. The successful mission planning then requires that $M \models L$. We present a two-layer compositional verification framework at this point to justify collective performance of the controlled cooperative multi-agent system $M$, as illustrated in Fig. 4.12. In particular, the SYM-N proof rule (cf. Section 2.4.2) is employed by the compositional verification framework to determine whether or not $M \models L$ holds for the $n$ agents.

One of the essential ingredients of the application of the SYM-N proof rule is to compute the weakest assumptions (cf. Definition 2.12) for the component modules. Towards this end, we leverage the idea of inductive learning techniques; and at the heart of the compositional verification framework shown in Fig. 4.12 lies another modification of the $L^*$ algorithm, namely the $L^*_{CV}$ algorithm, to learn the assumption DFAs for the controlled agent $M_i (i \in I)$. The reason why the conventional $L^*$ algorithm cannot be applied directly for the assume-guarantee paradigm is that it is impractical for the Teacher to answer the equivalence queries, i.e., the Teacher is not able to answer whether or not a string $t \in L_m(A_i)$ a priori, where $A_i$ is the DFA of an appropriate assumption\(^2\) associated with a component module $M_i (i \in I)$ in the SYM-N proof rule. The membership and equivalence queries of the $L^*_{CV}$ algorithm, along with the working procedure of the compositional

\(^1\)The subscript “CV” stands for “compositional verification”.

\(^2\)By an appropriate local assumption $A_i (i \in I)$ we mean that the assumption $A_i$ satisfies Premise $i$ of the SYM-N proof rule (with respect to $M_i$ and $L$)
verification framework in Fig. 4.12, are introduced in the sequel.

**Membership Queries** As shown in Fig. 4.12, in the first layer, the $L^*_CV$ algorithm deploys $n$ local Teachers, each of which corresponds to a controlled agent $M_i$ ($i \in I$) and is used to compute the DFA of the local assumption. More specifically, it follows from the SYM-N proof rule that for each $i \in I$, the event set of an assumption DFA $A_i$, denoted as $\Sigma(A_i)$, satisfies that

$$\Sigma(A_i) \subseteq \left( \bigcap_{i \in I} \Sigma_i \right) \bigcup \Sigma(L),$$  

(4.10)

where $\Sigma(L) = \Sigma$ is the event set of $L$ in this chapter.

At the $j$-th iteration step, the $L^*_CV$ algorithm should first learn an appropriate assumption $A^j_i$ for the controlled agent $M_i$ on the basis of the iteration history and the answers to the membership queries of the $L^*_CV$ algorithm. As the $L^*_CV$ algorithm executes the learning procedure iteratively, it incrementally (in the sense of the number of states) constructs a

---

Figure 4.12. The compositional verification framework.
sequence of assumption DFAs \( \{ A^j_i \}_{j \in \mathbb{N}} \) for \( M_i \), and we expect the sequence \( \{ A^j_i \} \) to converge to the DFA that recognizes the weakest appropriate assumption \( A^w_i \) within a finite step of iterations.

The membership queries that are employed by the \( L^*_CV \) algorithm is constructed on the basis of the following lemma.

**Lemma 4.1.** [132] Let \( t \in \Sigma^* \) and let \( L \) (which is represented by a DFA) be a regular property. For each \( i \in I \), let \( A^w_i \) be the weakest appropriate assumption of \( M_i \) and let \( \Sigma(A^w_i) \) satisfy (4.10). Then \( t \in L_m(A^w_i) \) if and only if \( \langle \text{DFA}(t) \rangle M_i \langle L \rangle \).

Lemma 4.1 hence provides the Teachers 1 to \( n \) with the guidelines of building up the membership queries for the construction of the local assumption DFAs for each controlled agent. Specifically, for each \( i \in I \) and \( t \in \Sigma^* \), the Teacher \( i \) constructs an observation table by answering the following membership queries:

\[
T_i(t) = \begin{cases} 
1, & \text{if } \langle \text{DFA}(t) \rangle M_i \langle L \rangle, \\
0, & \text{otherwise.} 
\end{cases} \tag{4.11}
\]

In practice, the membership queries (4.11) can be implemented by a conventional model checker. Instead of justifying whether or not a string \( t \in \Sigma^* \) satisfies \( t \in L_m(A^w_i) \), the Teacher answers whether or not \( \langle \text{DFA}(t) \rangle M_i \langle L \rangle \) holds. It then follows from Lemma 4.1 that if the model checker returns a positive answer, then \( t \in L_m(A^w_i) \). Note that in general, the size of \( \text{DFA}(t) \) is significantly smaller than the size of either \( A^w_i \) or \( M_i \) (\( i \in I \)); therefore the utilization of the \( L^*_CV \) algorithm provides a computationally efficient method of justifying \( M \models L \) and the computation of the composed system \( M \) can be avoided.

**Equivalence Queries** By answering the membership queries (4.11), the Teacher \( i \) is able to build up a closed and consistent observation table for the \( i \)-th controlled agent \( M_i \) \( (i \in I) \) by following the procedure (2.17). Afterwards, a local assumption DFA \( A^j_i \) is presented to the \( L^*_CV \) algorithm as the conjecture. The \( L^*_CV \) algorithm then presents two
types of equivalence queries to the Teacher in order to determine whether or not $A_i^j = A_i^w$. First, rather than answering $L_m(A_i^j) = L_m(A_i^w)$, the local Teacher $i$ needs to answer the equivalence conjecture

$$\langle A_i^j \rangle_{M_i} \langle L \rangle$$ (4.12)

by means of compositional model checking techniques. Once the Teacher denies the conjecture, a counterexample $t \in \Sigma^*$ is returned by the Teacher. According to Lemma 4.1, $P_{A_i}(t) \in L_m(A_i^j)$ but $P_{A_i}(t) \notin L_m(A_i^w)$, where $P_{A_i}$ is the natural projection from $\Sigma^*$ to $\Sigma_{A_i}$, indicating that $P_{A_i}(t)$ witnesses the difference between $A_i^j$ and $A_i^w$. Thus, $t$ is a local counterexample and the $L_{CV}^*$ algorithm adds $P_{A_i}(t)$ and all its prefixes back to the local iteration loop to update the observation tables until no more local counterexamples are generated. At this point, the obtained assumption DFA $A_i$ ($i \in I$) is an appropriate but not necessarily weakest assumption corresponding to $M_i$.

Next, when the local learning loop of the $L_{CV}^*$ algorithm terminates with no more counterexamples, we collect a family of appropriate assumptions $A_i$ for $M_i$ ($i \in I$). The $L_{CV}^*$ algorithm then involves the Teacher $n + 1$ in the second layer as shown in Fig. 4.12 to justify the $(n + 1)$-th premise of the SYM-N proof rule; i.e., to answer the equivalence conjecture

$$L_m(coA_1 \parallel \cdots \parallel coA_n) \subseteq L.$$ (4.13)

If the Teacher $n + 1$ returns “True”, then the $L_{CV}^*$ algorithm terminates with the conclusion that $M = ||_{i \in I} M_i = L$. Otherwise, Teacher $n + 1$ returns “False” with another counterexample $c \in \Sigma^*$.

**Counterexample Analysis** Once the Teacher $n + 1$ denies the equivalence conjecture (4.13) by returning a counterexample $c \in \Sigma^*$, the $L_{CV}^*$ algorithm needs to determine whether or not the coordination of the mission plans of all the agents indeed results in
the violation of the global mission \( L \). For such a pursuit, the \( L_{CV}^* \) algorithm justifies

\[
c \in L_m(M_i\|coL)
\]  \hspace{1cm} (4.14)

for each \( i \in I \). Based on the answer to (4.14), the counterexample \( c \) is analyzed as follows:

(i) if \( (\exists i \in I)[c \notin L_m(M_i\|coL)] \), then there exists at least one \( i \in I \) such that \( c \) is not a violating string for \( M_i \). In this case, it can be concluded that the current local assumption \( A_i \) associated with \( M_i \) is appropriate but is stronger than the weakest one \( A_i^{w} \), i.e., \( L_m(A_i) \subseteq L_m(A_i^{w}) \) and \( P_{A_i}(c) \in L_m(A_i^{w}) - L_m(A_i) \). Hence, the \( L_{CV}^* \) algorithm treats \( c \) as another local counterexample and adds \( P_{A_i}(c) \) and all its prefixes back to the local iteration loop, and the Teacher \( i \) is employed again to weaken the assumption \( A_i \).

(ii) Otherwise, if \( (\forall i \in I)[c \in L_m(M_i\|coL)] \), then \( c \) turns out to be a common violating string of all agents and the controlled cooperative multi-agent system \( M = \|_{i \in I} M_i \) indeed violates \( L \). As shown in Fig. 4.1, the violation of \( L \) leads to occurrences of undesired mission behaviors in the coordination of the local mission plans of each agent. In this case, a counterexample-guided re-synthesis of local mission plans will be triggered (see the next subsection for more details).

*The \( L_{CV}^* \) Algorithm* Based on the membership and equivalence queries introduced in (4.11), (4.12) and (4.13), respectively, along with the counterexample justification (4.14), the working procedure of the \( L_{CV}^* \) algorithm is summarized as Algorithm 4, where the notation \( L_{CV}^*(M,L) \) means that the algorithm justifies whether or not \( M = \|_{i \in I} M_i \models L \).

**Algorithm 4**: The \( L_{CV}^*(M,L) \) learning algorithm

| Input: The property DFA \( L \), the event set \( \Sigma \), and the local event sets \( \Sigma_i \) (\( i \in I \)). |
| Output: True if \( M = \|_{i \in I} M_i \models L \), or False with a counterexample \( c \in \Sigma^* \) otherwise. |
| 1: \( S \leftarrow \{\epsilon\} \), \( E \leftarrow \{\epsilon\} \), \( j \leftarrow 0 \) |
| 2: for all \( i \in I = \{1,2,\ldots,n\} \) do |
| 3: Select \( \Sigma(A_i) \) such that (4.10) is satisfied |
| 4: Construct \( (S_i, E_i, T_i) \) with membership function \( T_i \) (4.11) |

102
repeat
while \((S_i, E_i, T_i)\) is not closed or consistent do
  if \((S_i, E_i, T_i)\) is not closed then
    Find \(s \in S_i, \sigma \in \Sigma(A_i)\) such that \(\forall s' \in S_i : \text{row}(s') \neq \text{row}(s\sigma)\)
    \(S_i \leftarrow S_i \cup \{s\sigma\}\)
  end if
  if \((S_i, E_i, T_i)\) is not consistent then
    Find \(s_1, s_2 \in S_i, \sigma \in \Sigma(A_i)\) and \(e \in E_i\) such that \(\text{row}(s_1) = \text{row}(s_2)\)
    but \(T_i(s_1\sigma e) \neq T_i(s_2\sigma e)\)
    \(E_i \leftarrow E_i \cup \{\sigma e\}\)
  end if
  Extend \(T_i\) to \((S_i \cup S_i\Sigma(A_i))E_i\) using the membership queries (4.11)
end while
Construct \(A^i_j = M(S_i, E_i, T_i)\) based on the procedure in (2.17)
Perform the equivalence queries (4.12) with respect to \(L, M_i\) and \(A^i_j\)
if \(t \neq \epsilon\) and \(\langle A^i_j \rangle M_i\langle L \rangle\) then
  \(S_i \leftarrow S_i \cup P_{A_i}(\bar{t})\)
  \(j \leftarrow j + 1\)
  Extend \(T_i\) to \((S_i \cup S_i\Sigma)E_i\) using membership queries (4.11)
end if
until \(t = \epsilon\)
end for
while \(L_m(\text{co}A_1 \| \cdots \| \text{co}A_n) \not\subseteq L\) do
  \(c \leftarrow \) counterexample from the Teacher \(n + 1\)
  for all \(i \in I = \{1, 2, \ldots, n\}\) do
    if \(c \in L_m(M_i\|\text{co}L)\) then return False return \(c \in \Sigma^*\)
    else if \((\exists k \in I) [c \notin L_m(M_k\|\text{co}L)]\) then
      \(S_k \leftarrow S_k \cup P_{A_k}(c)\)
      Extend \(T_k\) to \((S_k \cup S_k\Sigma)E_k\) using membership queries (4.11)
    end if
  end for
end while return True

We use the following example in a two-robot coordination scenario to illustrate the working procedure of the \(L_{CV}^*\) algorithm.

**Example 4.2.** Consider two controlled robots, namely robot \(M_1\) and \(M_2\), both of which are equipped with identical localization and communication capabilities, along with a robotic gripper, as shown in Fig. 4.13.
In this example, $M_1$ should grab some parts from a work station and then puts them down in a designated area, which can be accomplished by executing the “down” event. Afterwards, it sends an event “send” to inform $M_2$ that it finishes the job. After being sent the information that $M_1$ has put down the parts, $M_2$ should move to the area and grab the parts to the next work station by executing the “up” event. Then, $M_2$ acknowledges $M_1$ that it has finished its work by sending $M_1$ the “ack” event. Finally, both robots return to their initial states so the process can be repeated. The DFA representations of $M_1$ and $M_2$ are depicted in Fig. 4.14 (a) and (b), respectively.

![Figure 4.13. A robot equipped with a gripper.](image)

![Figure 4.14. The two-robot coordination system.](image)
The global mission (property) of the two-robot team should capture the desired behaviors of the robots shown in Fig. 4.14. In particular, the global mission expresses the fact that putting down and grabbing up the parts should appear in pairs, with the parts should first be put down at the designated area and then be grabbed up. Therefore, the DFA of \( L \) can be obtained as shown in Fig. 4.15 (a), along with the completion DFA \( \tilde{L} \) of \( L \), as shown in Fig. 4.15 (b). The goal of the compositional verification is to determine whether or not \( M_1 \| M_2 \models L \).

![Diagram of the two-robot coordination system](Image)

Figure 4.15. The two-robot coordination system.

We apply the \( L_{CV}^* \) algorithm for the computation of the assumptions for the robots \( M_1 \) and \( M_2 \) in this example. We first set the event set of the assumption \( A_1 \) to be \( \Sigma(A_1) = \{up, send, ack\} \). By using the membership queries (4.11), we can obtain the first closed and consistent table, along with the corresponding \( A_1^1 \), as shown in Fig. 4.16.
The $L^e_{CV}$ algorithm utilizes the equivalence queries (4.12) to answer the conjecture $A^1_1$ by checking $\langle A^1_1 \rangle G_1 \langle L \rangle$. In this case, the Teacher denies the equivalence conjecture and returns a counterexample $t = \text{down send ack down}$. It then can be inferred from Theorem 2.5 that there exists a string in $A^1_1 \parallel M_1 \parallel \text{coL}$ that leads to the error state $q_e$. The Teacher then returns $P_{\Sigma(A_1)}(t) = \text{send.ack}$ to $L^e_{CV}$ to update the assumption, and the closed and consistent observation table $T_2$ is constructed with the corresponding assumption DFA $A^2_1$, which is shown in Fig. 4.17. It turns out that $A^2_1$ is an appropriate assumption for $M_1$ and is sufficient for the compositional verification.

Figure 4.16. The observation table $T_1$ and $A^1_1$. 

<table>
<thead>
<tr>
<th>$T_1$ in for Robot 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
</tr>
<tr>
<td>$S$</td>
</tr>
<tr>
<td>up</td>
</tr>
<tr>
<td>$S \Sigma$</td>
</tr>
<tr>
<td>up</td>
</tr>
<tr>
<td>ack</td>
</tr>
<tr>
<td>up.send</td>
</tr>
<tr>
<td>up.up</td>
</tr>
<tr>
<td>up.ack</td>
</tr>
</tbody>
</table>

$A^1_1$: 

\[
\text{send.ack}
\]
Since in this example only two robots are involved, we use the ASYM proof rule as a computationally convenient alternation of the SYM-N proof rule in this example. In this case, the Teacher then needs to check whether or not \( \langle \text{True} \rangle M_2 \langle A_1^2 \rangle \) holds (corresponding to the second premise of the ASYM proof rule). It turns out that this is also true, thus thanks to the \( L_{CV}^* \) algorithm we can conclude that \( M_1 \parallel M_2 \models L \).

Correctness and Termination Finally, the correctness and termination properties of the \( L_{CV}^* \) algorithm are summarized in the following theorem.

**Theorem 4.1.** For the global mission \( L \) and the component modules \( M_1, M_2, \ldots, M_n \), the
$L^*_{CV}$ algorithm as shown in Fig. 4.12 with the SYM-N proof rule terminates within a finite number of iterations and correctly returns whether or not $M = \|_{i \in I} M_i \models L$.

Proof. (Correctness) By comparing the membership queries (4.11) and the equivalence queries (4.12) and (4.13) with the SYM-N proof rule, the Teacher of the $L^*_{CV}$ algorithm returns true and stops presenting counterexamples only if all the $(n + 1)$ premises of the SYM-N proof rule are satisfied. Therefore, it follows from Theorem 2.6 that the correctness of the $L^*_{CV}$ algorithm can be guaranteed by the soundness of the proof rule. Otherwise, the Teacher only reports a counterexample if the equivalence conjecture (4.13) fails to hold, which implies that there exists a violating string in all the component modules, and thus $M = \|_{i \in I} M_i$ also violates $L$.

(Termination) At each iteration step of the $L^*_{CV}$ algorithm, the Teacher either reports whether or not $M = \|_{i \in I} M_i \models L$ indeed holds (in which case the algorithm terminates afterwards), or evolves to the next iteration by returning a counterexample to the $L^*_{CV}$ algorithm. It can then be implied from Lemma 4.1 that if the $L^*_{CV}$ algorithm keeps receiving counterexamples, the $n$ local Teachers eventually construct $A^w_1, A^w_2, \ldots, A^w_n$ for the component modules, respectively. According to Definition 2.12, premises 1 through $n$ of the SYM-N proof rule are automatically satisfied with respect to $A^w_i (i \in I)$. The Teacher then proceeds to check the $(n + 1)$-th premise with the equivalence conjecture (4.13), which either returns true and hence terminates, or returns false with a new counterexample. Since the weakest assumptions for all the component modules have already been used at this iteration step, it then follows from Theorem 2.6 that the completeness of the SYM-N proof rule guarantees that this counterexample indeed reveals a violation of $L$, and hence the $L^*_{CV}$ algorithm also terminates with the conclusion that $M = \|_{i \in I} M_i \not\models L$.  

4.5.3 Counterexample-guided Re-synthesis of the Mission Plans

Starting from the initialized local mission $L^{mi}_i (i \in I)$ (4.6), the $L^*_C$ algorithm is able to compute a maximally permissive mission supervisor $S_i$ such that $L(S_i || G_i) = \sup C(L^{mi}_i)$.  

108
For convenience of presentation, we still use $L_i^{mi}$ to denote the mission plan that is executed by the agent $G_i$ after the synthesis of $S_i$, i.e.,

$$L_i^{mi} = L(S_i | G_i) = L(M_i).$$ \hspace{1cm} (4.15)

After obtaining $S_i$, all the controlled agents $M_i (i \in I)$ are presented to the $L_{CV}^*$ algorithm to justify whether or not $\|_{i \in I} M_i \models L$. If the answer is positive, then the formal mission planning of the cooperative multi-agent system terminates with a series of satisfactory mission supervisors $S_i (i \in I)$. Otherwise, the $L_{CV}^*$ algorithm returns a counterexample $c = L_{CV}^*(M, L)$, indicating that the joint execution of $L_i^{mi} (i \in I)$ will violate the global mission $L$. In this case, we propose a formal mission planning scheme that invokes a counterexample-guided re-synthesis of the local mission supervisors.

**Local Mission Refinement** The re-synthesis procedure starts with a counterexample-guided local mission refinement process. In fact, rather than eliminating a single counterexample $c = L_{CV}^*(M, L)$ that is provided by the $L_{CV}^*$ algorithm, we need to eliminate the set of strings that are observationally equivalent to $c$ from each agent’s synthesized mission plan, i.e.,

$$L_i^{temp} := L_i^{mi} - P_i(c);$$ \hspace{1cm} (4.16)

Note that for each $i \in I$, $L_i^{temp} \subseteq L_i^{mi} \subseteq L(G_i)$ but $L_i^{temp}$ is not necessarily prefix-closed. Therefore, in order to apply the $L_{C}^*$ algorithm to re-synthesize the local mission supervisor, it is necessary to provide the agent $G_i (i \in I)$ with the supremal prefix-closed sublanguage of $L_i^{temp}$, which is given by

$$\tilde{L}_i^{mi} := L_i^{temp} - (\Sigma_i^* - L_i^{temp})\Sigma_i^*,$$ \hspace{1cm} (4.17)

Thus, $\tilde{L}_i^{mi}$ is used as the new local mission for the agent $G_i$ to synthesize the new local mission supervisor.
Re-synthesis of Mission Supervisors

As shown in the formal planning framework in Fig. 4.1, the refined local mission $\tilde{L}_{mi}$ is assigned to the agent $G_i \ (i \in I)$ to update the associated mission supervisor via the $L_C^*$ algorithm. More specifically, for the updated local mission supervisor $S_i$, we can write that

$$\tilde{S}_i = L_C^*(G_i, \Sigma_{i,uc}, \tilde{L}_{mi}).$$

(4.18)

To pursue succinct notations, we also slightly abuse the notations and denote $\tilde{L}_{mi} = L(\tilde{S}_i \| G_i)$. Furthermore, the component module corresponding to the agent $G_i$ under the supervision of the updated local mission supervisor $\tilde{S}_i$ is given by $\tilde{M}_i = \tilde{S}_i \| G_i$. We then present the component modules $\tilde{M}_i \ (i \in I)$ to the compositional verification framework shown in Fig. 4.12. Once the $L_{CV}^*$ algorithm denies the verification and returns with a new counterexample $c' \in \Sigma^*$, we follow the local mission refinement procedure in (4.16) and (4.17) and then to update the local mission supervisor (4.18) by applying the $L_C^*$ algorithm. As we will show in the sequel, the above procedure will keep iterating until no more counterexamples are returned from the $L_{CV}^*$ algorithm.

Correctness and Termination

Our investigation of the correctness and termination of the of the proposed formal mission planning framework for cooperative multi-agent system starts with the following property of separate controllability of the global mission, which characterizes the solvability of the mission planning part of Problem 4.1 for the cooperative multi-agent system.

Definition 4.2 (Separate Controllability). Consider the cooperative multi-agent system that consists of $n$ agents, and for each $i \in I$, the mission DFA $G_i$ of the $i$-th agent is equipped with the set of controllable missions $\Sigma_{i,c} \subseteq \Sigma_i$. A prefix-closed mission $L \subseteq \Sigma^*$ is said to be separately controllable with respect to $\Sigma_i, \Sigma_{i,uc}$ and $G_i \ (i \in I)$ if there exists a language $L_i \subseteq \Sigma_i^*$ for each $i \in I$ such that

(i) $L = \|_{i \in I} L_i$;
(ii) \( L_i \) is controllable with respect to \( G_i \) and \( \Sigma_{i,uc} \).

From Definition 4.2, separate controllability of \( L \) indicates that \( L \) is separable with respect to the local mission sets \( \Sigma_i \ (i \in I) \), and there exists a generating set \( \{ L_i \}_{i \in I} \in B(L) \) (2.7) such that \( L_i \) is a locally controllable mission for all \( i \in I \). The following theorem implies that the separate controllability be a sufficient and necessary condition for the existence of satisfactory mission supervisors that solve the cooperative multi-agent mission planning problem.

**Theorem 4.2.** Consider a cooperative multi-agent system \( G \) that consists of \( n \) agents, each of which admits a mission DFA \( G_i \) \( (i \in I) \) with local controllable mission events \( \Sigma_{i,c} \) and local uncontrollable mission events \( \Sigma_{i,uc} \). Given a non-empty and prefix-closed global mission \( L \subseteq L(G) \), there exists a series of local mission supervisors \( S_i \) \( (i \in I) \) for each agent \( G_i \) such that \( \| \bigcup_{i \in I} L(S_i \parallel G_i) = L \) if and only if \( L \) is separately controllable with respect to \( \Sigma_i, \Sigma_{i,uc} \) and \( G_i \) \( (i \in I) \).

**Proof.** \((\Leftarrow)\) When \( L \subseteq L(G) \) is separately controllable with respect to \( \Sigma_i, \Sigma_{i,uc} \) and \( G_i \), Definition 4.2 suggests that there exists a local mission language \( L_i \subseteq \Sigma_i^* \) such that \( \| \bigcup_{i \in I} L_i = L \) and \( L_i \) is controllable with respect to \( G_i \) and \( \Sigma_{i,uc} \), which implies that for each \( i \in I \), there exists a local supervisor \( S_i \) such that \( L(S_i \parallel G_i) = L_i \). Thus, \( \| \bigcup_{i \in I} L(S_i \parallel G_i) = \| \bigcup_{i \in I} L_i = L \) is assured.

\((\Rightarrow)\) Conversely, we assume that there exist local supervisors \( S_i \) \( (i \in I) \) such that \( \| \bigcup_{i \in I} L(S_i \parallel G_i) = L \). With slightly abusing the notations, let \( L_i^{mi} := L(S_i \parallel G_i) \subseteq L(G_i) \) be the mission executed by the controlled agent \( G_i \) under the supervision of \( S_i \) \( (i \in I) \). Since \( L_i^{mi} \) is a controlled behavior for the agent \( G_i \), it is clearly controllable with respect to \( G_i \) and \( \Sigma_{i,uc} \). Furthermore, it follows from \( \| \bigcup_{i \in I} L_i \) that \( (L_1, L_2, \ldots, L_n) \in B(L) \) (2.7), thus \( L \) is separately controllable with respect to \( \Sigma_i, \Sigma_{i,uc} \) and \( G_i \) \( (i \in I) \), which completes the proof. \qed
The following theorem states that in general, the separate controllability is a strictly stronger notion than the combination of global controllability and separability.

**Theorem 4.3.** For the cooperative multi-agent system $G$ that is composed of the agents $G_i$ ($i \in I$), if a non-empty prefix-closed specification language $L \subseteq L(G)$ is separately controllable with respect to $\Sigma_i$, $\Sigma_{i,uc}$ and $G_i$, then $L$ is separable with respect to $\{\Sigma_i\}_{i \in I}$ and is controllable with respect to $G$ and $\Sigma_{uc}$.

**Proof.** First, from Definition 4.2, the separate controllability of $L$ trivially assures the separability. Next we show that $L$ is globally controllable. Let $L_i \subseteq \Sigma_i^*$ ($i \in I$) be a set of prefix-closed languages that satisfies the conditions in Definition 4.2. Therefore, $L_i$ is controllable with respect to $G_i$ and $\Sigma_{i,uc}$, i.e.,

$$L_i \Sigma_{i,uc} \cap L(G_i) \subseteq L_i.$$  \hfill (4.19)

It then follows from the monotonicity of the inverse projection $P_i^{-1}$ that

$$P_i^{-1}[P_i(L) \Sigma_{i,uc} \cap L(G_i)] \subseteq P_i^{-1}(P_i(L_i)).$$  \hfill (4.20)

holds for each $i \in I$. Since

$$P_i^{-1}[L_i \Sigma_{i,uc} \cap L(G_i)] = P_i^{-1}(L_i \Sigma_{i,uc}) \cap P_i^{-1}[L(G_i)]$$  \hfill (4.21)

and

$$L = \bigparallel_{i \in I} L_i = \bigcap_{i \in I} P_i^{-1}(L_i),$$  \hfill (4.22)
we have

\[ L\Sigma_{i,uc} \cap L(G) = \left[ \bigcap_{i \in I} P_i^{-1}(L_i) \right] \Sigma_{i,uc} \cap \left[ \bigcap_{i \in I} P_i^{-1}[L(G_i)] \right] \]

\[ \subseteq P_i^{-1}(L_i)\Sigma_{i,uc} \cap P_i^{-1}[L(G_i)] \]  \hspace{1cm} (4.23)

\[ \subseteq P_i^{-1} [P_i(L)\Sigma_{i,uc}] \cap P_i^{-1}[L(G_i)] \]

\[ = P_i^{-1} [L_i\Sigma_{i,uc} \cap L(G_i)] \subseteq P_i^{-1}(L_i). \]

Note that the last inclusion in (4.23) always holds for any \( i \in I \). Therefore, we can write that

\[ \bigcap_{i \in I} [L\Sigma_{i,uc} \cap L(G)] \subseteq \bigcap_{i \in I} P_i^{-1}(L_i) = \|_{i \in I} L_i = L. \] \hspace{1cm} (4.24)

Note that by definition, \( \Sigma_{uc} = \Sigma - \Sigma_c = \Sigma - \bigcup_{i \in I} \Sigma_{i,c} = \bigcap_{i \in I} \Sigma_{i,uc} \), the above equation is equivalent to

\[ L\Sigma_{uc} \cap L(G) \subseteq L, \] \hspace{1cm} (4.25)

which ensures the global controllability of \( L \) with respect to \( G \) and \( \Sigma_{uc} \). The proof is hence completed.

\[ \square \]

**Remark 4.3.** From the proof of Theorem 4.3, it is worth pointing out that the assumption (4.2) is never used to prove the theorem. Therefore, different from the necessary and sufficient conditions established in [187], which include separability, assumption (4.2) and also global controllability, we propose alternative necessary and sufficient conditions that may get rid of the constraint posed by (4.2). As a trade-off, we need to use the notion of separate controllability, which is in general stronger than the combination of separability and global controllability.

Based on the aforementioned properties of separately controllable languages, the following theorem is established to illustrate the correctness of the proposed mission planning scheme.
Theorem 4.4. The integration of the $L^*_C$ and the $L^*_CV$ algorithms in the mission planning scheme shown in Fig. 4.1 returns satisfactory local mission supervisors $S_i$ (as well as local mission plans $L_i^{mi}$) for the agent $G_i$ ($i \in I$) such that the collective mission executions of the cooperative multi-agent system $G$ achieve a separately controllable sublanguage $\tilde{L}$ of $L(G)$ such that $\tilde{L} \models L$.

Proof. When $L$ is separable, the correctness of the proposed mission planning procedure trivially holds. When $L$ is not separable, we proceed to the alternative deployment of the $L^*_C$ and the $L^*_CV$ algorithms. At each iteration step of the $L^*_CV$ algorithm, whenever Teacher $n + 1$ denies the compositional verification, it returns a counterexample $c \in \Sigma^*$. From the counterexample analysis procedure (4.14), the counterexample $c$ is actually a string that belongs to $\|_{i \in I} L_i^{mi} = L$. The first step (4.16) actually eliminates all the observationally indistinguishable local mission behaviors with respect to $c$ from the original local mission plan $L_i^{mi}$; while the second step (4.17) in fact computes the supremal prefix-closed sublanguage [91] $\tilde{L}_i^{mi} (i \in I)$ of the resulting language in the first step. Since the initial $L_i^{mi}$ satisfies that $L_i^{mi} \subseteq L(G_i)$, for the refined local mission $L_i^{mi}$, we can write that

$$\tilde{L}_i^{mi} = \overline{L_i^{mi}} \subseteq L_i^{temp} \subseteq L(G_i).$$

(4.26)

On the other hand, when $\tilde{L}_i^{mi}$ is obtained as the refined local mission specification for the agent $G_i$ ($i \in I$), the $L^*_C$ algorithm can synthesize the updated mission supervisor $\tilde{S}_i$ such that

$$L(\tilde{M}_i) = L(\tilde{S}_i\|G_i) = \sup C_i(\tilde{L}_i^{mi}) \subseteq \tilde{L}_i^{mi}. \quad (4.27)$$

In this case, if the $L^*_CV$ algorithms denies the verification $\|_{i \in I} \tilde{M}_i \models L$ and generates another counterexample $c' \in \Sigma^*$, we use $c'$ to refine $\tilde{L}_i^{mi}$ and $\tilde{S}_i$ again. During this iterative procedure, we can obtain the following “monotonic” sequence of local mission plans for each agent $G_i$:

$$\emptyset \subseteq \cdots \tilde{L}_i \subseteq \tilde{L}_i^{mi} \subseteq L_i^{mi} = \sup C_i(L_i^{mi}) \subseteq L_i^{mi}, \quad (4.28)$$
where for the clarity of presentation, we use $L_{i,0}^{mi}$ in the above inclusion sequence to denote the initial $L_{i}^{mi}$ that is computed via (4.6).

If the formal planning procedure terminates with no more counterexamples from the $L_{CV}$ algorithm, we can write with slightly abusing the notations that

$$\tilde{L} := \| L_{\bar{I}}^{mi} \subseteq \|_{i \in I} L(G_i) = L(G)$$ (4.29)

indeed forms a separately controllable language. Furthermore, according to (4.6), it can be concluded that $\tilde{M} = \|_{i \in I} \tilde{M}_i \models L$, which solves the mission planning of the cooperative multi-agent system $G$.

Finally, we claim that the aforementioned mission planning procedure terminates within a finite number of iterations.

**Theorem 4.5.** Consider a cooperative multi-agent system $G$ that consists of $n$ agents, each of which admits a mission DFA $G_i$ ($i \in I$) with local controllable mission events $\Sigma_{i,c}$ and local uncontrollable mission events $\Sigma_{i,uc}$. The integration of the $L_{CV}$ and the $L_{CV}^*$ algorithms in the mission planning scheme shown in Fig. 4.1 terminates within a finite number of iterations.

**Proof.** On the one hand, from (4.28), we know that after applying both the $L_{CV}$ and the $L_{CV}^*$ algorithms for a local mission $L_{i}^{mi}$, the counterexample-guided refinement of the local mission supervisor always results in an updated local mission $\tilde{L}_{i}^{mi}$ such that $\tilde{L}_{i}^{mi} \subseteq L_{i}^{mi}$. On the other hand, from Theorem 4.1, the $L_{CV}^*$ algorithm always terminates with generating no more counterexamples. Therefore, the formal mission planning framework always terminates (but it is possible to return trivial solutions, as will be discussed later in this section).

We now use the multi-robot coordination example in Section 4.3.2 to illustrate the application of the $L_{CV}^*$ algorithm and the proposed mission planning scheme.
Example 4.3. From Section 4.4.2, the local mission supervisor $S_i$ ($i \in I$) can be successfully synthesized for the robot $G_i$ to fulfill the (initial) local specification $L^m_i$. Next, it is necessary to check whether or not the collective executions of the mission plans will accomplish the global mission $L$. Towards this end, we present $M_i = S_i \parallel G_i$ ($i \in I$) as the component modules to the $L^*_CV$ algorithm to justify whether or not $M_1 \parallel M_2 \parallel M_3 \models L$.

In this example, a counterexample $c = G_1 inR_3 G_3 inR_3$ is returned by the $L^*_CV$ algorithm, indicating a common violation of $L$ for all the robots. We examine this counterexample and it turns out that when one of the robots $G_1$ or $G_3$ stays in $R_1$, the other robot must stay in $R_3$; otherwise, as indicated by $c$, it is impossible to open the two-way door $D_1$. By adding $P_{A_i}(c) = c$ (note that in this example, $\Sigma(A_i) = \Sigma$ for all $i \in I$) and all of its prefixes back to the $L^*_CV$ algorithm. We then follow (4.16) and (4.17) to update the local mission specifications accordingly. By applying the $L^*_C$ algorithm with respect to the updated local mission specifications, the new local mission supervisors $\tilde{S}_i$ ($i \in I$) are derived as shown in Fig. 4.18, Fig. 4.19 and Fig. 4.20, respectively.

![Figure 4.18. Updated local mission supervisor $\tilde{S}_1$.](image-url)
Intuitively, the updated local mission supervisors $\tilde{S}_1$ and $\tilde{S}_3$ explicitly specify that the robot $G_1$ stays in $R_1$, while the robot $G_3$ enters $R_3$ such that $D_1$ can be opened jointly. We present the updated $\tilde{M}_i = \tilde{S}_i \| G_i$ to the $L_{CV}^*$ algorithm to justify whether the collective behaviors of the local supervisors cooperatively satisfy $L$; and in this time the $L_{CV}^*$ algorithm reports no more counterexample, which implies that $S_i$ ($i \in I$) can steer $G_i$ to accomplish an appropriate local mission. Since all the mission events in this example are assumed to be locally observable, Fig. 4.18, Fig. 4.19 and Fig. 4.20 also demonstrate the
DFA representations of both the implementation of the mission supervisors $\tilde{S}_i$ and the local mission plans $\tilde{L}_{mi}^i$ ($i \in I$), respectively.

**Remark 4.4.** We use $L_{mi}^i$ to denote the synthesized mission plans for the agent $G_i$ ($i \in I$) after the mission planning scheme in this chapter hereafter.

Soundness and Completeness Theorem 4.4 in fact claims the soundness of the formal mission planning framework developed in this section. In other words, if the $L^*_CV$ algorithm terminates with a positive answer, then the formal mission planning part of Problem 4.1 is solved successfully. Nevertheless, we have to admit that the completeness of the proposed planning framework does not generally hold. Thus it could be possible that the formal mission planning framework returns a trivial solution, i.e., $L_{mi}^i = \{\epsilon\}$ for all $i \in I$. In this case, we can conclude that there does not exist a separately controllable sublanguage $\tilde{L} \subseteq L(G)$ such that $\tilde{L} \models L$. To prevent this situation from arising, we do not accept the trivial solution returned by the mission planning procedure. Instead, we inherit the initial mission plan $L_{mi}^i$ computed from (4.6) and synthesize the corresponding local mission supervisor $S_i$ (4.7) for the agent $G_i$ ($i \in I$). Although in this case, $\|\sum_{i \in I} S_i\| G_i \not\models L$, it still provides us with an acceptable solution in the sense that at least the global mission $L$ can be accomplished as a consequence of the coordination of the agents.

4.6  Formal Motion Planning of Cooperative Agents

In this section, we deal with the problem of computing a satisfactory motion plan for an individual agent after deducing a mission plan. From an ACPS point of view, a motion plan should include both the cyber part and the physical part in applications. On the one hand, the cyber part of a motion plan describes an ordered sequence of regions to be visited by an individual agent so that its local mission can be accomplished. On the other hand, the physical part of the motion plan should also concern with the computation of continuous trajectories that can be tracked by the agent. In this section, we focus on the computa-
tion of formal (discrete) motion plans in the presence of environment uncertainties, while
dynamics-based control theory are applied for the implementation of the motion and path
plans in the physical level in Chapter 6.

4.6.1 Automatic Generation of Motion Plans

After the design of feasible local missions $L^m_i (i \in I)$, the motion planning problem
concentrates on finding a set of motion plans $L^{mo}_i$ associated with each $L^m_i$ such that the
integrated mission-motion plans $LP_i (i \in I)$ can solve Problem 4.1. A local motion plan
is composed of two ingredients: a local motion plan $L^{mo}_i \subseteq V^*$ that enumerates all the
regions visited by the agent, and a local door profile $D^{mo}_i \subseteq D^*$ associated with $L^{mo}_i$ that
records all possible doors through which the agents shall pass. It is desired that the local
motion plan is adequate, which is defined as follows.

**Definition 4.3 (Adequate Motion Plans).** A local motion plan $L^{mo}_i$ for the agent $G_i (i \in I)$
is said to be adequate if

(i) $L^{mo}_i \models \pi_i (L^m_i)$;

(ii) $L^{mo}_i \subseteq \text{Run} [L(G^m_i)]$.

From Definition 4.3, the synthesis of the local motion plan starts by exploiting the
mission-motion integration relation $\pi_i (i \in I)$. The intuition behind Definition 4.3 is that
$L^{mo}_i$ shall obey restrictions imposed not only by $L^m_i$, but by $G^m_i$ as well (adequacy).

Note that the mission-motion integration mapping $\pi_i$ maps the prefix-closed language
$L^m_i$ over $\Sigma_i$ into a prefix-closed language $\pi_i (L^m_i)$ over $V$. To efficiently compute the
motion plan for each agent (which is not known *a priori*), we employ a third modification
of the $L^*$ algorithm, termed as the $L^{MP}_*^3$ algorithm, to pursue an adequate local motion
plan. Specifically, the Teacher of the $L^{MP}_*$ algorithm is designed to determine the following

---

$^3$The subscript “MP” stands for “motion planning”.

119
membership queries for the agent $G_i$: for $i \in I$ and $t \in V^*$:

$$T_i(t) = \begin{cases} 
1, & \text{if } \text{DFA}(t) \models \pi_i(L_{mi}^i), \\
0, & \text{otherwise},
\end{cases} \quad (4.30)$$

where DFA$(t)$ is defined similarly to the membership queries (4.11) for the $L_{CV}^*$ algorithm, except that the underlying DFA is defined over $V$. Besides justifying the membership queries (4.30), the Teacher of the $L_{MP}^*$ algorithm answers the conjecture

$$L_{mo}^i \subseteq \text{Run}[L(G_{mi}^i)]. \quad (4.31)$$

If the conjecture is denied, a counterexample $t \in V^*$ is produced. Since $L_{mo}^i \subseteq \text{Run}[L(G_{mi}^i)]$ is false, we know that $t$ witnesses a difference between $L_{mo}^i$ and $\text{Run}[L(G_{mi}^i)]$; therefore, it is returned to the $L_{MP}^*$ algorithm to update the motion plan $L_{mo}^i$.

The correctness and termination properties of the $L_{MP}^*$ algorithm can be summarized as the following theorem.

**Theorem 4.6.** Given the local mission plan $L_{mi}^i$ and the mission-motion integration mapping $\pi_i$, the $L_{MP}^*$ algorithm terminates and correctly constructs an adequate local motion plan $L_{mo}^i$ ($i \in I$).

**Proof.** (Correctness) The Teacher of the $L_{MP}^*$ algorithm is designed in order to satisfy both of the conditions in Definition 4.3. When a counterexample is generated, the counterexample in fact represents a string in the symmetric difference between $\pi_i(L_{mi}^i)$ and $\text{Run}[L(G_{mi}^i)]$; therefore, if no more counterexample is generated, the current learned D-FA $L_{mi}^i$ (with slightly abusing the notations) satisfies both of the requirements of Definition 4.3, which guarantees the correctness.

(Termination) At any iteration step, after a local motion plan DFA $L_{mo}^i$ is conjectured, the $L_{MP}^*$ algorithm reports whether or not $L_{mo}^i$ is adequate and terminates, or continues the
construction of $L_{i}^{mo}$ by providing new counterexamples. By Theorem 2.4, the $L'$ learning procedure eventually terminates at some iteration step $j \in \mathbb{N}$, at that time, the $L_{MP}^{*}$ algorithm produces an adequate motion plan $L_{i}^{mo}$.

Furthermore, the door profile associated with the synthesized local motion plan $L_{i}^{mo}$ can be computed as

$$D_{i}^{mo} = \text{String}(L_{i}^{mo}),$$

where the operator $\text{String}(\cdot)$ computes all the corresponding strings $s \in D^{*}$ by treating strings in $L_{i}^{mo}$ as runs of $G_{i}^{m}$, i.e., $\text{String}(L_{i}^{mo}) = \{s \in L(G_{i}^{m})|\text{Run}(s) \in L_{i}^{mo}\}$.

4.6.2 Counterexample-guided Re-planning in Uncertain Environments

An integrated local plan $L_{i}P_{i}$ for the agent $G_{i}$ ($i \in I$) can be computed on the basis of $L_{i}^{mi}$ and $L_{i}^{mo}$. It is desired that the integrated plans $L_{i}P_{i}$ ($i \in I$) can also be implemented in the real environment $E$ rather than the environment characterized by the prior knowledge $E_{0}$. As mentioned in Section 4.3, feasible transitions among adjacent regions in $E_{0}$ may become infeasible in $E$ since some doors that are supposed to be open are actually closed. During the online coordination of the cooperative multi-agent systems, the underlying agents may detect the real door mapping $F_{E}^{D}$ and the real region transition diagram $\delta_{E}^{\mathcal{D}}$ through sensing capabilities and communication with each other. Here we assume that the motion capacities

$$G_{i}^{E} = (V, D, \delta_{i}^{E}, v_{i,0})$$

of the agent $G_{i}$ ($i \in I$), under the restriction of $E$, are also captured by a trim DFA. Based on the online-acquired knowledge of the environment, we develop a new algorithm to address the environment uncertainty, whose outputs include both the feasible integrated plan $L_{i}P_{i}^{E}$ and the associated door profile $D_{i}^{m}$ for the agent $G_{i}(i \in I)$ moving and performing missions in $E$.

As detailed in Algorithm 5, the adaption of $L_{i}P_{i}$ to the practical environment $E$ starts
Algorithm 5: Implementation of $LP_i$ in $E$

**Input:** Local integrated plan $LP_i$, local motion model $G_i^{m}$ and local door profile $D_i^{mo}$

**Output:** Local implementable integrated plan $LP_i^e$

1: Initialization: $LP_i^e = LP_i$, $D_i^{m} = D_i^{mo}$
2: Order the words in $LP_i$: $LP_i = \bigcup_{k=1}^{K}LP_i^k$
3: Construct $D_i^{mo} = \bigcup_{k=1}^{K}D_i^k$: $D_i^k = \text{Word}[P_V(LP_i^k)]$
4: for all $k \in \{1, 2, \ldots, K\}$ do
5: Let $LP_i^k = LP_i^k(0)LP_i^k(1) \ldots LP_i^k(m)$
6: for all $l \in \{1, 2, \ldots, m\}$ do
7: if $(\exists d)(\exists l): [LP_i^k(l+1) = \delta_i^l(LP_i^k(l), d)]$ but $[d \notin F_i^e(LP_i^k(l), LP_i^k(l+1)))]$
8: Find $d' \in D_i^k: LP_i^k(l+1) = \delta_i^m(LP_i^k(l), d')$ and $d' \in F_i^e(LP_i^k(l), LP_i^k(l+1))$)
9: Remove the words in $D_i^k$ that contains $d$
10: $LP_i^e = LP_i$
11: else
12: Check if there exists $l$ such that $LP_i^k(l+1) \neq \delta_i^l(LP_i^k(l), d)$ for all $d \in D$
13: if such $l$ does exist then
14: Find $v_1, v_2, \ldots, v_p : (LP_i^k(l), v_1), \ldots, (v_p, LP_i^k(l+1)) \in \rightarrow_e$
15: $LP_i^k(l)LP_i^k(l+1) = LP_i^k(l)v_1 \ldots v_pLP_i^k(l+1)$
16: Update $LP_i^e$ according to $LP_i^k$
17: Update $D_i^{mo}$ accordingly
18: end if
19: end if
20: end for
21: end for
return $LP_i^e$ and $D_i^{m}$

with the synthesized local plans $LP_i$ and corresponding door profiles $D_i^{mo}$. For $i \in I$, Algorithm 5 first enumerates (line 2) $LP_i$ as a collection of $K$ prefix-closed strings over $V \cup \Sigma_i$, i.e., $LP_i = \bigcup_{k=1}^{K}LP_i^k$, while if $LP_i^k$ admits a cycle such that $LP_i^k = uv^*$ for some $u \in \Sigma^*$ and $v \in \Sigma^* - \{\epsilon\}$, we replace $LP_i^k$ by $uv$. Next for each $LP_i^k$, we proceed to the investigation of two types of environment uncertainties. On the one hand (lines 7-10), if two adjacent regions in $LP_i$ are connected by multiple doors whereas some of the doors are closed, we require the agent to use alternative (redundant) doors to accomplish the motion plan and hence the integrated local plan $LP_i$ remains the same, while the door profile $D_i^m$ is formed by discarding all the strings in $D_i^{mo}$ that contain the symbols of closed doors. On the other hand (lines 12-17), when two consecutive regions $v$ and $v'$ that are supposed to
be visited by the agent $G_i$ are not connected by any doors in $E$, the assumption that $G_i^E$ is a trim DFA inspires us to replace the motion transition $vv'$ by a sequence of “intermediate” regions $v_1, v_2, \ldots, v_p$ such that

$$v_{i,0} \ldots v v_1 v_2 \ldots v_p v' \in \text{Run}[L(G_i^E)];$$

afterwards, we replace all the transition $vv'$ in $LP_i$ by $vv_1v_2 \ldots v_pv'$ to construct $LP_i^E$, and the door profile is updated accordingly.

Finally, we apply the $L^*_{MP}$ algorithm for the aforementioned multi-robot coordination example in order to construct local motion plans.

**Example 4.4.** We apply the $L^*_{MP}$ algorithm for the aforementioned multi-robot coordination example in order to construct local motion plan $L_i^{mo}$ ($i \in I$) for the robot $G_i$ after obtaining $L_i^{mi}$ ($i \in I$) as shown in Fig. 4.18, Fig. 4.19 and Fig. 4.20, respectively. Such construction relies on the local mission-motion integration mapping $\pi_i (i = 1, 2, 3)$, which are defined as follows, respectively:

(i) **Robot $G_1$:**

$$\pi_1(G_1 \text{to} D_1) = \pi_1(G_1 \text{on} D_1) = \pi_1(\text{Open}) = \pi_1(\text{Close}) = \pi_1(D_1 \text{open}) = \pi_1(D_1 \text{close})$$

$$= \pi_1(G_2 \text{in} R_1) = \{R_1, R_3\},$$

$$\pi_1(G_1 \text{in} R_3) = \{R_3\},$$

$$\pi_1(h_1) = \pi_1(R) = \pi_1(G_1 \text{in} R_1) = \{R_3\}.$$

(ii) **Robot $G_2$:**

$$\pi_2(h_2) = \pi_2(G_2 \text{in} R_1) = \pi_2(R) = \{R_1\},$$

$$\pi_2(D_1 \text{open}) = \{R_1, R_2\},$$

$$\pi_2(G_2 \text{in} R_2) = \{R_2\}.$$

(iii) **Robot $G_3$:**

$$\pi_3(G_1 \text{to} D_1) = \pi_3(G_1 \text{on} D_1) = \pi_3(\text{Open}) = \pi_3(\text{Close}) = \pi_3(D_1 \text{open}) = \pi_3(D_1 \text{close})$$

$$= \pi_3(G_2 \text{in} R_1) = \{R_1, R_3\},$$

$$\pi_3(G_3 \text{in} R_3) = \{R_3\},$$

$$\pi_3(h_3) = \pi_3(R) = \pi_3(G_3 \text{in} R_1) = \{R_1\}.$$
Facing the nominal environment \( \mathcal{E}_0 \) depicted in Fig. 4.2 (b), the local motion plans in this nominal environment are demonstrated in terms of the DFAs shown in Fig. 4.21.

Figure 4.21. Local motion plans \( L_i^{mo} \) \((i = 1, 2, 3)\) in the nominal environment \( \mathcal{E}_0 \).

Based on the local motion plans obtained as shown in Fig. 4.21, we can construct the integrated local plan \( LP_i \) for each robot \( G_i \). Fig. 4.22, Fig. 4.23 and Fig. 4.24 depict the integrated local plans \( LP_1, LP_3 \) and \( LP_2 \), respectively.

Figure 4.22. Integrated local plan \( LP_1 \) for the robot \( G_1 \).

Figure 4.23. Integrated local plan \( LP_3 \) for the robot \( G_3 \).
It is worth pointing out that, all the $LP_i$’s are designed with respect to $E_0$, and when implementing $LP_i$’s in the practical environment $E$, the robots may encounter with uncertainties. In this case, the robots shall automatically react to the uncertain environment by online replanning their motion plans. For example, in case the door $D_3$ is broken and cannot be opened when $G_3$ needs to return to $R_1$, $G_3$ shall then return to $R_1$ through $D_1$.

4.7 Conclusion

In this chapter, we present a learning-based approach to the formal mission and motion planning problem of cooperative multi-agent systems. Starting from an automata-based characterization of each agent’s mission and motion capabilities, our proposed framework solves the formal planning problems in a hierarchical architecture. We adopt three modified $L^*$ algorithms to synthesize the local mission supervisors, to check the joint efforts of the mission executions via compositional verification techniques, and to synthesize motion plans corresponding to the mission plans. In addition, we also present another algorithm that adaptively reconfigures the motion plans so that environment uncertainties can be resolved. An illustrative example of multi-robot coordination to achieve a request-response team mission is presented to justify the effectiveness of the proposed planning framework.
CHAPTER 5

COORDINATION AND CONTROL OF DISTRIBUTED DISCRETE EVENT SYSTEMS IN THE PRESENCE OF ACTUATOR AND SENSOR FAULTS

5.1 Introduction

In Chapter 4, we consider the synthesis of local mission and motion plans of multi-agent systems with distributed architectures within the DES formalism. Once the coordination schemes and the control policies of a multi-agent system are designed, its operational safety becomes a crucial property and it is necessary to prevent the possibly catastrophic consequences from occurring. Although the formal planning framework presented in Chapter 4 shows great adaptiveness to the inherent uncertainties from the model and/or environment, the increasingly sophisticated system architectures may also render large-scale ACPSs more vulnerable to unpredictable faults that may cause undesired consequences. It is therefore very important to figure out how to guarantee safe and reliable operations of these ACPSs with complex architectures after the occurrences of potential faults, thereby motivating us to study both the fault diagnosis and isolation (FDI) and the fault tolerant control (FTC) problems [16] for ACPSs in the presence of faults.

In this chapter, we study the FTC problem of a distributed DES that consists of multiple subsystems that are subject to potential actuator and sensor faults. We follow the previous work in Chapter 4 and assume that each subsystem of the given distributed DES can be modeled by a DFA over a local event set and is properly controlled by a nominal supervisor such that a global specification can be achieved through the coordination among the subsystems. The fault tolerance property then requires that the distributed DES satisfy
the given control specification prior to as well as after occurrences of certain faults. We propose an active approach for the FTC problem of the distributed DESs in the presence of possible loss of actuating and/or sensing capabilities. Specifically, by modeling actuator faults as loss of local controllability of certain actuator events, an active FTC architecture is developed in which the local supervisor is reconfigured to achieve a degraded but safe post-fault performance. Secondly, by characterizing sensor faults as permanent observability failure of certain sensor readings, we develop an automata-theoretic modeling framework of a controlled subsystem in the presence of various faulty sensors, upon which appropriate local supervisors are carried out corresponding to different faulty modes of the subsystem. Furthermore, we allow occurrences of multiple actuator and sensor faults, and we investigate the local FTC problem under multiple faults by introducing a novel model of switching DESs. Finally, by leveraging the idea of compositional verification developed in [36], we present an assume-guarantee post-fault coordination scheme for the distributed DESs after the synthesis of local post-fault supervisors so as to ensure the accomplishment of the global specification (with possible degradation). Compared to our previous conference publication [44] on FTC of multi-agent systems modeled as DESs, we propose different definitions of actuator and sensor faults along with novel synthesis methods of post-fault supervisors. In addition, more rigorous coordination schemes among fault-pruned subsystems are developed by accounting for the safe operations of the subsystems in the presence of faults. It turns out that the fault tolerant supervisor synthesis methods and coordination strategies proposed in this chapter will eventually accomplish the desired (possibly degraded) performance of the distributed DESs.

The remainder of this chapter is organized as follows. We present a brief review on fault diagnosis and fault tolerant control within the DES formalism in Section 5.2. The actuator and sensor faults of distributed DESs are defined in Section 5.3 and the fault tolerant coordination and control problem is formulated. We provide a fault tolerant coordination and control framework in Section 5.4 with an illustrative example. The necessary and suf-
ficient conditions for the existence of an actuator fault tolerant supervisor and guidelines to test them are derived in Section 5.5. Section 5.6 introduces a novel modeling method of a subsystem that is subject to sensor faults and establishes necessary and sufficient conditions for the existence of an sensor fault tolerant supervisor in order to present a safe diagnosis and active sensor fault tolerant control architecture. In Section 5.7, we consider the fault tolerant control problem in which a combination of various actuator and sensor faults may take place consecutively. We exploit an assume-guarantee paradigm to achieve post-fault coordination in Section 5.8. We conclude this chapter Section 5.9.

5.2 Related Work

Nowadays, as cyber-physical systems have become more and more sophisticated in structures, unpredictable faults and failures in both cyber and physical worlds also become more and more likely to cause undesired or even catastrophic consequences. Since control systems are often safety-critical (e.g., infrastructures, transportation networks, etc.), it is important to figure out how to detect and diagnose faults before the engineering systems evolve to any unsafe states and how to guarantee safe and reliable operations of these systems after the occurrences of faults. Due to the fact that sequential decision-making and control of many practical ACPSs show strong event-driven features, DESs become particularly useful for addressing the FDI and FTC problems from the viewpoint of the cyber world, and a considerable amount of research efforts has been devoted to the two types of problems [16] for engineering systems in the recent two decades. Following the pioneering work of Sampath et al. [155], many contributions have been made to the FDI problem of DESs where faults are modeled as unobservable events [201], including centralized [78, 156], decentralized [51, 139], modular [38, 160] and distributed [170] architectures. On the other hand, diagnosability of DESs where faults are modeled as visit of certain faulty states were investigated in [108], which can be viewed as a variance of state estimation problem. Depending on how a faulty state needed to be determined, various of fault
detectabilities have also been studied in recent years [166, 167].

Despite the extensive study on the FDI problem, relatively little work has been done on the FTC problem of DESs. Lafortune and Lin[98] established a framework for synthesizing supervisors of DESs to accomplish both desired and tolerable control objectives. Cho and Lim [30] investigated the synthesis of fault tolerant supervisor for automated manufacturing systems with a case study. Resilience and fault tolerance of Petri nets (P-Ns) were assured in [74] by reconfiguring PN-based supervision with uncontrollable transitions. Originated from the adaptive and robust supervisory control schemes presented in [107], respectively, the impact of potential faults can be addressed using either a passive or an active approach in the literature. The passive FTC is often achieved via robust control techniques and a unified controller is designed to ensure that the closed-loop system remains insensitive to certain faults. In contrast, the active FTC aims at achieving the control objectives by adapting the control law to the fault-pruned system behaviors once the occurrences of faults are determined by the FDI filters. By following a passive approach, Rohloff [147] derived a robust supervisor to assure fault tolerance in which a sensor fault may take place, whereas Sánchez and Montoya [157] expanded this method by taking safety enforcement into consideration. Darabi et al. [49] proposed an active FTC method by developing a control-switching theory for DESs so that the underlying system was switched to a post-fault supervisor when sensors fail. However, [49, 147, 157] only dealt with faults in sensors. By modeling a fault as an unobservable event whose occurrence causes a faulty behavior, Wen et al. [186] presented a necessary and sufficient condition for the existence of a unique fault tolerant supervisor that can enforce a nominal specification for the non-faulty plant and a tolerable specification for the overall plant, and such an approach can be deemed as a passive FTC approach. The passive FTC approach was also adapted by Karimadini and Lin [82] to cooperative tasking of multi-agent systems within the DES formalism. On the other hand, Paoli et al. [130] proposed an active FTC approach by actively reconfiguring the post-fault supervisor in response to the online di-
agnostic information presented by a fault diagnoser. Such an active approach was adopted by Shu and Lin [165] to synthesize state-feedback and state-estimate-feedback supervisors so that safe operations of a DES can be enforced in the presence of faults. Carvalho et al. [24] investigated the intrusion detection and the active FTC problem of DESs whose actuators might be compromised. To cope with supervisor faults, the problem of reliable decentralized supervisory control was addressed in [174] and [113].

Different from most of these prior works which deal with centralized supervisory control architectures, we aim at investigating the FTC problem of distributed DESs in this chapter. Furthermore, this chapter also differs from our previous work [44] that addressed the impact of a single sensor/actuator fault, and the post-fault control reconfiguration and coordination against multiple faults is also considered.

5.3 Problem Formulation

In this section, we first present models of DESs that are subject to potential sensor and actuator faults. Next, we formulate the fault tolerant coordination and control problem for distributed DESs that are composed of fault-pruned subsystems, followed by an outline of our approach for solving this problem.

5.3.1 Actuator and Sensor Faults in Discrete Event Systems

Similar to the cooperative multi-agent systems in Chapter 4, the distributed DES $G$ under consideration also consists of $n$ heterogeneous subsystems with unique identities, namely $I = \{1, 2, \ldots, n\}$. Each subsystem is modeled as an accessible DFA $G_i = (Q_i, \Sigma_i, \delta_i, q_{i,0})$ ($i \in I$). The global event set is defined as $\Sigma = \bigcup_{i \in I} \Sigma_i$ and we denote by $P_i$ the natural projection from $\Sigma^*$ to $\Sigma_i^*$. All the subsystems are coordinated synchronously, i.e., $G = \|_{i \in I} G_i$. For any $\sigma \in \Sigma$, we denote by $In(\sigma) = \{ i \in I | \sigma \in \Sigma_i \}$ the set of subsystems that take $\sigma$ as a local event. For each $i \in I$, the local event set $\Sigma_i$ is partitioned into the set of locally controllable events $\Sigma_{i,c}$ and the set of locally uncontrollable events $\Sigma_{i,uc}$. For
the purpose of distributed control, we re-state the requirement for the local controllability of shared events in (4.2) as follows.

\[(\forall i,j \in I)[(i \neq j) \Rightarrow (\Sigma_{i,uc} \cap \Sigma_{j,c} = \emptyset)].\]  
(5.1)

Furthermore, \(\Sigma_i\) is partitioned into the set of locally observable events \(\Sigma_{i,o}\) and the set of locally unobservable events \(\Sigma_{i,uo}\), with \(P_{i,o}\) being the natural projection from \(\Sigma^*_i\) to \(\Sigma^*_{i,o}\).

We are interested in the faults whose occurrences may interfere with the nominal functionality of the actuators and/or sensors of the subsystems. In particular, we assume that a local supervisor \(S_i\) implements the control decisions on the subsystem \(G_i\ (i \in I)\) via \(K_i\) local actuators in the nominal mode:

\[\Sigma_{i,a} = \{\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,K_i}\} \subseteq \Sigma_{i,c} \cap \Sigma_{i,o}.\]  
(5.2)

Since an actuator event represents the execution of a certain control action, it is reasonable to assume the actuator event to be both controllable and observable. Furthermore, we assume that actuator events are not shared by various subsystems, i.e., for any \(\eta_{i,m} \in \Sigma_{i,a}\) \((m \in \{1, 2, \ldots, K_i\})\). The actuators in \(\Sigma_{i,a}\) are supposed to be vulnerable to malfunctions. Formally, we define \(h_{i,m}\) as an actuator fault event corresponding to the case in which a local supervisor loses the local controllability of the actuator event \(\eta_{i,m}\).

**Definition 5.1 (Actuator Faults).** For \(m \in \{1, 2, \ldots, K_i\}\), an actuator fault \(h_{i,m}\) occurred in the subsystem \(G_i\ (i \in I)\) indicates that the actuator event \(\eta_{i,m} \in \Sigma_{i,a}\) becomes locally uncontrollable for \(G_i\). The set of possible actuator fault events in \(G_i\) is represented by the set \(\Sigma^F_{i,a} = \{h_{i,1}, h_{i,2}, \ldots, h_{i,K_i}\}\).

According to Definition 5.1, an undesired control action may not be prohibited by a local supervisor as a consequence of an actuator fault; hence Definition 5.1 is consistent with the general understanding of actuator faults [25]. In this chapter, we assume that the
loss of the local controllability of an actuator event is permanent and cannot be recovered after the fault takes place.

On the other hand, a local supervisor $S_i$ ($i \in I$) persistently monitors and controls the evolution of $G_i$ by receiving the local sensor readings from the set

$$
\Sigma_{i,s} = \{\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,N_i}\} \subseteq \Sigma_{i,o} \cap \Sigma_{i,uc}.
$$

(5.3)

We assume that the local observability of the sensor readings contained in $\Sigma_{i,s}$ is suspicious of loss [23, 157]; such a circumstance may correspond to the malfunction of the sensors that monitor and record the occurrences of the event. Other than actuators, it is reasonable to assume a sensor reading to be locally uncontrollable since the local supervisor shall not prevent a sensor reading from being received. For all $k \in \{1, 2, \ldots, N_i\}$, we use a sensor fault event $f_{i,k}$ to capture the circumstance that the sensor reading $\sigma_{i,k} \in \Sigma_{i,s}$ fails to be obtained by $S_i$.

**Definition 5.2 (Sensor Faults).** For $k \in \{1, 2, \ldots, N_i\}$, by a sensor fault $f_{i,k}$ that occurs in $G_i$ ($i \in I$) we mean the local observability of the sensor reading $\sigma_{i,k} \in \Sigma_{i,s}$ is lost. The set of possible sensor fault events in $G_i$ is collected in the set $\Sigma_{i,s}^F = \{f_{i,1}, f_{i,2}, \ldots, f_{i,N_i}\}$.

We also assume that observability loss of a subsystem’s sensor reading is permanent; that is, once a sensor reading fails to be received by the local supervisor, it cannot be repaired. For convenience of presentation, we attach a fault label $f$ to distinguish a faulty sensor reading from a nominal one after the fault takes place. Thus we can define the following set of faulty sensor readings:

$$
\Sigma_{i,s}^f := \{\sigma_{i,k}^f | \sigma_{i,k} \in \Sigma_{i,s}\}.
$$

(5.4)

Note that any $\sigma_{i,k}^f \in \Sigma_{i,s}^f$ is locally unobservable.
5.3.2 Coordination and Control in the Presence of Faults

In this chapter, we are concerned with developing fault tolerant coordination and control strategies for a distributed DES so that its safe operation can be guaranteed in spite of possible loss of actuating and/or sensing capabilities. To this end, we associate with each subsystem \(G_i\) \((i \in I)\) with a non-empty and prefix-closed safety language \(L_{\text{safe}}^i \subseteq L(G_i)\), which includes all the tolerable behaviors of \(G_i\) that should be satisfied (in the sense of Definition 2.5) in both nominal and post-fault operations. Furthermore, the safety requirements for the overall system \(G\) is given by

\[
L_{\text{safe}} := (\|_{i \in I} L_{\text{safe}}^i) \cap L(G). \tag{5.5}
\]

In addition to safety, it is also desired that \(G\) can fulfill a non-empty and prefix-closed control specification \(L \subseteq L_{\text{safe}}\) through coordination among the subsystems. Since it has been proved in Chapter 4 and in the literature [77] that under the assumption (5.1), one can synthesize a local supervisor \(S_i\) \((i \in I)\) for \(G_i\) such that \(L(S_i\|G_i) = L_i \subseteq L_{\text{safe}}^i\) and \(\|_{i \in I} L_i \subseteq L\). We hence assume that each subsystem has already been equipped with a nominal supervisor \(S_i\) \((i \in I)\) in order to achieve the global specification while prohibiting any undesired behaviors from occurring when no faults occur. Nevertheless, it is worth pointing out that when faults take place, the controlled DES may still generate strings that violate the specification due to execution of effective control actions issued from the nominal local supervisor on the faulty subsystem.

Based on the aforementioned notations and models, we formulate the following coordination and control problems of a distributed DES in the presence of potential actuator and/or sensor faults.

**Problem 5.1.** Consider a controlled distributed DES \(G\) that consists of \(n\) subsystems, where each subsystem \(G_i\) is controlled by a nominal supervisor \(S_i\) \((i \in I)\) so as to satisfy both the local safety \(L_{\text{safe}}^i\) and the prefix-closed global specification \(L\). For each subsystem
with the set of actuators $\Sigma_{i,a}$ and the set of sensor readings $\Sigma_{i,s}$, find a post-fault supervisor $S^F_i$ after the occurrences of any faults such that:

(i) Safe fault detection: for any $i \in I$, the occurrences of any actuator and/or sensor faults should be detected before the closed-loop subsystem $S_i \| G_i$ generates any unsafe behaviors;

(ii) Fault tolerant control for local safety: the local safety requirements shall always be satisfied despite the occurrences of faults, i.e., $S^F_i \| G^F_i \models L^afe_i$, where $G^F_i$ denotes the fault-pruned subsystem;

(iii) Post-fault coordination: for any subset $I' \subseteq 2^I$ of subsystems whose nominal operations are influenced by faults, the joint effort of the post-fault supervisors $S^F_i (i \in I')$ will at least guarantee the global safety, i.e., either $(\|_{i \in I-I'} S_i \| G_i) \| (\|_{j \in I'} S^F_j \| G^F_j) \models L$ or $(\|_{i \in I-I'} S_i \| G_i) \| (\|_{j \in I'} S^F_j \| G^F_j) \models L^afe$ must be satisfied.

5.4 The Fault Tolerant Coordination and Control Framework

In this section, we present our approach to solve Problem 5.1 and present a multi-robot coordination example to illustrate the proposed planning framework.

5.4.1 The Active Fault Tolerant Coordination and Control Framework

A hierarchical fault tolerant coordination and control framework is proposed in Fig. 5.1 to solve Problem 5.1. Starting from the distributed DES $G = \|_{i \in I} G_i$ equipped with local supervisors $S_i (i \in I)$ such that $\|_{i \in I} S_i \| G_i \models L$, the performance objectives presented in Problem 5.1 are solved via the joint effort of safe diagnosis, active fault tolerant control reconfiguration of the fault-pruned subsystem(s) as well as post-fault coordination between the post-fault and nominal subsystems.

(i) Objective (i) of Problem 5.1 is achieved by the safe detection of the subsystem(s) in the presence of potential faults. More specifically, the property of safe detectability guarantees that potential occurrences of certain actuator and sensor faults can be detected before the fault-pruned subsystem generates any unsafe behaviors.

(ii) Once actuator and/or sensor faults are detected in the nominal mode of a subsystem, Objective (ii) of Problem 5.1 is attained by employing the active fault tolerant control reconfiguration. We first construct DFA models of the subsystem after the
detection of the faults, based on which appropriate post-fault supervisors are synthesized to steer the fault-pruned subsystem to meet new (possibly degraded) post-fault local specifications while obeying the local safety requirements.

Figure 5.1. The fault tolerant coordination and control framework.

(iii) Objective (iii) will be attained by leveraging the idea of assume-guarantee post-fault coordination (cf. Section 5.8). By exploiting an assume-guarantee paradigm, the post-fault supervisors are refined whenever necessary to jointly satisfy the global (safety) specification.

5.4.2 A Multi-robot Coordination Example

Throughout the rest of this chapter, we demonstrate the effectiveness of the proposed fault tolerant coordination and control framework in Fig. 5.1 via a more comprehensive example in multi-robot coordination.

Let us re-consider the multi-robot system with three robots $G_1, G_2$ and $G_3$ that are pre-
presented in Section 4.4.2. As shown in Fig. 5.2, all the robots initially stay in Room 1. Room 2 and Room 1 are connected by the one-way door $D_2$ and the two-way door $D_1$, while Room 3 and Room 1 are connected by $D_1$ and another two-way door $D_3$. $D_1$ is heavy and should be opened by two robots cooperatively. All doors shall close automatically unless there is an external force to keep them open and all the robots have identical capability of opening the doors. In this chapter, we go beyond the multi-robot system presented in Section 4.4.2 and investigate the safe detection, post-fault control reconfiguration and coordination of the robots in the presence of faults. Instead of Table 4.1, the events that are associated to the robots are re-defined in Table 5.1 in this example.

The local event set $\Sigma_i$ ($i = 1, 2, 3$) for robot $G_i$ is defined as follows:

$$\Sigma_i = \{h_i, G_i to D_1, G_i on D_1, OP, C L, G_2 in R_1, G_i to R_3, G_i in R_3, D_1 closed, D_1 open, G_i to R_1, G_i in R_1\}, i = 1, 3;$$

$$\Sigma_i = \{h_2, G_2 to R_2, G_2 in R_2, D_1 open, G_2 to R_1, G_2 in R_1\}, i = 2.$$ 

It is reasonable to assume $G_2 in R_1 \in \Sigma_1 \cap \Sigma_3$ since it can be viewed as the information that is transmitted among $G_1$, $G_2$ and $G_3$ via communication. All the events are assumed to be locally observable in the nominal mode. The set of each robot’s sensor
TABLE 5.1

EVENTS OF THE MULTI-ROBOT SYSTEM

<table>
<thead>
<tr>
<th>Event</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_i$</td>
<td>Robot $G_i$ receives the service request, $i = 1, 2, 3$.</td>
</tr>
<tr>
<td>$G_i to D_1$</td>
<td>Robot $G_i$ approaches the door $D_1$, $i = 1, 3$.</td>
</tr>
<tr>
<td>$G_i on D_1$</td>
<td>Robot $G_i$ at the door $D_1$, $i = 1, 3$.</td>
</tr>
<tr>
<td>$G_i to R_k$</td>
<td>Robot $G_i$ heads for Room $k$, $k = 1, 2, 3$.</td>
</tr>
<tr>
<td>$G_i in R_k$</td>
<td>Robot $G_i$ stays at Room $k$, $k = 1, 2, 3$.</td>
</tr>
<tr>
<td>$Open$</td>
<td>command for moving forward to open $D_1$.</td>
</tr>
<tr>
<td>$Close$</td>
<td>command for moving backward to close $D_1$.</td>
</tr>
<tr>
<td>$D_1 open$</td>
<td>$D_1$ is open.</td>
</tr>
<tr>
<td>$D_1 closed$</td>
<td>$D_1$ is closed.</td>
</tr>
</tbody>
</table>

readings is given by $\Sigma_{i,s} := \{h_i, G_2 in R_1, G_i in R_3, G_i in R_1\}$ for $i = 1, 3$ and $\Sigma_{2,s} := \{h_2, G_2 in R_2, D_1 open, G_2 in R_1\}$. On the other hand, the set of actuators for each robot is given by $\Sigma_{i,a} = \Sigma_{i,c} := \{G_i to D_1, G_i on D_1, OP, CL, G_i to R_3, G_i to R_1\}$ for $i = 1, 3$ and $\Sigma_{2,a} = \Sigma_{2,c} := \{G_2 to R_2, G_2 to R_1\}$, respectively.

Starting from Room 1, $G_2$ can enter Room 2 through $D_2$ and can also move to Room 3 through $D_3$. When $D_1$ is open by the other two robots, $G_2$ can move to both Rooms 2 and 3 through $D_1$. In this example, we only consider the possible behaviors of $G_2$ between Rooms 1 to Room 2 and the corresponding model $G_2$ is depicted in Fig. 5.3. Similarly, the DFA model that characterizes the behaviors of the robot $G_i$ ($i = 1, 3$) between Room 1 and 3 are shown in Fig. 5.4. The model of the distributed multi-robot system is then obtained as $G = G_1 \parallel G_2 \parallel G_3$.

In this example, we require that $G_2$ should move between Room 1 and Room 2, while
the other two robots should move between Room 1 and Room 3, therefore, we can write that $L_{safe}^{safe} = L(G_i) \ (i \in I)$. The goal for the multi-robot team is that $G_2$ must respond promptly to a fire alarm in Room 2 by entering the room through $D_2$ and then returning to Room 1. Since $D_2$ is a one-way door and cannot open in Room 2, $G_1$ and $G_3$ need to open $D_1$ jointly so that $G_2$ can successfully return. After that, $G_1$ and $G_3$ should close $D_1$ and return to Room 1 as well. Such a global specification $L$ consists of the following local specifications $L_{i}^{spe} (i \in \{1, 2, 3\})$ for $G_i$.

$$L_{1}^{spe} = h_1 G_1 to D_1 G_1 on D_1 OP D_1 open G_2 in R_1 CL D_1 closed;$$

$$L_{2}^{spe} = h_2 G_2 to R_2 G_2 in R_2 D_1 open G_2 to R_1 G_2 in R_1;$$

$$L_{3}^{spe} = h_3 G_3 to R_3 G_3 in R_3 G_3 to D_1 G_3 on D_1 OP D_1 open G_2 in R_1 \overline{CL D_1 closed G_3 to R_1 G_3 in R_1}$$
The global specification $L$ is then given by $L = L_{1}^{spe} \parallel L_{2}^{spe} \parallel L_{3}^{spe}$. To satisfy $L$ jointly, on the one hand, robot $G_1$ stays in Room 1 while $G_3$ goes to Room 3 in order to open $D_1$. On the other hand, $G_2$ should enter Room 2 to respond to the fire alarm and then return to Room 1 as long as $D_1$ is open. The nominal supervisors $S_1$, $S_2$ and $S_3$ can then be synthesized by applying the $L_c^*$ algorithm for the corresponding specifications. The supervisors are illustrated in Fig. 5.5, Fig. 5.6 and Fig. 5.7, respectively.

![Figure 5.5. The nominal supervisor $S_1$.](image)

![Figure 5.6. The nominal supervisor $S_2$.](image)

![Figure 5.7. The nominal supervisor $S_3$.](image)
It is worth pointing out that Fig. 5.5, Fig. 5.6 and Fig. 5.7 also demonstrate the controlled subsystems $G_1^0$, $G_2^0$ and $G_3^0$ in the nominal mode, respectively; in other words, $L_i = L(S_i||G_i) = L(S_i)$.

In the following sections, we will study the local fault tolerant control reconfiguration and post-fault coordination of the aforementioned multi-robot system in the presence of potential actuator and/or sensor faults.

5.5 An Active Approach to Actuator Fault Tolerant Control

From Definition 5.1, occurrence of an actuator fault results in unexpected loss of (local) controllability of the corresponding actuator event; in other words, the controlled subsystem $S_i||G_i$ may deviate from the presumed nominal behaviors $L_i := L(S_i||G_i)$ in a faulty mode, since the local supervisor $S_i$ can no longer disable a faulty actuator event so as to prevent an undesired string from being generated. As a consequence, behaviors of the fault-pruned subsystem may even jeopardize the accomplishment of the global specification $L$. In this section, by assuming that actuator fault events are locally observable, we address the actuator fault tolerant control problem via an active approach [16].

5.5.1 Safe Detection of Actuator Faults

We start our investigation of the active control reconfiguration approach from the detection of an actuator fault. Formally, we define

$$\forall m \in \{1, 2, \ldots, K_i\} : \Psi_i(\eta_{i,m}) = \{ s\eta_{i,m} \in L(G_i) | \eta_{i,m} \in \Sigma_{i,a} \}$$

as the set of local strings generated by $G_i$ ($i \in I$) whose last event is $\eta_{i,m}$; furthermore, we denote by $\Psi_i(\Sigma_{i,a}) = \bigcup_{\eta_{i,m} \in \Sigma_{i,a}} \Psi_i(\eta_{i,m})$ as the union of all the strings that end in an actuator event. For convenience of presentation, we assume that an actuator $\eta_{i,m} \in \Sigma_{i,a}$ ($m = 1, 2, \ldots, K_i$) becomes locally uncontrollable with respect to the subsystem $G_i$ ($i \in$
I) when a string \( t_{i,0} \in L(G_i) \) is generated. Since it is always required that the actuator fault be detected before the (controlled) subsystem generates any unsafe behaviors, we introduce the following notion of \( \text{AF-safe detectability}^1 \) as a property that the string \( t_{i,0} \) should satisfy.

**Definition 5.3 (AF-safe Detectability).** For any \( m \in \{1, 2, \ldots, K_i\} \), the actuator fault event \( h_{i,m} \) corresponding to the loss of local controllability of \( \eta_{i,m} \) is AF-safe detectable in a string \( t_{i,0} \in L(G_i) \) if

\[
[t_{i,0} \in \Psi_i(\eta_{i,m}) - L_i] \land [t_{i,0} \subseteq L_i^{safe}].
\]

(5.7)

Furthermore, we say \( \Sigma_{i,a} \) is AF-safe detectable in the string \( t_{i,0} \) if it is AF-safe detectable for all \( \eta_{i,m} \in \Sigma_{i,a} \), i.e.,

\[
[t_{i,0} \in \Psi_i(\Sigma_{i,a}) - L_i] \land [t_{i,0} \subseteq L_i^{safe}].
\]

(5.8)

**Remark 5.1.** Since local observability of all the actuator events holds both before and after the occurrence of an actuator fault, the intuition behind Definition 5.3 hence implies that an actuator event fault cannot be detected until the nominal supervisor becomes incapable of disabling a faulty actuator event and generates a behavior that violates the presumed local performance. Furthermore, the detection of the actuator fault should always be accomplished before the subsystem violates the local safety requirements.

5.5.2 Construction of the Post-fault Subsystem

Now we consider the case that an actuator event \( \eta_{i,m} \in \Sigma_{i,a} \) \((i \in I)\) becomes faulty when the controlled subsystem \( S_i \parallel G_i \) generates a string \( t_{i,0} \in L(G_i) \). If \( \eta_{i,m} \) is AF-safe detectable in the string \( t_{i,0} \), the corresponding actuator fault event \( h_{i,m} \) is instantaneously

---

^1Here we refer to “AF” as “actuator faults”. 

141
released for fault detection and disable the operation of the nominal supervisor so that no unsafe strings will be generated. Under this circumstance, the possible successive behaviors of the subsystem after the detection of an actuator fault event are characterized in terms of the following “suffix automaton”.

**Definition 5.4 (Suffix Automaton).** The suffix automaton of an accessible DFA $G = (Q, \Sigma, \delta, q_0)$ following a string $t \in L(G)$ is another DFA

$$G^{suft}(t) := (Q^{suft}(t), \Sigma, \delta^{suft}, q_0^{suft}(t)) = \text{Ac}(Q, \Sigma, \delta(q_0, t)),$$

(5.9)

where Ac denotes the operator of computing the accessible part of a given DFA.

It can be verified that the suffix automaton $G^{suft}(t)$ preserves all the possible successive behaviors of the DES $G$ after the string $t$, i.e., $L(G^{suft}(t)) = L(G) \setminus t$.

With the help of Definition 5.4, we can now model the uncontrolled post-fault subsystem. Formally, the post-fault model $G^{m,a}_i(t_{i,0})$ of the subsystem $G_i$ ($i \in I$) after the generation of $t_{i,0}$ and the detection of $h_{i,m}$ is defined as

$$G^{m,a}_i(t_{i,0}) := (Q^{m,a}_i, \Sigma^{m,a}_i, \delta^{m,a}_i, q^{m,a}_i(0)),$$

(5.10)

where

$$Q^{m,a}_i = \{q^{m,a}_{i,0}\} \cup Q^{suft}_i(t_{i,0}),$$

$$\Sigma^{m,a}_i = \{h_{i,m}\} \cup \Sigma_i,$$

$$\delta^{m,a}_i = \{(q^{m,a}_{i,0}, h_{i,m}, q^{suft}_{i,0}(t_{i,0}))\} \cup \delta^{suft}_i;$$

(5.11)

and $(Q^{suft}_i(t_{i,0}), \Sigma, \delta^{suft}_i, q^{suft}_{i,0}(t_{i,0}))$ forms the suffix automaton $G^{suft}_i(t_{i,0})$ of $G_i$. It can then be inferred from (5.11) that $L(G^{m,a}_i) = h_{i,m}(L(G_i) \setminus t_{i,0})$.

Different from $G_i$, the local controllability and observability status of the post-fault
events in $\Sigma_{i}^{m,a}$ is given by

$$\Sigma_{i,c}^{m,a} = \Sigma_{i,c} - \{\eta_{i,m}\}, \Sigma_{i,uc}^{m,a} = \Sigma_{i,uc} \cup \{\eta_{i,m}, h_{i,m}\},$$  \hspace{1cm} \text{(5.12)}

and

$$\Sigma_{i,o}^{m,a} = \Sigma_{i,o} \cup \{h_{i,m}\}, \Sigma_{i,uo}^{m,a} = \Sigma_{i,uo},$$  \hspace{1cm} \text{(5.13)}

respectively. Note that from (5.12) and (5.13), the actuator fault event is locally observable but not locally controllable. We use the following example to illustrate AF-safe detectability and the construction of the post-fault model of a given subsystem.

**Example 5.1.** Consider a subsystem $G_i$ for some $i \in I$ with $Q_i = \{q_{i,0}, q_{i,1}, q_{i,2}, q_{i,3}\}$ and $\Sigma_{i,c} = \Sigma_{i,a} = \Sigma_i = \{\eta_{i,1}, \eta_{i,2}, \eta_{i,3}, \eta_{i,4}\}$. As shown in Fig. 5.8, $L(G_i) = (\eta_{i,1}\eta_{i,2})^{*}\eta_{i,3}\eta_{i,2}\eta_{i,4}^{*}$.

![Figure 5.8. The subsystem $G_i$.](image)

The state $q_{i,3}$ (marked with a red label) is an unsafe state and should not be visited; thus, $L_i^{safe} = (\eta_{i,1}\eta_{i,2})^{*}\eta_{i,3}$. We assume that the local control objective for the subsystem $G_i$ is to obey the safety requirements. Therefore, the nominal supervisor $S_i$ can be obtained as shown in Fig. 5.9, which implies that $L_i = L_i^{safe} = (\eta_{i,1}\eta_{i,2})^{*}\eta_{i,3}$.

![Figure 5.9. The nominal supervisor $S_i$.](image)
If the actuator event $\eta_{i,2}$ becomes faulty during the evolution of $S_i \parallel G_i$. Such an actuator fault can only be detected when a string that deviates from $L_i$ is generated. In this example, we assume that the string $t_{i,0} = \eta_{i,1}\eta_{i,2}\eta_{i,3}\eta_{i,2}$ indicates the occurrence of the actuator fault. Note that $t_{i,0} \in \Psi_i(\eta_{i,2})$. The actuator fault event $h_{i,2}$ is issued immediately after the generation of Hence, the post-fault subsystem $G_i^{2,a}$ can be constructed according to (5.10). From (5.11), the set of states of $G_i^{2,a}$ is obtained as $Q_i^{2,a} = \{q_{i,0}^{2,a}, q_{i,1}^{2,a}\}$, and the DFA representation of $G_i^{2,a}$ is therefore illustrated in Fig. 5.10.

![Figure 5.10. The post-fault subsystem $G_i^{2,a}(t_{i,0})$.](image)

It is worth pointing out that the string $t_{i,0} \notin L_i^{safe}$ in this example. Thus from Definition 5.3, we can conclude that $h_{i,2}$ is not AF-safe detectable in $t_{i,0}$. In fact, from (5.12) and Fig 5.10, the post-fault subsystem will always evolve to the unsafe state $q_{i,3}$ after $t_{i,0}$.

5.5.3 The Active Actuator Fault Tolerant Control Architecture

In this subsection, we aim at actively mitigating the influence of a faulty actuator. To this end, we propose a supervisor-switching architecture in Fig. 5.11 to achieve active fault tolerant control of the subsystem $G_i$ ($i \in I$) in the presence of an faulty actuator $\eta_{i,m}$.
Figure 5.11. The active fault tolerant control architecture of $G_i$ against the actuator fault $h_{i,m}$.

When no fault takes place, the control loop is closed on the nominal supervisor $S_i$ that enables appropriate locally controllable events based on the observation projection $P_{i,o}$. Once the actuator $\eta_{i,m}$ becomes faulty when a string $t_{i,0} \in L(G_i)$ is generated, we first justify the AF-safe detectability of the corresponding fault event $h_{i,m} \in \Sigma_{i,a}^F$. If the actuator fault event $h_{i,m}$ is AF-safe detectable in the string $t_{i,0}$, $h_{i,m}$ is issued to interrupt the operation of $S_i$ to prevent any unsafe strings from emerging in $G_i$ and the subsystem switches to the $m$-th ($m \in \{1, 2, \ldots, K_i\}$) faulty mode.

After the detection of $h_{i,m}$, a set of requirements are posed on the controlled behaviors of the post-fault uncontrolled subsystem $G_{i,a}^{m,a}$ (5.10), resulting in a degraded post-fault specification $L_{i}^{post}$, which is specified in the form of a non-empty and prefix-closed sub-language of $L(G_{i,a}^{m,a})$. Here we require that a post-fault supervisor $S_{i,a}^{m,a}$ be synthesized to ensure the local safety of the fault-pruned subsystem $G_{i,a}^{m,a}$. In other words, the post-fault specification for $G_{i,a}^{m,a}$ is given by:

$$L_{i}^{post}(t_{i,0}) = h_{i,m}(L_{i, safe} \setminus t_{i,0}).$$  \hspace{1cm} (5.14)

It then follows from Definition 5.4 and the prefix-closeness of $L_{i, safe}$ that $L_{i}^{post}(t_{i,0})$ in
(5.14) is a non-empty and prefix-closed sublanguage of \(L(G_i^{m,a})\). Recall Definition 2.5, a post-fault supervisor \(S_i^{m,a}\) that guarantees the specification \(L_{i}^{\text{post}}(t_{i,0})\) after \(t_{i,0}\) shall also result in a satisfaction of the local safety requirement \(L_{i}^{\text{saf}}\).

We present the following notion of actuator fault tolerance of the post-fault specification \(L_{i}^{\text{post}}(t_{i,0})\) to determine whether or not local safety can be assured by a the post-fault subsystem \(G_i^{m,a}\).

**Definition 5.5** (Actuator Fault Tolerance). Language \(L(G_i^{m,a}) (i \in I)\) is said to be actuator fault tolerant with respect to the actuator fault \(h_{i,m} \in \Sigma_{i,a}\) and the string \(t_{i,0} \in L(G_i)\) if

(i) \(h_{i,m}\) is AF-safe detectable in \(t_{i,0}\);

(ii) there exists a non-empty sublanguage of \(L_{i}^{\text{post}}(t_{i,0})\) that is both controllable with respect to \(G_i^{m,a}\) and \(\Sigma_{i,uc}\) and observable with respect to \(G_i^{m,a}\) and \(\Sigma_{i,o}\).

The actuator fault tolerance actually provides us with the necessary and sufficient condition for the existence of a satisfactory post-fault supervisor. Recall the control architecture shown in Fig. 5.11, we need to test the actuator fault tolerance of the post-fault subsystem \(G_i^{m,a}\) and if the actuator fault tolerance holds, there must exist a post-fault supervisor \(S_i^{m,a}\) that steers \(G_i^{m,a}\) to fulfill the post-fault specification \(L_{i}^{\text{post}}(t_{i,0})\) (5.14).

We present the following theorem to test the property of actuator fault tolerance of \(L(G_i^{m,a}(t_{i,0}))\) in terms of the standard \(\inf C(\cdot)\) operator (cf. Section 3.2.1) after the detection of an actuator fault.

**Theorem 5.1.** Suppose that the actuator fault event \(h_{i,m}\) is AF-safe detectable in the string \(t_{i,0} \in L(G_i)\), then the language \(L(G_i^{m,a})\) is actuator fault tolerant with respect to \(h_{i,m}\) and \(t_{i,0}\) if and only if language \(\inf C_i^{m,a}(\{\epsilon\})\), computed with respect to \(L(G_i^{m,a})\) and \(\Sigma_{i,uc}\), satisfies that \(\emptyset \neq \inf C_i^{m,a}(\{\epsilon\}) \subseteq L_{i}^{\text{post}}(t_{i,0})\).

**Proof.** According to (5.12), \(\{h_{i,m}\} \in \inf C_i^{m,a}(\{\epsilon\})\) must hold and thus \(\inf C_i^{m,a}(\{\epsilon\})\) is non-empty. Furthermore, according to (3.3), \(\inf C_i^{m,a}(\{\epsilon\})\) computed with respect to
\(L(G_i^{m,a})\) turns out to be

\[
\{\epsilon\}^{*}_{i,uc} \cap L(G_i^{m,a}) = \Sigma^{*}_{i,uc} \cap L(G_i^{m,a}).
\]

In other words, \(\inf C^{m,a}_i(\{\epsilon\})\) contains all the shortest and feasible continuations of \(t_{i,0}\) in \(L(G_i^{m,a})\) after the occurrence of \(h_{i,m}\) in which the post-fault subsystem can be controlled; that is, it includes all the possible evolutions after the system switches to \(G_i^{m,a}\) by disabling all the events that can be feasibly disabled in the faulty mode. Suppose that there exists a string \(s_{i,0} := h_{i,m}s_{i,0}' \in \inf C^{m,a}_i(\{\epsilon\})\) but \(s_{i,0} \not\in L^{post}_i(t_{i,0})\). In this case, \(s_{i,0}' \in \Sigma^{*}_{i,c}\) but \(s_{i,0}' \not\in L_i^{safe} \cap t_{i,0}\). That is, there is no way to prevent the string \(s_{i,0}'\) from emerging in \(G_i^{m,a}\) after the occurrence of \(h_{i,m}\). Thus, it follows immediately from (5.14) that \(t_{i,0}s_{i,0}' \not\in L_i^{safe}\), which is equivalent to \(t_{i,0}s_{i,0}' = L_i^{safe}\) and the actuator fault tolerance of \(L(G_i^{m,a})\) is violated.

Conversely, suppose that the actuator fault tolerance of \(L(G_i^{m,a})\) fails to be satisfied with respect to \(h_{i,m}\) and \(t_{i,0}\) even if \(h_{i,m}\) is AF-safe detectable in \(t_{i,0}\), which implies that there does not exist a post-fault supervisor \(S_i^{m,a}\) capable of steering \(G_i^{m,a}\) to satisfy \(L_i^{post}(t_{i,0})\). In this case, there always exists at least one string \(s_{i,0} \in L(G_i^{m,a}) - L_i^{post}(t_{i,0})\) that is formed by uncontrollable events in \(\Sigma^{m,a}_{i,uc}\) (otherwise the occurrence of \(s_{i,0}\) can be prohibited \(a\ priori\) by simply disabling a locally controllable event in the string). Since \(\inf C^{m,a}_i(\{\epsilon\})\) enumerates all the concatenations of locally uncontrollable events feasible in \(L(G_i^{m,a})\), it holds that \(s_{i,0} \in \inf C_i^{m,a}(\{\epsilon\})\); therefore \(\inf C_i^{m,a}(\{\epsilon\}) \not\subseteq L_i^{post}(t_{i,0})\) is satisfied, which leads to a contradiction.

Intuitively, Theorem 5.1 presents the necessary and sufficient condition for the existence of a post-fault supervisor that guarantees local safety after the detection of the fault. In fact, a local supervisor that achieves \(\inf C_i^{m,a}(\{\epsilon\})\) suffices to be a post-fault supervisor; however, it may not yield a satisfactory solution as \(\inf C_i^{m,a}(\{\epsilon\})\) may be too restrictive. To efficiently compute a satisfactory \(S_i^{m,a}\), we adopt online control techniques for the syn-
thesis of $S_i^{m,a}$ (see, e.g., [70, 72]), which generally possess polynomial complexity at each locally observable event along a trajectory in $L_i^{\text{post}}(t_{i,0})$.

We now revisit the multi-robot system in Section 5.4.2 when an actuator fault may take place.

**Example 5.2.** We consider the case in which after receiving the task request $h_3$, $G_3$ is unable to go to open $D_1$ in Room 3 and therefore cannot cooperate with robot $G_1$. Such a circumstance may correspond to the malfunction of $G_3$’s capability of opening the door $D_3$, and we model this fault by the loss of local controllability of the actuator event $G_3\text{to}D_1$. Note that according to Fig. 5.7, the actuator event $G_3\text{to}D_1$ is not supposed to occur before the event $G_3\text{to}R_3$ in the nominal mode, whereas the occurrence of this fault can be detected after a string $t_{3,0} = h_3 G_3 \text{to} D_1 \in \Psi_3(G_3\text{to}D_1)$ is generated. Since $t_{3,0} \notin L_3$ but $\overline{t_{3,0}} \subseteq L_1^{3\text{safe}}$, the condition of Definition 5.3 is satisfied and the actuator fault event $h_{3,G_3\text{to}R_3}$ is hence AF-safe detectable. The post-fault model $G_3^{G_3\text{to}D_1,a}$ of $G_3$ is given in Fig. 5.12.

![Diagram of the post-fault model of the robot $G_3$.](image)

**Figure 5.12.** The post-fault model of the robot $G_3$.

Next, the actuator fault tolerant architecture proposed in Section 5.5 is utilized at this point to synthesize the post-fault supervisor $S_{G_3^{G_3\text{to}D_1,a}}$. From (5.14), the post-fault specification is given by

$$L_3^{\text{post}}(t_{3,0}) = \overline{h_{3,G_3\text{to}D_1}(L_3^{\text{safe}} \setminus t_{3,0})} = \overline{h_{3,G_3\text{to}D_1}(L(G_3) \setminus h_3 G_3 \text{to} D_1)}.$$
Since we can write that

$$\inf C_{G^3 \to D_1, a}^3(\{\epsilon\}) = \overline{h_{G^3 \to D_1}} \subseteq L_{G^3}^{post}(t_{G^3, 0}),$$

therefore Theorem 5.1 implies that the actuator fault tolerance of $L(G^3_{G^3 \to D_1, a})$ is satisfied; that is, there exists a satisfactory post-fault supervisor $S_{G^3 \to D_1, a}^3$. We then compute $S_{G^3 \to D_1, a}^3$ as the DFA shown in Fig. 5.13 in order to enforce the local safety requirement $L_{G^3}^{safe} = L(G^3)$ in the faulty mode, which turns out to be identical to $G^3_{G^3 \to D_1, a}$.

5.5.4 Fault Tolerant Control with Multiple Actuator Faults

In the previous subsection, we discussed local fault tolerant control under the assumption that there are no consecutive occurrences of multiple actuator faults. However, multiple faults may occur in many practical engineering systems. Therefore, it is necessary to extend the proposed active fault tolerant control architecture to address consecutive occurrences of multiple actuator faults. Towards this end, the subsystem may switch from either the nominal mode to a faulty mode, or one faulty mode to another. We use $h_{G_i}^{m_1, m_2}$ ($i \in I$) to denote the switch from the $m_1$-th mode to the $m_2$-th of the subsystem $G_i$, where the 0-th mode is deemed as the nominal mode. The set of these mode-switching events associated
with \( G_i \) is given as

\[
\Sigma_{i,a}^{sw} = \{ h_i^{m_1,m_2} | m_1, m_2 \in \{0, 1, 2, \ldots, K_i\},
\]
\[
m_1 \neq m_2, m_2 \neq 0 \},
\]

where \( h_i^{m_1,m_2} \) is generated when \( \eta_i,m_2 \) becomes faulty after loss of local controllability of \( \eta_i,m_1 \). To pursue unified notations, we write \( h_i^{0,m} = h_i,m \). It is reasonable to set \( m_2 \neq 0 \) in (5.15) since we assume all the faults are permanent and the subsystem cannot return to the nominal mode from a faulty one. Similar to the actuator fault events, all the mode-switching events in \( \Sigma_{i,a}^{sw} \) are assumed to be locally uncontrollable but observable.

We are interested in the case where the actuator faults may occur consecutively. Two approaches are developed at this point to solve the actuator fault tolerant control problem with multiple faults.

**The Single-supervisor Approach** We first propose a single-supervisor fault tolerant control approach which employs a unified post-fault supervisor to address the multiple actuator faults. Since the actuator faults may take place in arbitrary orders, we define the following sets of locally controllable and observable events in the presence of possible faulty actuators, respectively:

\[
\Sigma_{i,c}^{F,a} = \Sigma_{i,c} - \bigcup_{m=1}^{K_i} \{ \eta_i,m \} = \Sigma_{i,c} - \Sigma_{i,a},
\]

\[
\Sigma_{i,uc}^{F,a} = \Sigma_{i,uc} \cup \Sigma_{i,a} \cup \Sigma_{i,sw}^{sw},
\]

\[
\Sigma_{i,o}^{F,a} = \Sigma_{i,o} \cup \Sigma_{i,sw}, \Sigma_{i,uo}^{F,a} = \Sigma_{i,uo}.
\]

For any actuator \( \eta_i,m \in \Sigma_{i,a} \), we assume that \( \eta_i,m \) becomes faulty when a string \( t_{i,0} \) is generated. If all the mode-switching events in \( \Sigma_{i,a}^{sw} \) are AF-safe detectable in \( t_{i,0} \), the operation of the nominal supervisor \( S_i \) can be stopped before any unsafe behaviors emerge. In this case, we construct the DFA model \( G_i^{F,a} \) of the uncontrolled subsystem after the detection of the fault. As shown in Fig. 5.11, the control objective of the post-fault supervisor
is to guarantee the fulfillment of the local safety requirement and thus, the post-fault specification is obtained as
\[
L_i^{\text{post}}(t_i,0) = h_i^{0,m}(L_i^{\text{safe}} \setminus t_i,0).
\] (5.17)

The solvability of the fault tolerant control problem in the presence of multiple actuator faults is given as follows.

**Theorem 5.2.** There exists a unified post-fault supervisor \( S_i^{F,a} \) that can steer the faulty subsystem \( G_i^{F,a} \) (\( i \in I \)) to satisfy \( L_i^{\text{post}}(t_i,0) \) (5.17) in the presence of possible faulty actuators in \( \Sigma_{i,a} \) if language \( \inf C_i^{F,a}(\{\epsilon\}) \), computed with respect to \( L(G_i^{F,a}) \) and \( \Sigma_i^{F,a} \), satisfies that \( \inf C_i^{F,a}(\{\epsilon\}) \subseteq L_i^{\text{post}}(t_i,0) \).

**Proof.** The proof follows the proof of the sufficiency part of Theorem 5.1.

**Remark 5.2.** The reason why \( \inf C_i^{F,a}(\{\epsilon\}) \subseteq L_i^{\text{post}}(t_i,0) \) is only sufficient in the multi-fault case is that \( S_i^{F,a} \) is designed under the assumption that all the actuator events in \( \Sigma_{i,a} \) are locally uncontrollable, which clearly results in a more conservative post-fault performance of the subsystem.

**The Multi-supervisor Approach** Different from the single-supervisor approach, the multi-supervisor approach utilizes multiple post-fault supervisors, each of which is corresponding to a faulty actuator. Without loss of generality, we assume that actuators \( \eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,K_i} \) of \( G_i \) (\( i \in I \)) become faulty in a consecutive manner. When no fault takes place, the subsystem stays in the nominal mode and the fault tolerant control is inactive. After some faults occur and the subsystem enters a faulty mode, the active fault tolerant control framework depicted in Fig. 5.11 is employed to resolve the impacts of the faults. We still assume that the actuator \( \eta_{h,1} \) becomes locally uncontrollable when a string \( t_{i,0} \) is generated. If the actuator fault event \( h_i^{0,1} = h_{i,1} \) is AF-safe detectable in \( t_{i,0} \), the operation of the nominal supervisor \( S_i \) is stopped before any unsafe behaviors emerge. Following the detection of
$h_{i,m}$, the uncontrolled post-fault subsystem model of $G_i$ is given by $G_i^{1,a}(t_{i,0})$, which is a DFA that satisfies

$$L(G_i^{1,a}(t_{i,0})) = h_i^{0,1}(L(G_i) \setminus t_{i,0});$$

(5.18)

with the post-fault event sets

$$
\begin{align*}
\Sigma_{i,c}^{1,a} &= \Sigma_{i,c} \setminus \{\eta_{i,1}\}, & \Sigma_{i,ac}^{1,a} &= \Sigma_{i,ac} \cup \{\eta_{i,1}, h_i^{0,1}\}, \\
\Sigma_{i,o}^{1,a} &= \Sigma_{i,o} \cup \{h_i^{0,1}\}, & \Sigma_{i,uo}^{1,a} &= \Sigma_{i,uo}.
\end{align*}
$$

(5.19)

On the other hand, if $L(G_i^{1,a})$ is actuator fault tolerant with respect to $h_i^{0,1}$ and $t_{i,0}$, a post-fault supervisor $S_i^{1,a}$ can be synthesized to satisfy the post-fault specification

$$L_{i}^{\text{post},1} = h_i^{0,1}(L_i^{\text{safe}} \setminus t_{i,0}) \subseteq L(G_i^{1,a}(t_{i,0})).$$

(5.20)

Next, for any $m = 2, 3, \ldots, K_i$, we assume that the actuator $\eta_{i,m}$ becomes faulty when a string $h_i^{m-2,m-1}t_{i,m-1} \in L(G_i^{m-1,a})$ is generated after switching to the post-fault supervisor $S_i^{m-1,a}$ (since the event $h_i^{m-2,m-1}$ is always locally uncontrollable, it must be included in the closed-loop behavior after switching to the post-fault specification in the $(m - 1)$-th faulty mode). If $h_i^{m-1,m}$ is AF-safe detectable in the string $h_i^{m-2,m-1}t_{i,m-1}$, then this mode-switching event is issued to indicate the switching from the $(m - 1)$-th to $m$-th faulty mode and to disable the operation of the post-fault supervisor $S_i^{m-1,a}$. In this case, the uncontrolled subsystem of the $m$-th faulty mode is updated as $G_i^{m,a}$ that satisfies

$$L(G_i^{m,a}) = h_i^{m-1,m}[L(G_i^{m-1,a}) \setminus (h_i^{m-2,m-1}t_{i,m-1})]$$

$$= h_i^{m-1,m}[L(G_i) \setminus (t_{i,0}t_{i,1} \cdots t_{i,m-1})];$$

(5.21)
while the local event sets in the $m$-th faulty mode is updated accordingly as

$$
\Sigma_{i,c}^{m,a} = \Sigma_{i,c}^{m-1,a} - \eta_{i,m} = \Sigma_{i,c} - \bigcup_{j=1}^{m} \{\eta_{i,j}\};
$$

$$
\Sigma_{i,uc}^{m,a} = \Sigma_{i,uc}^{m-1,a} \cup \{\eta_{i,m}, h_{i, m-1, m}\};
$$

$$
\Sigma_{i,o}^{m,a} = \Sigma_{i,o} \cup \{h_{i, m-1, m}\}; \Sigma_{i,uo}^{m,a} = \Sigma_{i,uo}.
$$

(5.22)

Meanwhile, the post-fault specification corresponding to $G_{m,a}^i$ is

$$
L_{i}^{post,m} = h_{i, m-1, m}^{i,m-1,m}[L_{i}^{safe} \backslash \{t_{i,0}t_{i,1} \cdots t_{i,m-1}\}].
$$

(5.23)

By following the conclusion of Theorem 5.1, we can test the actuator fault tolerance of $L(G_{m,a}^i)$ with respect to $h_{i, m-1, m}^{i,m} \in \Sigma_{sw,i,a}$, $h_{i, m-1, m}^{i,m} \in \Sigma_{sw,i,a}$, and $L_{i}^{m,post}$. When $L(G_{m,a}^i)$ is indeed actuator fault tolerant, then there exists a post-fault supervisor $S_{m,a}^i$ that ensures $G_{m,a}^i$ to satisfy $L_{i}^{post,m}$.

With slightly abusing the notations, we use $G_{i}^F$ to represent $G_i$ ($i \in I$) prior to and after the detection of mode-switching events $h_i^{0,m} \in \Sigma_{sw,i,a}$, and by $S_{i}^F \| G_{i}^F$ the overall closed-loop model, respectively. By definition, $S_{i}^F \| G_{i}^F$ is equivalent to $S_i \| G_i$ in the nominal mode and switches to $S_{m,a}^i \| G_{m,a}^i$ in the $m$-th faulty mode. With the aforementioned notations, we present the following theorem, suggesting that the active fault tolerant architecture proposed in this section ensure local safety regardless of faults.

**Theorem 5.3.** Consider the subsystem $G_i$ ($i \in I$) with the set of local actuators $\Sigma_{i,a}$ and the set of mode-switching events $\Sigma_{sw,i,a}$. Based on the fault tolerant control architecture shown in Fig. 5.11, the switching among the nominal supervisor $S_i$ and the post-fault supervisors $S_{m,a}^i$ ($m = 1, 2, \ldots, K_i$) ensures that the closed-loop subsystem will not violate the local safety specification $L_{i}^{safe}$, i.e., $S_{i}^F \| G_{i}^F \models L_{i}^{safe}$.

**Proof.** With slightly abusing the notations, we use $L(S_{i}^F \| G_{i}^F)$ to denote the behaviors of
the closed-loop subsystem $S_i^F G_i^F$. By construction,

$$
S_i^F G_i^F = \begin{cases} 
S_i \lvert G_i, & m = 0 \text{ (nominal mode)}; \\
S_i^{m,a} \lvert G_i^{m,a}, & m = 1, 2, \ldots, K_i.
\end{cases}
$$

Therefore, $L\left(S_i^F G_i^F\right)$ should contain the strings that are formed by concatenating strings from $L(S_i \lvert G_i)$ and strings from each the $L(S_i^{m,a} \lvert G_i^{m,a})$ ($m = 1, 2, \ldots, K_i$). Formally, we can write that

$$
L\left(S_i^F G_i^F\right) = \left\{ t_{i,0}, h_{i,1} t_{i,1} t_{i,2} \cdots h_{i,K_i-1} t_{i,K_i} \right\}.
$$

where $t_{i,0} \in L(G_i)$ and $h_{i,m-1,m} t_{i,m} \in L(G_i^{m,a})$ for $1 \leq m \leq K_i$. Note that according to (5.21), $L(G_i^{m,a}) = h_{i,m-1,m} L(G_i) \setminus \{ t_{i,0} \cdots t_{i,m-1} \}$. On the other hand, for each $2 \leq m \leq K_i$, the mode-switching event $h_{i,m-1,m}$ is AF-safe detectable in the string $h_{i,m-2,m-1} t_{i,m-1}$ ($t_{i,0}$ for $m = 1$), and the post-fault supervisor $S_i^{m,a}$ is synthesized to enforce the accomplishment of the specification $L_i^{\text{post,m}}$ (5.23). We can then write that

$$
t_{i,m} \in L_i^{\text{safe}} \setminus \{ t_{i,0} t_{i,1} \cdots t_{i,m-1} \}.
$$

Let $\Sigma_i^{F,a} = \Sigma_i \cup \Sigma_i^{aw}$ be the set of the post-fault local events, and Let $P_i^{F,a}$ be the natural projection from $\Sigma_i^{F,a*}$ to $\Sigma_i^*$. We then have

$$
P_i^{F,a} \left[ L\left(S_i^{F,a} G_i^{F,a}\right) \right] = t_{i,0} t_{i,1} \cdots t_{i,K_i} \subseteq L_i^{\text{safe}},
$$

which is to $S_i^F G_i^F \models L_i^{\text{safe}}$ from Definition 2.5. The proof is complete.

Remark 5.3. Since both single-fault and single-supervisor cases can be viewed as two special cases of the multi-supervisor case (which involve only one switching among the supervisors), the conclusion of Theorem 5.3 also holds for these two cases.
5.6 Safe Detection and Active Fault Tolerant Control against Sensor Faults

This section is concerned with the synthesis of the local post-fault supervisor for a subsystem whose nominal operation suffers from sensor faults. The proposed fault tolerant supervisory control scheme includes two major ingredients: (i) construction of an automaton model of the subsystem in the presence of local sensor faults; (ii) synthesis of the corresponding post-fault supervisor(s) after the sensor faults.

5.6.1 Supervisory Control System with Sensor Faults

Let \( G^0_i = S_i \| G_i \) denote the controlled subsystem \( G_i \) \((i \in I)\) in the nominal mode. As defined in (5.3), the suspicious sensor readings in \( \Sigma^f_{i,s} \) introduce \( N_i \) faulty modes to the subsystem \( G_i \) in addition to the nominal mode \( G^0_i \). Since undesired behaviors may arise in a faulty mode as a consequence of executing nominal supervision commands on the fault-pruned subsystem, we aim at exploring the generated behaviors of \( G^0_i \) when one sensor reading \( \sigma_{i,k} \in \Sigma_{i,s} \) becomes unavailable at this point. For \( k \in \{1, 2, \ldots, N_i\} \), we denote by \( G_{i,k} \) the DFA that models the possible behaviors of \( G_i \) in the presence of both the nominal and the faulty sensor readings.

\[
G_{i,k} = (Q_{i,k}, \Sigma_{i,k}, \delta_{i,k}), \quad k = 1, 2, \ldots, N_i, \tag{5.25}
\]

where \( Q_{i,k} = \{q_{i,k,l}|q_{i,l} \in Q_i\} \) is a copy of \( Q_i \) with an additional label \( k \), \( \Sigma_{i,k} = (\Sigma_i - \sigma_{i,k}) \cup \{\sigma^f_{i,k}\} \), and the transition function \( \delta_{i,k} \) is defined as follows: for any \( q_{i,k,l} \in Q_{i,k} \) and \( \sigma \in \Sigma_{i,k} \),

\[
\delta_{i,k}(q_{i,k,l}, \sigma) = \begin{cases} 
q_{i,k,p'}, & \text{if } (\sigma \in \Sigma_i - \{\sigma_{i,k}\}) \land (\delta_i(q_{i,l}, \sigma) = q_{i,p'}); \\
q_{i,k,p'}, & \text{if } (\sigma = \sigma^f_{i,k} \in \Sigma^f_{i,s}) \land (\delta_i(q_{i,l}, \sigma_{i,k}) = q_{i,p'}). 
\end{cases}
\]
In other words, any transitions that are labeled by the nominal sensor reading $\sigma_{i,k}$ in $G_i$ shall be replaced by the faulty one $\sigma_{i,k}^f$ in the $k$-th faulty mode $G_{i,k}$, which is in consistence with the mode automaton defined in [147]. Note that we do not specify the initial state of $G_{i,k}$ at this point since it is determined by the state of $G_i$ at which the sensor fault $f_{i,k}$ takes place. Next, we use $f_{i,k}$ to denote the mode-transition mapping

$$f_{i,k} : Q_i \rightarrow Q_{i,k}, \quad k = 1, 2, \ldots, N_i,$$

i.e., the occurrence of the sensor fault $f_{i,k}$ implies the transition from the nominal mode $G_i$ to the $k$-th faulty mode $G_{i,k}$. In other words, $f_{i,k}(q_{i,l}) = q_{i,k,l}$ indicates that $f_{i,k}$ occurs at the state $q_{i,l}$ of $G_i$ and the initial state of $G_{i,k}$ turns to be $q_{i,k,l}$ after switching to the faulty mode. The unified model that characterizes the behaviors of $G_i$ in the presence of both the nominal sensor reading $\sigma_{i,k}$ and the faulty one $\sigma_{i,k}^f$, written as $G_i^k$, can now be constructed as

$$G_i^k = (Q_i^k, \Sigma_i^k, \delta_i^k, q_{i,0}^k), \quad (5.26)$$

where $Q_i^k = Q_i \cup Q_{i,k}$, $\Sigma_i^k = \Sigma_i \cup \Sigma_{i,k} \cup \{f_{i,k}\}$, $q_{i,0}^k = q_{i,0}$, along with the transition function $\delta_i^k = \delta_i \cup \delta_{i,k} \cup \{(q_{i,l}, f_{i,k}, q_{i,k,l})|\forall q_{i,l} \in Q_i\}$. In other words, the sensor fault $f_{i,k}$ may occur at any state of $G_i$ in the nominal mode.

On the other hand, we proceed to the construction of $S_i^k$, which can be viewed as the counterpart of $G_i^k$ in the presence of $\sigma_{i,k}^f$. To pursue a DFA representation of $S_i^k$, we first assume that the nominal supervisor $S_i$ can be realized as

$$S_i = (X_i, \Sigma_i, \xi_i^0, x_{i,0}). \quad (5.27)$$

Similar to $G_i^k$, we also use $S_{i,k}$ to denote the potential behaviors of $S_i$ in the $k$-th faulty
mode, which is given by the following automaton without a specified initial state:

\[ S_{i,k} = (X_{i,k}, \Sigma_{i,k}, \xi_{i,k}), \quad k = 1, 2, \ldots, N_i, \]

where \( X_{i,k} = \{x_{i,k,l} | x_{i,l} \in X_i \} \), and for any \( x_{i,k,l} \in X_{i,k} \) and \( \sigma \in \Sigma_{i,k}, \xi_{i,k}(x_{i,k,l}, \sigma) \) is defined as

\[
\xi_{i,k}(x_{i,k,l}, \sigma) = \begin{cases} 
  x_{i,k,l}', & \text{if } \xi_i(x_{i,l}, \sigma) = x_{i,l}'; \\
  x_{i,k,l}, & \text{if } [(\sigma = \sigma_{i,k}^f) \land (\xi_i(x_{i,l}, \sigma_{i,k})!)] \lor \\
  [\sigma \in \Sigma_{i,uc} \cup \{\sigma_{i,k}^f\}] \land (\neg \xi_i(x_{i,l}, \sigma)!].
\end{cases}
\]

Intuitively, locally uncontrollable events shall never be disabled in \( S_{i,k} \) regardless of possible sensor faults by adding self-loops that are labeled by uncontrollable events. Furthermore, since the faulty sensor reading \( \sigma_{i,k}^f \) cannot be observed by \( S_i \), we add the self-loop labeled by \( \sigma_{i,k}^f \) to all the states at which a \( \sigma_{i,k} \)-labeled transition is defined as well. Therefore, the unified model \( S_i^k \) is obtained as

\[ S_i^k = (X_i^k, \Sigma_i^k, \xi_i^k, x_i^k, 0), \quad (5.28) \]

where \( X_i^k = X_i \cup X_{i,k}, x_{i,k,0} = x_{i,0} \) and \( \xi_i^k = \xi_i \cup \xi_i,k \cup \{(x_{i,l}, f_{i,k}, x_{i,k,l}) | \forall x_{i,l} \in X_i \} \). Finally, by leveraging (5.26) and (5.28), the closed-loop model of the controlled subsystem \( G_i^0 \) in the presence of a faulty sensor reading \( \sigma_{i,k}^f \) is computed as

\[ G_i^{k,s} = S_i^k \parallel G_i^k := (Q_i^{k,s}, \Sigma_i^{k,s}, \delta_i^{k,s}, q_i^{k,s,0}), \quad (5.29) \]

where the set of post-fault local events is given by

\[ \Sigma_i^{k,s} = \Sigma_i^k = \Sigma_i \cup \Sigma_i,k \cup \{f_{i,k}\} = \Sigma_i \cup \{\sigma_{i,k}^f, f_{i,k}\}; \quad (5.30) \]
and can be further partitioned as

\[
\Sigma_{i,c} = \Sigma_{i,c} \cup \{\sigma_{i,k}, f_{i,k}\}; \\
\Sigma_{i,uc} = \Sigma_{i,uc} \cup \{\sigma_{i,k}, f_{i,k}\}; \\
\Sigma_{i,o} = \Sigma_{i,o} \cup \{\sigma_{i,k}, f_{i,k}\}; \\
\Sigma_{i,uo} = \Sigma_{i,uo} \cup \{\sigma_{i,k}, f_{i,k}\}.
\] (5.31)

Note that each step (5.26)—(5.29) of the construction procedure of \(G^{k,s}_i\) requires no prior knowledge of the state at which the sensor fault event occurs. Hence \(G^{k,s}_i\) can be computed offline and can be restored for the purposes of safe fault detection and post-fault control reconfiguration a priori.

We use the following example to illustrate the aforementioned construction procedure of \(G^{k,s}_i\) and to explain why execution of nominal control commands will result in unsafe behaviors in the presence of a faulty sensor.

**Example 5.3.** (Rephrased from [25], Example 4) We consider a subsystem \(G_i\) \((i \in I)\) whose DFA representation is depicted in Fig. 5.14 (a), where \(\Sigma_i = \Sigma_{i,o} = \{a, b, c\}, \Sigma_{i,c} = \{a, c\}\). Let the set of sensor readings be the singleton \(\Sigma_{i,s} = \{b\}\). The state \(q_{i,4}\) (marked with double circles) is an unsafe state and the local safety language is hence given by \(L^{safe}_i = ab + ac\). In this example, a local supervisor \(S_i\) that ensures \(L^{safe}_i\) suffices to be a satisfactory nominal supervisor, which is depicted in Fig. 5.14 (b).

![Figure 5.14. The DFA model of \(G_i\).](image-url)
Next, we can compute the fault-pruned model $G^k_i$ and $S^k_i$ of $G_i$ and $S_i$ in the presence of $\sigma^f_{i,k}$, respectively. By following (5.26) and (5.28), respectively, the DFA models of $G^k_i$ and $S^k_i$ are obtained and are shown in Fig. 5.15 (a) and (b), respectively.

![DFA models](image)

Figure 5.15. The DFA models of $G^k_i$ and $S^k_i$.

Finally, the DFA model $G^{k,s}$ of $G^0_i$ subject to $\sigma^f_{i,k}$ is obtained by taking the parallel composition of $S^k_i$ and $G^k_i$, as shown in Fig. 5.16. Note that from Fig. 5.16, as long as the sensor fault $f_b$ occurs before the sensor reading $b$, the nominal supervisor may still allow unsafe behaviors to emerge in the fault-pruned subsystem (in this example, we can see that the string $af_bbf_c$ reaches the unsafe state $q_{i,4}$, and hence $\text{DFA}(a f_b b f_c) \not\in L_{safe}^i$, where $\text{DFA}(a f_b b f_c)$ is defined similarly as in Lemma 4.1.)
Remark 5.4. The construction procedure of $G_{i}^{k,s}$ is similar to that of the closed-loop system subject to “sensor erasure attack” in [25]. Nevertheless, we use the sensor fault event to clearly distinguish the nominal part of the system from the faulty part. Furthermore, the procedure developed in [25] considers the worst-case scenario, whereas by introducing the sensor fault event, we can see from Example 2 that if a sensor fault takes place after the last occurrence of the corresponding sensor reading, the fault will pose no impact on the successive behaviors of the system.

5.6.2 SF-safe Controllability and Active Sensor Fault Tolerant Control

By construction, $G_{i}^{k,s}$ ($i \in I, k \in \{1, 2, \ldots, N_{i}\}$) contains all the potential behaviors of the closed-loop subsystem subject to $f_{i,k}$ and $\sigma_{i,m}^{f}$. Facing the fault-pruned controlled subsystem $G_{i}^{k,s}$, two objectives should be fulfilled in order to mitigate the influence of
the faulty sensor: (i) the occurrence of sensor fault event \( f_{i,k} \) should be unambiguously determined before \( G^{k,s}_{i} \) generates any unsafe behaviors; (ii) after the detection of the sensor fault, \( G^{k,s}_{i} \) should be able to stop the operation of \( S_{i} \) before generating any unsafe strings.

Let 
\[
\Psi(\sigma^{f}_{i,k}) = \{ t \in L(G^{k,s}_{i}) | t = t'\sigma^{f}_{i,k}, t' \in \Sigma^{k,s'}_{i} \}
\]
denote the set of all strings in \( L(G^{k,s}_{i}) \) that end with the faulty sensor reading \( \sigma^{f}_{i,k} \). Let \( P^{k}_{i,o} \) be the local observation projection from \( \Sigma^{k,s'}_{i} \) (5.30) to \( \Sigma^{k,s'}_{i,o} \) (5.31) in the \( k \)-th faulty mode. We introduce the property of SF-safe controllability\(^2\) to achieve the aforementioned control objectives for a fault-pruned subsystem.

**Definition 5.6** (SF-safe Controllability). The prefix-closed language \( L(G^{k,s}_{i}) \) is SF-safe controllable with respect to the post-fault projection \( P^{k}_{i,o} \), the faulty sensor reading \( \sigma^{f}_{i,k} \) and the local safety property \( L^{safe}_{i} \) if

\[
(\forall s_{i} \in \Psi(\sigma^{f}_{i,k}))(\forall t_{i} \in L(G^{k,s}_{i}) \setminus s_{i}) \left[ \left( \text{DFA}(s_{i}t_{i}) \neq L^{safe}_{i} \right) \land 
(\forall s'_{i} \in s_{i}t_{i} - \{ s_{i}t_{i} \}, \text{DFA}(s'_{i}) = L^{safe}_{i}) \right] \Rightarrow SC,
\]

where \( \text{DFA}(\cdot) \) defines the string automaton as illustrated in Lemma 4.1, and the safe controllability condition \( SC \) states as follows:

\[
SC : (\exists t_{i,1}, t_{i,2} \in \Sigma^{k,s'}_{i}, t_{i} = t_{i,1}t_{i,2}) \left[ \left( \left( \exists w_{i} \in L(G^{k,s}_{i}) \right) \left( \left( P^{k}_{i,o}(w_{i}) = P^{k}_{i,o}(s_{i}t_{i,1}) \right) \land (\sigma^{f}_{i,k} \notin w_{i}) \right) \right) \land 
(\Sigma^{k,s}_{i,c} \in t_{i,2}) \right],
\]

where by \( \Sigma^{k,s}_{i,c} \in t_{i,2} \) we mean that \( T_{i,2} \cap \Psi(\Sigma^{k,s}_{i,c}) \neq \emptyset \).

Definition 5.6 can be viewed as a variant of safe controllability [130] in the context of active fault tolerant control; nevertheless, Definition 5.6 is different from the safe controllability defined in [130] in the sense that (local) observability of a sensor reading may

\(^2\)“SF” is short for “sensor faults”.
vary depends on the occurrence of the fault. We now present the necessary and sufficient condition for $G_{i}^{k,s}$ to ensure safety in terms of SF-safe controllability.

**Theorem 5.4.** The controlled subsystem $G_{i}^{k,s}$ with the faulty sensor reading $\sigma_{i,k}^{f}$ can prevent any string that violates the local safety requirement $L_{i}^{\text{safe}}$ (in the sense of Definition 2.5) from emerging if and only if $L(G_{i}^{k,s})$ is SF-safe controllable with respect to $P_{i,o}^{k}$, $\sigma_{i,k}^{f}$ and $L_{i}^{\text{safe}}$.

**Proof.** According to Definition 5.6, $L(G_{i}^{k,s})$ is SF-safe controllable with respect to $P_{i,o}^{k}$, $\sigma_{i,k}^{f}$ and $L_{i}^{\text{safe}}$ if for any string $t_{i}$ in $L(G_{i}^{k,s})$ following the faulty sensor reading $\sigma_{i,k}^{f}$ that may cause violation of $L_{i}^{\text{safe}}$ (in the sense of Definition 2.5): (i) there exists a proper prefix $t_{i,1} \in T_{i}$ that assures the unambiguous detection of $\sigma_{i,k}^{f}$ (and hence $f_{i,k}$) before $G_{i}^{k,s}$ generates any unsafe behavior (safe detectability); (ii) once $\sigma_{i,k}^{f}$ is detected to replace $\sigma_{i,k}$ in the $k$-th faulty mode, there also exists a locally controllable event after in the prefix in (i) while still prior to the execution of the unsafe behavior (safe controllability). Therefore, unsafe behaviors can always be prohibited by disabling this locally controllable event after the detection of the fault.

The occurrence of $\sigma_{i,k}^{f}$ (and hence $f_{i,k}$) is determined by a diagnoser. The construction procedure of the diagnoser has already been presented in [155] and is omitted here. Before proceeding to the sensor fault tolerant control strategy when facing $G_{i}^{k,s}$, we first review the concept of first-entered certain states in a diagnoser.

**Definition 5.7** (First-entered Certain States). [130] Let $G_{i}^{d} = (Q_{i}^{d}, \Sigma_{i,0}^{d}, \delta_{i}^{d}, q_{i,0}^{d})$ be the diagnoser constructed for $G_{i}^{k,s}$ and $P_{i,o}^{k}$. Define $Q_{i}^{YN} = \{ q \in Q_{i}^{d} | q \text{ is uncertain} \}$, $Q_{i}^{N} = \{ q \in Q_{i}^{d} | q \text{ is normal} \}$ and $Q_{i}^{Y} = \{ q \in Q_{i}^{d} | q \text{ is certain} \}$. The set of first-entered certain states is $\mathcal{FC}_{i} = \{ q \in Q_{i}^{Y} | (\exists q' \in Q_{i}^{N} \cup Q_{i}^{YN}, \sigma \in \Sigma_{i,0}^{k,s})[\delta_{i}^{d}(q', \sigma) = q] \}$.

Let $Q_{i,B} = \{ q \in Q_{i}^{k,s} | (\exists s \in L(G_{i}^{k,s}), \delta_{i}^{k,s}(q_{i,0}^{k,s}, s) = q) \} \setminus L_{i}^{\text{safe}}$ denote the set of unsafe states in $G_{i}^{k,s}$. By introducing $Q_{i,B}$, $G_{i}^{d}$ can be modified as a safe diagnoser.
[128] and the SF-safe controllability of \( L(G_i^{k,s}) \) can then be verified offline, as stated in the following proposition.

**Proposition 5.1.** Language \( L(G_i^{k,s}) \) is SF-safe controllable with respect to \( P_{i,o}^k \), \( \sigma_{i,k}^f \) and \( L_i^{\text{safef}} \) if and only if for the safe diagnoser \( G_d^i \):

(i) there does not exist a state \( q_i^{YN} = \{(q_{i1}^{k,s}, l_{i1}), (q_{i2}^{k,s}, l_{i2}), \ldots, (q_{iK}^{k,s}, l_{iK})\} \in Q_i^{YN} \) such that \( \exists j \in \{1, 2, \ldots, K\}, l_{ij} = Y \) but \( q_{ij}^{k,s} \in Q_i,B \);

(ii) there does not exist a state \( q_i^{Y} = \{(q_{i1}^{k,s}, Y), (q_{i2}^{k,s}, Y), \ldots, (q_{iK}^{k,s}, Y)\} \in FC_i \) such that \( \exists j \in \{1, 2, \ldots, K\}, q_{ij}^{k,s} \in Q_i,B \);

(iii) for any \( q_i^{Y} = \{(q_{i1}^{k,s}, Y), (q_{i2}^{k,s}, Y), \ldots, (q_{iK}^{k,s}, Y)\} \in FC_i \) and \( \forall j \in \{1, 2, \ldots, K\}, \) there does not exist a state \( q_{ij}' \in Q_i,B \) and a string \( s \in \Sigma_{i,uc}^{k,s} \) such that \( q_{ij}' = \delta_i^{k,s}(q_{ij}, s) \).

**Proof.** Conditions (i) and (ii) are the necessary and sufficient conditions for safe diagnosability and the proof is presented in [128]. The necessity and sufficiency of Condition (iii) follow immediately from Definition 5.6.

We now determine the SF-safe controllability of the subsystem \( G_i \) in Example 5.3.

**Example 5.4.** Let us reconsider the fault-pruned subsystem \( G_i^{k,s} \) in Example 5.3.
In this example, we assume that the sensor fault event \( f_b \) occurs before the sensor reading \( b \) can be received by \( G^{k,s}_i \). It then can be inferred from Example 5.3 that under this circumstance the system \( G^{k,s}_i \) may evolve to an unsafe state. The safe diagnoser is then constructed accordingly and is shown in Fig. 5.18.

![Diagram of the safe diagnoser \( G^d_i \) of \( G^{k,s}_i \).](image)

Figure 5.18. The safe diagnoser \( G^d_i \) of \( G^{k,s}_i \).

As shown in Fig. 5.18, the diagnoser state corresponding to the observation of a string \( ac \) is given by \( q_i = \{(x_{i,2}, q_{i,2}), N\}, \{(x_{i,b,2}, q_{i,b,2}), N\}, \{(x_{i,b,2}, q_{i,b,4}), Y\}\), which is an \( \sigma^{f,k}_{i,k} \)-uncertain state. Furthermore, it is worth pointing out that \( (x_{i,b,2}, q_{i,b,4}), Y\) \( \in q_i \) contains the unsafe state \( q_{i,b,4} \) of the original subsystem. Therefore, the fault and the nominal supervisor \( S_i \) jointly guide the subsystem \( G_i \) to the unsafe state and thus Condition (i) in Proposition 5.1 fails to be satisfied and therefore \( L(G^{k,s}_i) \) is not SF-safe controllable with respect to \( P^b_{i,o}, b^f \) and \( L^\text{safe}_i \).

5.6.3 Active Sensor Fault Tolerant Control Architecture

Thanks to the property of SF-safe controllability, we are able to adopt an active approach to address the loss of a sensor reading in a subsystem. The sensor fault tolerant
control architecture under consideration is shown in Fig. 5.19. Different from the actuator fault tolerant control architecture shown in Fig. 5.11, the sensor fault event \( f_{i,k} \) here is assumed to be locally unobservable and therefore, detection of \( f_{i,k} \) should be performed by equipping \( G_{i}^{k,s} \) with the safe diagnoser \( G_{i}^{d} \). If \( L(G_{i}^{k,s}) \) is SF-safe controllable with respect to \( P_{i,o}^{k}, \sigma_{i,k}^{f} \) and \( L_{i}^{safe} \), then the generation of the faulty sensor reading \( \sigma_{i,k}^{f} \) is guaranteed to be determined by \( G_{i}^{d} \) without generating any unsafe behaviors.

![Figure 5.19. The active fault tolerant control framework of \( G_{i}^{k,s} \).](image)

After \( G_{i}^{d} \) detects \( f_{i,k} \) and reports that \( G_{i}^{k,s} \) has evolved to a first-entered certain state \( q_{i}^{Y} \) = \( \{(q_{i1}^{k,s}, Y), (q_{i2}^{k,s}, Y), \ldots, (q_{iK}^{k,s}, Y)\} \) \( \in FC_{i} \), the post-fault uncontrolled subsystem \( G_{i}^{k,s}(q_{i}^{Y}) \) can be formed by first disabling the nominal supervisor \( S_{i} \) and then taking the accessible part of of the uncontrolled faulty subsystem \( G_{i}^{k} \) starting from each \( q_{ij}, j \in \{1, 2, \ldots, K\} \). In order to make \( G_{i}^{k,s}(q_{i}^{Y}) \) deterministic, we inherit the idea of constructing \( G_{i}^{m,a} \) in the previous section and add a new initial state \( q_{i,0}^{Y} \) so that it can be connected with each \( q_{ij} \) with a transition labeled as \( (q_{i,0}^{Y}, detect_{j}, q_{ij}) \), where \( j \in \{1, 2, \ldots, K\} \) and \( detect_{j} \) is locally uncontrollable. The SF-safe controllability of \( L(G_{i}^{k,s}) \) ensures that there always exists a locally controllable event \( \sigma_{j} \) corresponding to each \( j \) that can be disabled to prevent
the subsystem from generating an unsafe string after evolving to \( q_{ij} \). Let \( t_{ij,0} \in L(G_{ij}^{k,s}) \) be the string such that \( \delta_i^{k,s}(q_{ij,0}, t_{ij,0}) = q_{ij} \). We require that local safety can still be assured after the detection of the fault and the post-fault specification \( L_i^{\text{post}}(q_i^Y) \) is given by

\[
L_i^{\text{post}}(q_i^Y) = \bigcup_{j=1}^{K} \text{detect}_j(L_i^{\text{safe}} \setminus t_{ij,0}), \tag{5.34}
\]

which turns to be a prefix-closed sublanguage of \( L(G_{i}^{k,s}(q_i^Y)) \). The following property of sensor fault tolerance is hence formally defined in terms of \( G_{i}^{k,s}(q_i^Y) \) and \( L_i^{\text{post}}(q_i^Y) \).

**Definition 5.8 (Sensor Fault Tolerance).** Language \( L(G_{i}^{k,s}(q_i^Y)) \) is said to be sensor fault tolerant with respect to \( \sigma_{f,i,k} \) and \( L_i^{\text{post}}(q_i^Y) \) if there exists a non-empty sublanguage of \( L_i^{\text{post}}(q_i^Y) \) that is controllable with respect to \( G_{i}^{k,s}(q_i^Y) \) and \( \Sigma_{i,uc}^{k,s} \cup \{\text{detect}_j|q_i \in q_i^Y\} \) and observable with respect to \( G_{i}^{k,s}(q_i^Y) \) and \( \Sigma_{i,o}^{k,s} \). \( L(G_{i}^{k,s}) \) is said to be sensor fault tolerant if for all \( q_i^Y \in FC_i \), \( L(G_{i}^{k,s}(q_i^Y)) \) is sensor fault tolerant with respect to \( L_i^{\text{post}}(q_i^Y) \).

We present the following theorem that states the necessary and sufficient conditions for the sensor fault tolerance of \( L(G_{i}^{k,s}(q_i^Y)) \).

**Theorem 5.5.** Language \( L(G_{i}^{k,s}(q_i^Y)) \) is sensor fault tolerant with respect to \( \sigma_{f,i,k} \) and \( L_i^{\text{post}}(q_i^Y) \) if and only if language \( \inf C_{i}^{k,s}({\epsilon}) \), computed with respect to \( L(G_{i}^{k,s}(q_i^Y)) \) and \( \Sigma_{i,uc}^{k,s} \cup \{\text{detect}_j|q_i \in q_i^Y\} \), satisfies that \( \text{DFA}\left(\inf C_{i}^{k,s}({\epsilon})\right) \models L_i^{\text{post}}(q_i^Y) \).

**Proof.** The theorem can be proved in a similar way as that of Theorem 5.1. Note that we use \( \models \) instead of \( \subseteq \) in this theorem due to the introduction of the detection event \( \text{detect}_j \) for each distinct \( q_{ij} \in q_i^Y \).

Theorem 5.5 indeed guarantees the existence of a satisfactory post-fault supervisor. Since the fault-pruned subsystem \( G_{i}^{k,s} \) and the corresponding safe diagnoser \( G_{i}^{d} \) can both be computed offline, all the possible post-fault transition diagram of \( G_i \) after entering a state \( q_i^Y \in FC_i \) can also be obtained offline. Therefore, a post-fault supervisor \( S_i^{k,s}(q_i^Y) \)
starting from the states in $q^Y_i$ can be implemented online and the system shall switch to $S^{k,s}_i(q^Y_i)$ after $G^{k,s}_i$ visits any state in $q^Y_i$.

**Remark 5.5.** One may notice that the sensor fault tolerance of the subsystem $G_i$ in the presence of any faulty sensor readings can always be assured by synthesizing the nominal supervisor $S_i$ with respect to $\Sigma^{k,s}_{i,o}$ rather than $\Sigma_{i,o}$ in the first place; however, this approach would presumably lead the controlled subsystem to perform more restrictive behaviors.

5.6.4 Fault Tolerant Control with Multiple Faulty Sensors

We now consider the case in which any sensor readings in $\Sigma_{i,s}$ may become unavailable during the evolution of $G^0_i$. For such a purpose, we first construct a DFA model $G^{F,s}_i$ that characterizes the behaviors of $G^0_i$ in the presence of loss of sensor readings in $\Sigma_{i,s}$. The construction procedure of $G^{F,s}_i$ starts with the counterpart $G_{i,F}$ of $G_{i,k}$ in the presence of multiple sensor faults, which is written as

$$G_{i,F} = (Q_{i,F}, \Sigma_{i,F}, \delta_{i,F}), \quad (5.35)$$

where $Q_{i,F} = \{q_{i,F,l} | q_{i,l} \in Q_i\}$ is a copy of $Q_i$, $\Sigma_{i,F} = (\Sigma_i \cup \Sigma_{i,s}^f) \cup \Sigma_{i,s}^f$, and the transition function $\delta_{i,F}$ is defined as follows: for any $q_{i,F,l} \in Q_{i,F}$ and $\sigma \in \Sigma_{i,F}$,

$$\delta_{i,F}(q_{i,F,l}, \sigma) = \begin{cases} 
q_{i,F,l'}, & (\sigma \in \Sigma_i - \Sigma_{i,s}) \land (\delta_i(q_{i,l}, \sigma) = q_{i,l'}) \\
q_{i,F,l'}, & (\sigma = \sigma_{i,k}^f \in \Sigma_{i,s}^f) \land \\
& (\delta_i(q_{i,l}, \sigma_{i,k}) = q_{i,l'})
\end{cases}$$

The DFA model $G^F_i$ of the uncontrolled subsystem $G_i$ in the presence of multiple sensor faults can then be formed in a similar way as $G^k_i$ in the single-fault case (5.26):

$$G^F_i = (Q^F_i, \Sigma^F_i, \delta^F_i, q^F_{i,0}), \quad (5.36)$$
where $Q_i^F = Q_i \cup Q_{i,F}$, $\Sigma_i^F = \Sigma_i \cup \Sigma_{i,F} \cup \Sigma_{i,s}^F$, $q_{i,0}^F = q_{i,0}$, with the transition function $\delta_i^F = \delta_i \cup \delta_{i,F} \cup \{(q_{i,l}, f_{i,k}, q_{i,k,l})|\forall q_{i,l} \in Q_i, \forall f_{i,k} \in \Sigma_{i,s}^F \}$.

In addition to $G_i^F$, the DFA model $S_i^F$ of $S_i$ in the multi-fault is obtained as

$$S_{i,F} = (X_{i,F}, \Sigma_{i,F}, \xi_{i,F}),$$

where $X_{i,F} = \{x_{i,F,l}|x_{i,l} \in X_i\}$, and for any $x_{i,F,l} \in X_{i,F}$ and $\sigma \in \Sigma_{i,F}$, $\xi_{i,F}(x_{i,F,l}, \sigma)$ is formally defined as

$$\xi_{i,F}(x_{i,F,l}, \sigma) = \begin{cases} x_{i,F,l}', & \text{if } \xi_i(x_{i,l}, \sigma) = x_{i,l}'; \\ x_{i,F,l}, & \text{if } [\sigma = \sigma_{i,k}^f \in \Sigma_{i,s}^F) \land \\
& (\xi_i(x_{i,l}, \sigma_{i,k})!)] \lor [(\sigma \in \Sigma_{i,uc} \lor \Sigma_{i,s}^F) \\
& \land (\neg \xi_i(x_{i,l}, \sigma)!)]. \end{cases}$$

The unified model of $S_i^F$ is hence obtained as

$$S_i^F = (X_i^F, \Sigma_i^F, \xi_i^F, x_{i,0}^F), \quad (5.37)$$

where $X_i^F = X_i \cup X_{i,F}$, $x_{i,0}^F = x_{i,0}$ and $\xi_i^F = \xi_i \cup \xi_{i,F} \cup \{(x_{i,l}, f_{i,k}, x_{i,F,l})|\forall x_{i,l} \in X_i, \forall f_{i,k} \in \Sigma_{i,s}^F\}$. Finally, by following (5.29), the closed-loop model $G_i^{F,s}$ is obtained as

$$G_i^{F,s} = S_i^F \parallel G_i^F := (Q_i^{F,s}, \Sigma_i^{F,s}, \delta_i^{F,s}, q_{i,0}^{F,s}), \quad (5.38)$$

where

$$\Sigma_i^{F,s} = \Sigma_i^F = \Sigma_i \cup \Sigma_{i,F} \cup \Sigma_{i,s}, \quad (5.39)$$
and

\[
\Sigma_{i,c}^{F,s} = \Sigma_{i,c}, \quad \Sigma_{i,uc}^{F,s} = \Sigma_{i,uc} \cup \Sigma_{i,uc}^f \cup \Sigma_{i,s}^f; \quad \Sigma_{i,o}^{F,s} = \Sigma_{i,o} \cup \Sigma_{i,o}^f \cup \Sigma_{i,s}^f.
\]

(5.40)

The sensor fault tolerant control framework to deal with the impacts of multiple sensor faults is developed in Fig. 5.20, where \( P_{i,o}^F : \Sigma_{i}^{F,s} \rightarrow \Sigma_{i,o}^{F,s} \) stands for the post-fault local observation projection in the multi-fault case. In the multi-fault case, the safe diagnoser \( G_{d,i} \) can be modified to distinguish different sensor faults by introducing fault labels corresponding to each sensor fault event. When no sensor fault is detected by \( G_{d,i} \), the subsystem \( G_{i}^{F,s} \) remains in the nominal mode. If for each \( k \in \{1, 2, \ldots, K_i\} \), the language \( L(G_{i}^{F,s}) \) is SF-safe controllable with respect to \( P_{i,o}^F, f_{i,k} \) and \( L_{i}^{safe} \), the safe diagnoser \( G_{d,i} \) is able to correctly detect the occurrence of \( f_{i,k} \) before \( G_{i}^{F,s} \) generates any (locally) unsafe behaviors.

With slightly abusing the notations, we still denote by \( FC_i \) the set of first-entered certain states with respect to all possible sensor fault events in \( \Sigma_{i,s}^{F,s} \). Thanks to \( FC_i \), we can apply Proposition 1 for the verification of the SF-safe controllability of \( L(G_{i}^{F,s}) \) in the presence of multiple sensor faults.

![Figure 5.20. Sensor fault tolerant control with multiple faults.](image_url)
We leverage the idea of the single-supervisor approach in the context of actuator fault tolerant control here to address multiple sensor faults. When entering a certain state \( q_i^Y \in FC_i \), the diagnoser reports the occurrence of the corresponding sensor fault \( f_{i,k} \) for some \( k \in \{1, 2, \ldots, K_i\} \) and interrupts the operation of the nominal supervisor \( S_i \). In response to the detection of the sensor fault, the construction procedure for the post-fault uncontrolled subsystem in the single-fault case can be inherited to compute \( G_{i,k}^{F,s}(q_i^Y) \), while the post-fault control specification is obtained as \( (5.34) \). The sufficient condition for the solvability of the fault tolerant control problem under multiple sensor faults is presented as follows.

**Theorem 5.6.** There exists a post-fault supervisor \( S_{i,F,s}(q_i^Y) \) after \( G_i^F \) evolves to a certain state \( q_i^Y \) that drives the post-fault subsystem \( G_{i,F,s}(q_i^Y) \) to satisfy \( L_{i,post}(q_i^Y) \) under arbitrary order of loss of sensor readings in \( \Sigma_{i,F,s} \) if language \( \inf C_{i,F,s}(\{\epsilon\}) \), computed with respect to \( L(G_{i,F,s}(q_i^Y)) \) and \( \Sigma_{i,F,s}^{F,s} \cup \{\text{detect} j | q_i^j \in q_i^Y\} \), satisfies that \( \text{DFA} \left( \inf C_{i,F,s}(\{\epsilon\}) \right) \models L_{i,post}(q_i^Y) \).

**Proof.** The proof follows the proof of the sufficiency part of Theorem 5.1. \( \square \)

**Remark 5.6.** The condition \( \text{DFA} \left( \inf C_{i,F,s}(\{\epsilon\}) \right) \models L_{i,post}(q_i^Y) \) is only sufficient in the multi-fault case rather than sufficient and necessary as stated in Theorem 5.5. The reason is due to two aspects: (i) not all the unsafe behaviors may arise in the unified model \( G_{i,F,s} \); (ii) the single post-fault supervisor \( S_{i,F,s} \) is designed with respect to \( \Sigma_{i,F,s}^{F,s} \) and \( \Sigma_{i,F,o}^{F,s} \), which just considers the worst-case scenario and clearly results in a more conservative post-fault closed-loop performance.

The notion \( S_{i,F}^F \parallel G_i^F \) is also adopted here to represent the overall closed-loop model of the subsystem \( G_i \) \( (i \in I) \) before and after the detection of multiple sensor faults. The following theorem states that the active fault tolerant architecture proposed in Fig. 5.20 shall ensure the local safety of \( G_i \) regardless of faults.

**Theorem 5.7.** Consider the subsystem \( G_i \) \( (i \in I) \) subject to possible loss of sensor readings \( \sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,K_i} \). If for each \( k \in \{1, 2, \ldots, K_i\} \), \( L(G_i^{k,s}) \) is SF-safe controllable
with respect to $P^F_{i,o}, \sigma^f_{i,k}$ and $L^\text{safe}_i$ and for each $q^Y_i \in FC_i$, $L(G_i^{k,s}(q^Y_i))$ is sensor fault tolerant with respect to $\sigma^f_{i,k}$ and $L^\text{post}_i(q^Y_i)$, then the nominal supervisor $S_i$ and the post-fault supervisor $S^{F,s}_i$ will jointly enforce the fulfillment of local safety requirement $L^\text{safe}_i$, i.e., $S^F_i \parallel G^F_i \models L^\text{safe}_i$.

**Proof.** By definition, the behaviors $L(S^F_i \parallel G^F_i)$ of the subsystem $G_i$ under the influence of multiple sensor faults should be the concatenation of three parts: the nominal behaviors of $G_i$ under the supervision of $S_i$ before the occurrence of the fault, the behaviors generated after the occurrence but before the detection of the fault, and the behaviors generated by the post-fault subsystem controlled by $S^{F,s}_i$. More specifically, $L(S^F_i \parallel G^F_i)$ should be represented as the following form:

$$L(S^F_i \parallel G^F_i) = \left\{ s_i f_{i,k} t_i \sigma^f_{i,k} \text{detect}_j t'_i \right\},$$

where $s_i \in L(G^0_i)$ is the string generated before $f_{i,k}$, $t_i \in L(G_i^{k,s}) \setminus \Psi(\sigma^f_{i,k})$ is the string executed by $G_i^{k,s}$ in the faulty mode but before the switching to $\sigma^f_{i,k}$ is detected, and $t'_i$ is the string generated by the post-fault controlled subsystem $S^{F,s}_i(q^Y_i) \parallel G^{F,s}_i(q^Y_i)$ if $G^d_i$ enters a certain state $q^Y_i$. In other words, $\overline{s_i} \subseteq L_i \subseteq L^\text{safe}_i$ and $\text{DFA}(s_i f_{i,k} t_i) \models L^\text{safe}_i$ can be guaranteed due to the SF-safe controllability of $L(G_i^{k,s})$ for each $k \in \{1, 2, \ldots, K_i\}$.

Furthermore, when evolving to any fault certain state $q^Y_i$, the sensor fault tolerance property can assure the existence of a post-fault supervisor $S^{F,s}_i(q^Y_i)$ that satisfies $L^\text{post}_i(q^Y_i)$ in the faulty mode in the presence of $\Sigma^F_{i,uc}$ and $\Sigma^F_{i,uo}$; therefore, recall (5.34), we can write that

$$\overline{\text{detect}_j t'_i} \subseteq L^\text{post}_i(q^Y_i) \setminus (s_i t_i)$$

for some $j$ such that there exists $q_{i,j} \in q^Y_i$. Let $P^{F,s}_i$ denote the natural projection from $[\Sigma^F_i \cup \{\text{detect}_j q_{i,j} \in q^Y_i\}]^*$ to $\Sigma^*_i$. It then follows that

$$P^{F,s}_i \left[ L(S^F_i \parallel G^F_i) \right] = \left\{ s_i f_{i,k} \overline{t'_i} \right\} \subseteq L^\text{safe}_i,$$

(5.41)
which is equivalent to $S^F_i \| G^F_i \models L^{safe}_i$.

5.7 Fault Tolerant Control in the Presence of Combinations of Faults

So far, we have only considered one class of faults that may occur in a subsystem. In this subsection, we aim at extending the proposed fault tolerant control approaches to take both actuator and sensor faults into consideration. Without loss of generality, we consider a subsystem $G_i (i \in I)$ in which both $\Sigma_{i,s} = \{\sigma_i\}$ and $\Sigma_{i,a} = \{\eta_i\}$ are singletons. We assume that the sensor fault occurs before the actuator fault, and the fault tolerance of the supervisory control strategy of $G_i$ is sketched as follows. Note that our methodology can be generalized to other combinations of faults.

(1) When no fault is detected in $G_i$, the nominal supervisor $S_i$ is employed such that $L(G^0_i) = L_i \subseteq L^{safe}_i$. With $\Sigma^F_{i,s} = \{f_i\}$ and $\Sigma^F_{i,a} = \{\sigma^F_i\}$, we can construct the fault-pruned model $G^F,s_i$ of $G^0_i$. Furthermore, $G^F,s_i$ is monitored by the safe diagnoser $G^d_i$.

(2) If $L(G^F,s_i)$ is SF-safe controllable and $G^d_i$ detects the occurrence of the faulty sensor reading $\sigma^F_i$ rather than the nominal one $\sigma_i$ by entering a certain state $q^Y_i \in FC_i$, the operation of the nominal supervisor $S_i$ can then be disabled before generating any unsafe behaviors. Facing the post-fault model $G^F,s_i(q^Y_i)$ of the uncontrolled subsystem and the post-fault specification $L^\text{post}_i(q^Y_i)$, the sensor fault tolerant supervisor $S^F,s_i(q^Y_i)$ can be synthesized and implemented online, provided that $L(G^F,s_i(q^Y_i))$ is sensor fault tolerant with respect to $f_i$ and $L^\text{post}_i(q^Y_i)$. After switching to $S^F,s_i$, we denote by $S^F_i \| G^F_i$ the overall closed-loop subsystem.

(3) We further assume that $t_i \in L(S^F_i \| G^F_i)$ is generated when the actuator fault event $h_i$ is detected. If $h_i$ is AF-safe detectable in the string $t_i$, we are able to construct the uncontrolled post-fault subsystem $G^F,a_i(t_i)$ as stated in Section 5.5.2 and the post-fault specification is updated as $L^{\text{post}}_i = h_i[L^{safe}_i \setminus P^F,s_i(t_i)]$ as the post-fault specification, where $P^F,s_i$ is defined in Theorem 5.7. In this faulty mode, we update the local event sets as (5.12) and (5.13) accordingly, and if actuator fault tolerance of $L(G^F,a_i(t_i))$ is satisfied, we can switch
to a second post-fault supervisor $S^{F,a}_i$ that assures local safety with the faulty actuator and sensor.

For brevity of presentation, we still use $S^{F}_i \parallel G^{F}_i$ ($i \in I$) as a unified notation to represent the closed-loop subsystem in the presence of combinations of faults. The following theorem, suggesting that the integration of the fault tolerant control techniques jointly enforce the local safety of a subsystem subject to faults, can be viewed as an immediate result by following the conclusions of Theorem 5.3 and Theorem 5.7.

**Theorem 5.8.** For the subsystem $G_i$ ($i \in I$) that is subject to actuator faults in $\Sigma_{i,a}^F$ and sensor faults in $\Sigma_{i,s}^F$, if there exists a set of post-fault supervisors that can be computed via the proposed fault tolerant techniques depicted in Fig. 5.11 and Fig. 5.20, then these post-fault supervisors will jointly assure that $S^{F}_i \parallel G^{F}_i \models L_{safe}^i$.

### 5.8 Assume-guarantee Post-fault Coordination of Distributed DESs

Theorem 5.8 guarantees that local safety of each subsystem can be enforced after switching to post-fault supervisor(s). Nevertheless, undesirable behaviors may still arise when post-fault subsystems are coordinated with the nominal ones, leading to the violation of the global specification $L$. We resolve this concern in this section by developing fault tolerant coordination strategies among the subsystems. In particular, an assume-guarantee paradigm [36] is exploited to efficiently refine the local supervisors in order to achieve the global specification. In this subsection, we propose an assume-guarantee scheme to coordinate nominal subsystems with subsystems after switching to post-fault supervisor(s). To apply the SYM-N proof rule for the coordination of the controlled distributed DES, we use $L \subseteq \Sigma^*$ as the property to be verified and let $M_i := G^0_i$ ($i \in I$) denote component module of the controlled subsystem in the nominal mode. In this case, the weakest assumption $A_i$ with respect to $M_i$ and $L$ is a DFA that is defined over $\Sigma_{A_i} := \Sigma_{-i}$. Since the SYM-N rule is sound and complete, all the assumptions $A_i$ ($i \in I$) jointly satisfy the $(n+1)$-th premise of the SYM-N proof rule.
For the sake of simplicity of presentation, we assume that operation of one subsystem $M_i (i \in I)$ suffers from possible actuator/sensor faults and switches to a post-fault supervisor. The post-fault counterpart of $M_i$, written as $M^F_i$, is defined as a DFA that satisfies $L(M^F_i) = P^F_i \left[ L(S^F_i \parallel G^F_i) \right] \subseteq \Sigma^*_i$, where $P^F_i$ denotes the natural projection from $\Sigma^F_i$ to $\Sigma^*_i$. According to Definition 2.5, $[\parallel j \in I - \{i\} \left( S_j \parallel G_j \right) \parallel (S^F_i \parallel G^F_i) \models L]$ reduces to $(\parallel j \in I - \{i\} M_j) \parallel M^F_i \models L$. The weakest assumption with respect to the post-fault subsystem module $M^F_i$ and the global specification $L$, written as $A^F_i$, is also defined over $\Sigma_{-i}$ and can be computed accordingly via existing methods [36].

![Figure 5.21. Fault tolerant assume-guarantee coordination.](image)

The assume-guarantee paradigm for the post-fault coordination among the subsystems of $G$ is illustrated in Fig. 5.21. The proposed coordination scheme first seeks to maintain...
the global specification even in the presence of a subsystem with post-fault local supervisor(s). The following proposition states the necessary and sufficient condition under which the successful coordination can be ensured without further supervisor refinement.

**Proposition 5.2.** \( (\| j \in I - \{i\} M_j \| M^F_i \models L \) if and only if \)

\[ L_m \left( \| j \in I - \{i\} coA_j \| coA_i^F \right) \subseteq L. \]

**Proof.** On the one hand, defined over \( \Sigma_{-i} \), \( A_i^F \) is the weakest assumption such that the \( i \)-th premise of the SYM-N proof rule, \( \langle A_i^F \rangle M^F_i \langle L \rangle \), is satisfied. Furthermore, for each \( j \in I - \{i\} \), the \( j \)-th premise of the SYM-N rule, \( \langle A_j \rangle M_j \langle L \rangle \), holds automatically. If \( L_m \left[ \| j \in I - \{i\} coA_j \| coA_i^F \right] \subseteq L \), the \((n + 1)\)-th premise of the SYM-N rule is satisfied and \( (\| j \in I - \{i\} M_j \| M^F_i \models L \) is enforced by the soundness of the SYM-N proof rule.

On the other hand, if \( (\| j \in I - \{i\} M_j \| M^F_i \models L \) holds, then for the weakest assumptions \( A_j \) (with respect to \( M_j \) \((j \in I - \{i\}) \)) and \( A_i^F \) (with respect to \( M^F_i \) and \( L \)), premises 1 through \( n \) of the SYM-N proof rule are satisfied. Therefore, serving as the \((n + 1)\)-th premise of the SYM-N proof rule, \( L_m \left[ \| j \in I - \{i\} coA_j \| coA_i^F \right] \subseteq L \) is assured by the completeness of the SYM-N proof rule. \( \square \)

If Proposition 5.2 cannot be satisfied, we need to figure out how to refine the controlled behaviors of both the nominal and faulty subsystem equipped with the post-fault supervisor. The coordination architecture shown in Fig. 5.21 implements the refinement by associating each subsystem with a **coordination** supervisor \( S^{CO}_i \) \((i \in I)\). For the nominal subsystems, we reconfigure the control policies of the nominal subsystems and define \( M^{CO}_j = S^{CO}_j || G_j \) \((j \neq i)\) as the component module; whereas the component module \( M^{CO}_i \) of the post-fault subsystem \( S^{F}_i \| G^F_i \) is defined as a DFA over \( \Sigma^F_i \) such that \( L(M^{CO}_i) = P^F_i \left[ L(S^{CO}_i || S^F_i || G^F_i) \right] \). Let \( A_i^{CO} \) and \( A_j^{CO} \) denote the weakest assumptions (with respect to \( L \)) for \( M^{CO}_i \) and \( M^{CO}_j \) \((j \neq i)\), respectively. Furthermore, we assume an “infimally permissive” supervisor \( S^{inf}_i \) \((i \in I)\) to realize either \( \inf C_j(\{\epsilon\}) \) (computed
with respect to $G_j$ and $\Sigma_{j,uc} (j \neq i)$ or $\inf C_F^i(\{\epsilon\})$ (computed with respect to $S_i^F || G_i^F$ and $\Sigma_{i,uc}^F$). We refer to $M_i^{inf}$ and $A_i^{inf} (i \in I)$ as the component module and the weakest assumption DFAs when the coordination supervisor $S_i^{CO}$ reduces to $S_i^{inf}$, respectively. The following theorem derives the necessary and sufficient condition for the existence of the $S_i^{CO}$'s in terms of $A_i^{inf} (i \in I)$.

**Theorem 5.9.** There exists a coordination supervisor $S_i^{CO}$ for each subsystem $G_i (i \in I)$ such that $||\{i \in I\} M_i^{CO} \models L$ if and only if $L_m (||\{i \in I\}coA_i^{inf}) \subseteq L$.

**Proof.** Suppose that $L_m (||\{i \in I\}coA_i^{inf}) \subseteq L$, where $A_i^{inf}$ is the weakest assumption with respect to $M_i^{inf}$ and $L$. The soundness of the SYM-N proof rule implies that $||\{i \in I\} M_i^{CO} \models L$, which suggests that $S_i^{inf} (i \in I)$ suffice to be a satisfactory coordination supervisor.

Conversely, suppose that there exists a coordination supervisor $S_i^{CO}$ for either $S_i^F || G_i^F$ or $G_j (j \neq i)$ to jointly satisfy the global specification $L$. By definition, $L(M_i^{inf}) \subseteq L(M_i^{CO})$ always holds for each $i \in I$. Hence, we can write that

$$L(M_i^{inf} || A_i^{CO}) = L(M_i^{inf}) || L(A_i^{CO}) \subseteq L(M_i^{CO}) || L(A_i^{CO}) \subseteq L,$$

which implies that $\langle A_i^{CO} \rangle M_i^{inf} (L)$; that is, $A_i^{CO}$ is an appropriate assumption with respect to $M_i^{inf}$ and $L$. Since $A_i^{inf}$ is the weakest assumption, $A_i^{CO}$ is stronger than $A_i^{inf}$ and thus $L(A_i^{CO}) \subseteq L(A_i^{inf})$. Furthermore, since all the states in $A_i^{CO}$ and $A_i^{inf}$ are marked, $L(A_i^{CO}) \subseteq L(A_i^{inf})$ implies that $L_m(coA_i^{inf}) \subseteq L_m(coA_i^{CO})$. Therefore, we have that

$$L_m (||\{i \in I\}coA_i^{inf}) = ||\{i \in I\} L_m(coA_i^{inf}) \subseteq ||\{i \in I\} L_m(coA_i^{CO}) = L_m (||\{i \in I\}coA_i^{CO}) \subseteq L$$

where the last inclusion is enforced by the satisfaction of $||\{i \in I\} M_i^{CO} \models L$ and the completeness of the SYM-N rule. The proof is hence completed.

The intuition behind Theorem 5.9 states that if the infimally feasible behaviors per-
formed by each subsystem in the fault operation cannot jointly maintain the global specification, there is no other way to achieve a successful coordination.

Starting from the component modules $M_j$ ($j \neq i$) and $M_i^F$, the coordination procedure shown in Fig. 5.21 states as follows:

(i) Following the detection of actuator/sensor faults, the subsystem $G_i$ switches to the post-fault supervisor $S_i^F$ to satisfy the local safety requirement $L_i^{safe}$ and the post-fault component module $M_i^F$ is computed. With $M_i^F$ and $L$, the weakest assumption $A_i^F$ can be obtained and the satisfaction of Proposition 5.2 is justified. If Proposition 5.2 is satisfied, the fulfillment of $L$ is still maintained and no supervisor refinement is required.

(ii) If Proposition 5.2 fails to be satisfied, we apply Theorem 5.9 to determine whether or not the global specification can be accomplished by synthesizing appropriate coordination supervisor for each subsystem. If the coordination supervisors do exist, we apply the following counterexample-guided synthesis algorithm named SYN-CO for computing the coordination supervisor $S_i^{CO}$ ($i \in I$) for $S_i^F || G_i^F$ and $G_j$ ($j \neq i$).

The working procedure of the SYN-CO algorithm is explained as follows. First, $M_i^{CO}$ and $M_j^{CO}$ ($j \neq i$) are initialized to be $M_i^F$ and $M_j$ (lines 1 and 2), respectively. Whenever $L_m \left( \| i \in I CoA_i^{CO} \right) \subseteq L$ holds, the fulfillment of $L$ is assured automatically by Proposition 5.2 (lines 3 and 4). Otherwise, we aim at the synthesis of coordination supervisors and set $M_j^{CO}$ to be $G_j$ for the nominal subsystems (lines 5 to 8). If $L_m \left( \| i \in I CoA_i^{CO} \right) \not\subseteq L$, then a counterexample $c \in \Sigma^*$ that causes $\| i \in I M_i^{CO}$ to violate $L$ is returned by the assume-guarantee compositional verification procedure (lines 9 and 10). The counterexample is utilized to generate new local specification $L_i^{CO}$ for each subsystem by first eliminating $P_i(c)$ from the local behavior $L(M_i^{CO})$ (line 11) and then computing the supremal prefix-closed sublanguage of the resulting language (line 12) for each $i \in I$. The candidate coordination supervisor $S_i^{CO}$ is synthesized with respect to the updated $L_i^{CO}$ accordingly (lines 13 and 14). Finally, we update the component module for each subsystem in such a way that $L(M_i^{CO}) = P_i^F \left[ L(S_i^{CO} || S_i^F || G_i^F) \right]$ and $L(M_j^{CO}) = L(S_j^{CO} || G_j)$ if $j \neq i$, respectively (lines 15 to 17). The updated component modules are returned to the compositional

\[^3\text{SYN-CO stands for \textquoteleft	extquoteleft synthesis for coordination} \]
Algorithm 6: The SYN-CO Algorithm

Input: $M^F_i, G_j (j \neq i), \Sigma^F_i, \Sigma_j, \text{and } L$
Output: $M^{CO}_i$ and $S^{CO}_i (i \in I)$

1: Initialization: $M^{CO}_i \leftarrow M^F_i, M^{CO}_j \leftarrow M_j (j \neq i)$
2: Compute $A^{CO}_i$ such that $\langle A^{CO}_i \rangle M^{CO}_i \langle L \rangle (i \in I)$
3: if $L_m (\|_{i \in I} A^{CO}_i) \subseteq L$ then return $S^{CO}_i$ and $M^{CO}_i (i \in I)$
4: else
5: $M^{CO}_i \leftarrow M^F_i, M^{CO}_j \leftarrow G_j (j \neq i)$
6: Compute $A^{CO}_i$ such that $\langle A^{CO}_i \rangle M^{CO}_i \langle L \rangle (i \in I)$
7: end if
8: while $L_m (\|_{i \in I} A^{CO}_i) \not\subseteq L$ do
9: A counterexample $c \in \Sigma^*$ is returned by the compositional verification procedure
10: $L^\text{temp}_i \leftarrow L(M^{CO}_i) - P_i(c) (i \in I)$
11: $L^{CO}_i \leftarrow L^\text{temp}_i - (\Sigma^*_i - L^\text{temp}_i) \Sigma^*_i (i \in I)$
12: Compute a maximally permissive coordination supervisor $S^{CO}_i$ such that $S^{CO}_i \| G^F_i \models L^{CO}_i$
13: Compute a maximally permissive coordination supervisor $S^{CO}_j$ such that $S^{CO}_j \| G_j \models L^{CO}_j (j \neq i)$
14: Update $M^{CO}_i (i \in I)$
15: Compute $A^{CO}_i$ with respect to $M^{CO}_i$ and $L$
16: end while
return $S^{CO}_i$ and $M^{CO}_i (i \in I)$

verification to determine whether or not $\|_{i \in I} M^{CO}_i \models L$ until no more counterexample is generated. The correctness and termination of the SYN-CO algorithm is stated as follows.

**Theorem 5.10.** Given the component modules $M^F_i$ and $M_j (j \neq i)$, the SYN-CO algorithm terminates and correctly returns the coordination supervisors $S^{CO}_i (i \in I)$.

**Proof.** The termination of the SYN-CO algorithm holds due to the fact that during each iteration, each of the updated component module $M^{CO}_i (i \in I)$ possesses a finite number of states regardless of possible faults, and the deployment of the coordination supervisor $S^{CO}_i$ introduces a reduction of the number of states.

Furthermore, it has been shown that the compositional verification with the SYM-N proof rule always terminates and correctly reports whether or not $\|_{i \in I} M^{CO}_i \models L$ [132]; in other words, when no more counterexample is generated, we can conclude that $\|_{i \in I} M^{CO}_i \models L$ will not violate the specification $L$, i.e., the correctness of the SYN-CO algorithm can be achieved.
Remark 5.7. It is worth pointing out that although Theorem 5.10 ensures a correct post-fault coordination strategy among the subsystems, the SYN-CO algorithm may still come up with a trivial solution, i.e., \( \|i \in I \| M_i^{CO} = \emptyset \), which always solves Problem 5.1. To prevent this situation from emerging, we abandon the trivial solution returned by the SYN-CO algorithm; instead, we inherit the post-fault supervisor(s) \( S_i^F \) for \( G_i^F \) and the nominal supervisor \( S_j \) for \( G_j \) for all \( j \neq i \). In this case, the collective behaviors of \( G \) are given by \( \| j \in I - \{i\} \| M_j \| M_i^F \neq L \); nevertheless, from Theorem 5.8, the coordinated system is still tolerable in the sense that local safety of each subsystem is always assured while additional computation of the coordination supervisor is not required for the nominal subsystems.

We now study the coordination of the multi-robot system in Section 5.4.2 after the robot \( G_3 \) has switched to the post-fault supervisor \( S_{3\,to\,D_1,a}^{G_3} \) in Example 5.2.

Example 5.5. Under the supervision of \( S_{3\,to\,D_1,a}^{G_3} \), the robot \( G_3 \) should open \( D_1 \) from Room 1 since it is not able to enter Room 3. After the post-fault control reconfiguration, the assume-guarantee coordination framework shown in Fig. 5.21 is applied to coordinate \( G_3 \) with the other two robots. In this case, we can verify that the Proposition 5.2 cannot hold any more since both \( G_1 \) and \( G_3 \) will stay in Room 1 and \( D_1 \) cannot be opened jointly. Therefore, we need to design the coordination supervisors for \( G_1 \) and \( G_2 \). In particular, \( G_2 \) need not reconfigure the local control policies and \( S_2^{CO} = S_2 \), whereas the coordination supervisor \( S_1^{CO} \) can be synthesized for the robot \( G_1 \), as shown in Fig. 5.22

![Figure 5.22. The coordination supervisor \( S_1^{CO} \).](image-url)
In other words, the robot $G_1$ should enter Room 3 to open $D_1$ cooperatively with $G_3$ so that the global task can be achieved.

5.9 Conclusion

In this chapter, we present a fault tolerant coordination and control framework for distributed DESs that are composed of multiple subsystems. The proposed coordination and control framework ensures the accomplishment of the global specification in the presence of sensor and actuator faults. By introducing automata-theoretic methods to characterize the behaviors of each subsystem that is affected by various faults, appropriate post-fault supervisors are synthesized such that local safety can be ensured. In addition, an assume-guarantee coordination scheme is exploited to accomplish the global specification after the post-fault supervisor reconfiguration. The effectiveness of our proposed approach is demonstrated by an illustrative example.
CHAPTER 6

CONTINUOUS-CURVATURE PATH PLANNING OF NONHOLONOMIC MOBILE ROBOTS

6.1 Introduction

Mobile robots with various sensing and actuation capabilities represent a typical class of autonomous CPSs. In Chapter 4 and Chapter 5, we consider the motion planning of robotic systems from a cyber and logical perspective, and a decision-making procedure is developed for the accomplishment of the given specifications in the presence of either environmental uncertainties or system faults. Nevertheless, a successful design for a robotic system from a CPS perspective shall take not only logic behaviors, but also appropriate path plans and motion controllers into consideration, which corresponds to the planning level of Fig. 1.2 in Chapter 1. To implement the satisfactory paths in the physical level, we aim at applying dynamics-based control theory for the motion and path planning problem of robotic systems with (possibly) nonholonomic dynamics in this chapter.

The path planning problem of mobile robots and/or vehicles has been extensively investigated during the past few decades, with the consideration of various kinematic, dynamic and environmental constraints see, e.g., [31, 103] and the references therein. Despite studies focusing on robots with simple dynamics [10, 58], a vast majority of research efforts has been devoted to path planning of robots with nonholonomic dynamics [53, 54, 66, 102, 144] and particularly car-like models, due to the broad applications of car-like platforms, such as autonomous driving [67, 79] and parking maneuvers [184]. The classical path planning problem aims at steering a robot from a given initial configuration to
some final configurations while avoiding collision with any obstacles along the way. Many computationally efficient planning methods have been proposed in this context to compute such collision-free paths, see, e.g., [101, 102] and the references therein. Nevertheless, some of the aforementioned path planning algorithms require explicit description of the robot’s model as well as the environment, which may result in an excessive computational burden if the dimension of the robot’s model is high and in turn may severely limit the practical applications of these algorithms. To overcome the computational issues and to avoid the construction of the explicit representation of the model and environment, many research efforts have been devoted to the development of *sampling-based* path planning methods, which have been proved to be effective when dealing with high-dimensional models. Instead of using an explicit representation of the environment, sampling-based planning algorithms are developed on the foundation of a collision-checking module, which provides the planner with information about the feasibility of candidate trajectories, and connects a set of way-points sampled from the obstacle-free configuration space in order to build a graph (roadmap) of feasible trajectories. The solution of the original planning problem can then be extracted from the roadmap. Among various sampling-based path planning algorithms, the most commonly used methods in robotics are the Probabilistic Roadmap Method (PRM) [84] and the Rapidly-exploring Random Tree (RRT) [104]. The applications and improvements of these algorithms were investigated in [57]; however, the paths computed by the original algorithms in the literature were non-optimal. To address the limitations of these path planning algorithms, Karaman and Frazzoli [81] improved the performance of the RRT algorithm. The modified algorithm, named as RRT*, was guaranteed to converge to an optimal solution.

One of the major ingredients of the RRT and RRT* algorithms is the local steering between two way-points in the configuration space. The motion of a mobile robot with nonholonomic dynamics may be subject to constraints on both linear and angular velocities and accelerations, upon which feasible paths should be computed. In addition to
planning feasibility, another problem of interest is to search for optimal (minimum-length) paths among the family of feasible paths. It has been shown in the literature [54, 144] that minimum-length paths for mobile robots with car-like dynamics, also known as the Reeds-Shepp (RS) paths, are sequential composition of line segments and tangential circular arcs of the minimum turning radius of the robot. Nevertheless, it is worth pointing out that the curvature profiles corresponding to the classical RS paths are discontinuous at the connecting points between straight line segments and circular arcs, which renders the obtained paths jerky and therefore undesirable in practice, since at these way-points with discontinuous curvature, the mobile robot has to stop and perform stationary steering, leading to unnecessary time delay and extra wearing of tires [63, 184]. To deal with discontinuity of the curvature of the paths, many contributions have been made to constructing optimal and continuous-curvature (CC) paths for mobile robots. By exploiting Pontryagin’s Maximum Principle, the existence conditions for the optimal CC paths were derived in [17, 171], in which an optimal CC path was composed of clothoid curves and straight line segments [19, 183]; however, it was also shown in [159] that in the absence of obstacles, an optimal path that connected the initial and final way-points may possess infinite chattering, which implied an infinite number of clothoid curves. This discouraging discovery motivated the study of finding computationally efficient sub-optimal and non-chattering CC paths. The authors of [8] developed a numerically efficient planning scheme, which however was applicable only to forward-moving robots. A class of parametric curves named Bézier curves were utilized by Yang and Sukkarieh [197] to approximate clothoid curves and to construct CC paths; despite the computational convenience, such kind of Bézier curve paths lacked malleability when the degree of the curve increased. Alternatively, splines were also used for the computation of CC paths [13, 65]; nonetheless, optimality of the CC paths formed by splines was not studied thoroughly.

The aforementioned prior art, assuming obstacle-free environment, was typically employed to implement local steering in decomposition-based and sampling-based path plan-
ning [84, 102, 103]. It is understood that exploration of collision-free configuration space involves a huge amount of steering operations. In order to achieve real-time path planning, the computational efficiency of the underlying steering algorithms is of paramount importance. This chapter considers schemes to enable real-time computation of sub-optimal CC paths for steering car-like robots with boundedness on both path curvature and its time derivative. The sub-optimal CC paths admit the same driving patterns as the RS paths, and thus can be viewed as a generalization of the RS paths. Based on the $\mu$–tangency conditions [63], we establish existence conditions of the sub-optimal CC paths. The existence conditions can be used to determine whether a sub-optimal CC path with a specific driving pattern exists, upon which computationally efficient CC paths can be extracted. The proposed approach not only allows fast determination of the existence of a CC path, but also guarantees to construct the desired CC path.

The remainder of this chapter is organized as follows. Section 6.2 presents the kinematic model of a car-like mobile robot and formulates the optimal path planning problem. In Section 6.3, we review briefly the RS path patterns, as well as clothoid curves and $\mu$–tangency conditions in the literature. The clothoid curves and $\mu$–tangency conditions are further utilized in Section 6.4 to derive the existence conditions for CC paths with RS path types. Section 6.5 compares RS steering and the proposed CC steering by simulation examples. Concluding remarks are presented in Section 6.6.

6.2 Robot Models and Problem Formulation

In this section, we first introduce the kinematic model of a car-like robot that accounts for continuity of the curvature, and then we formulate the main objective of our work and outline our approach.
6.2.1 Kinematic Model of Car-like Mobile Robots

Fig. 6.1 illustrates the mobile robot with car-like motions, which is equipped with a front-fixed steering wheel and fixed parallel rear wheels. The reference point of the robot \( R \) is located at the mid of the rear wheels. The pose of the robot is uniquely described by a triple \((x, y, \theta)^T\) where \((x, y)^T\) represents the coordinates of \( R \) in the 2-D global coordinate frame and \( \theta \) is the orientation angle of the robot with respect to the positive \( x \)-axis of the frame. In order to ensure the continuity of the paths for the robot shown in Fig. 6.1, we assume that the robot moves with velocity \( v \), and the angle of the steering wheel is given by \( \phi \).

![Figure 6.1. A car-like robot with reference point \( R \) [63].](image)

Under the assumption of small velocities (which implies that the wheels of the robot can roll without slipping), a kinematic model suffices to characterize the motion of the mobile robot shown in Fig. 6.1. Specifically, in order to address potential discontinuity of
the path, [17] introduced the following kinematic model for a car-like mobile robot:

\[
\begin{pmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta} \\
\dot{\kappa}
\end{pmatrix} =
\begin{pmatrix}
\cos \theta \\
\sin \theta \\
\kappa \\
0
\end{pmatrix} v +
\begin{pmatrix}
0 \\
0 \\
0 \\
1
\end{pmatrix} \sigma,
\]  

(6.1)

which extended the 3-dimensional model in [54] by taking the curvature variable \( \kappa \) as the fourth parameter in addition to \((x, y, \theta)^T\). The control inputs of the system (6.1) are given by the 2-dimensional vector \( u = (v, \sigma)^T \), where \( v \) is the driving velocity of the rear wheels and \( \sigma \) the steering rate of the robot. The relationship among \( \phi \), \( \kappa \) and \( \sigma \) is established as follows:

\[
\kappa = \frac{\tan \phi}{b}, \sigma = \dot{\kappa} = \frac{\dot{\phi}}{b \cos^2 \phi}.
\]  

(6.2)

6.2.2 Optimal Path Planning Problem

We assume that both forward and backward motions are allowed for the robot (6.1). To pursue an optimal path that connects a given initial configuration and a final one, it is assumed that the driving velocity is bounded, whereas the angle of the steering wheel is subject to mechanical constraints, i.e.,

\[
|v| \leq v_{max}, |\phi| \leq \phi_{max},
\]  

(6.3)

which implies that the curvature \( \kappa \) should also obey the constraint

\[
|\kappa| \leq \kappa_{max} := \frac{\tan \phi_{max}}{b}.
\]  

(6.4)
Furthermore, the steering rate $\sigma$ is also assumed to be bounded, i.e.,

$$|\sigma| \leq \sigma_{\text{max}}.$$  \hspace{1cm} (6.5)

Based on the constraints (6.3), (6.4) and (6.5), the set of admissible control inputs to the mobile robot (6.1) is hence given by

$$\mathcal{U} = \{ u \in \mathcal{U} : u(t) \in U, t \in [0, T_f] \},$$ \hspace{1cm} (6.6)

where $T_f > 0$, $\mathcal{U}$ is the set of all measurable functions defined over $[0, T_f]$, and the domain $U$ is given by $U := [-v_{\text{max}}, v_{\text{max}}] \times [-\sigma_{\text{max}}, \sigma_{\text{max}}]$.

Next, we proceed to the statement of the optimal path planning problem for the mobile robot. Given an initial configuration $q_0 = (x_0, y_0, \theta_0, \kappa_0)^T$, a final configuration $q_f = (x_f, y_f, \theta_f, \kappa_f)^T$ and the robot model (6.1) with constraints (6.3), (6.4) and (6.5), we need to find a feasible and optimal path between $q_0$ and $q_f$. By normalizing the velocity of the robot, i.e., $|v| = 1$, planning a shortest path that connects two configurations is equivalent to solving the following optimal control problem.

**Problem 6.1 (Optimal Path Planning Problem).** Given a robot with dynamics (6.1) and the cost functional

$$J(u) = \int_0^{T_f} dt = T_f,$$ \hspace{1cm} (6.7)

determine the optimal control input $u^* \in \mathcal{U}$ with $u^* : [0, T_f] \mapsto U$ and the associated optimal state trajectory $q^* : [0, T_f] \mapsto \mathbb{R}^2 \times S^1 \times \mathbb{R}$ (where $S^1$ is the circle group), such that the optimal control input $u^*$ and the optimal state trajectory $q^*$ jointly satisfy

(i) **Boundary conditions:**

$$q^*(0) = q_0 = (x_0, y_0, \theta_0, \kappa_0)^T$$ \hspace{1cm} (6.8)

and

$$q^*(T_f) = q_f = (x_f, y_f, \theta_f, \kappa_f)^T;$$ \hspace{1cm} (6.9)
(ii) **Feasibility:** \( \forall t \in [0, T_f], \mathbf{q}^*(t) \in \mathbb{R}^2 \times S^1 \times [-\kappa_{max}, \kappa_{max}] \).

(iii) **Optimality:** The optimal control \( u^* \) minimizes the cost functional \( J(u) \).

Similar to [63], we assume that \( q_0, q_f \) have null-curvature configurations, i.e., \( \kappa_0 = \kappa_f = 0 \). The results presented in this chapter can be also generalized to connection of non-zero curvature configurations. In this chapter, we assume without loss of generality that the initial configuration \( q_0 \) of Problem 6.1 is the origin of the global frame, i.e., \( q_0 = (0, 0, 0, 0)^T \).

As mentioned in Section 6.1, solving Problem 6.1 via classical optimal control techniques may result in paths that admit infinite chattering. In order to construct a non-chattering CC path from \( q_0 \) to the null-curvature final configuration \( q_f \), our path planning scheme is composed of the following three steps: first, we ignore the boundedness of the derivative of the curvature and generate all feasible paths that belong to the class of optimal paths proposed in [144] (termed as RS paths); next, we convert each candidate path to a corresponding continuous-curvature one by introducing clothoid curves and turns; finally, we use the path of minimum-length as our sub-optimal solution. Such conversion is accomplished by integrating clothoid-based planar curves [17, 159] with additional geometric analysis; whereas sub-optimality study of the proposed paths will also be presented.

### 6.3 Continuous Curvature Paths via Line Segments and Clothoids

Before proceeding to the solution of Problem 6.1, we first review classes and patterns of the Reeds-Shepp (RS) paths, and basic concepts that relate to clothoid turns and \( \mu \)-tangency conditions in this section. Interested readers are referred to [54], [144] and [63] for more details.
6.3.1 The Class of Reeds-Shepp’s Paths

When the bound of the steering rate $\sigma_{\text{max}} \to \infty$, Problem 6.1 reduces to the Dubins problem [54] (when only forward motion is allowed) or the Reeds-Shepp problem [144] (when both forward and backward motions are allowed), whose solutions are composed of straight line segments and circular arcs of radius $\kappa_{\text{max}}^{-1}$.

**TABLE 6.1**

**RS PATH CLASSES**

<table>
<thead>
<tr>
<th>Path Classes</th>
<th>Path Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CSC - 1$</td>
<td>$L^+ S^+ L^+, L^- S^- L^-, R^+ S^+ R^+, R^- S^- R^-$</td>
</tr>
<tr>
<td>$CSC - 2$</td>
<td>$L^+ S^+ R^+, L^- S^- R^-, R^+ S^+ L^+, R^- S^- L^-$</td>
</tr>
<tr>
<td>$C</td>
<td>C</td>
</tr>
<tr>
<td>$C</td>
<td>C C$</td>
</tr>
<tr>
<td>$C</td>
<td>C$</td>
</tr>
<tr>
<td>$C C u</td>
<td>C u C$</td>
</tr>
<tr>
<td>$C</td>
<td>C C_u</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
<tr>
<td>$C</td>
<td>C_2^\frac{1}{2}</td>
</tr>
</tbody>
</table>

189
It has been proved in [144] that for any pair of initial and final configurations, the RS paths can be categorized into 12 classes, and admit a total of 48 path profiles. All classes and patterns are summarized in Table 6.1, where $C$ stands for a circular curve of the minimum turning radius of the robot and $S$ represents a straight line segment. In addition, the symbols $L$ and $R$ in each path pattern specify left and right turns, respectively; and the superscripts $+$ and $-$ denote a forward and backward motion, respectively. Finally, the subscript associated with a $L$ or $R$ symbol in a path pattern in Table 6.1 denotes the (absolute) angular value of a certain circular arc, and $|$ represents a cusp [144] emerging in the path when change of the direction of the robot’s motion takes place. Computation procedure of an RS path that is able to connect $q_0$ and $q_f$ (ignoring the curvature parameter $\kappa$) can be found in [144] and is omitted here due to space limit.

6.3.2 Clothoid Curves and Clothoid Turns

![Path Curvature]

Figure 6.2. The curvature profile along with an RS path [63].

The RS path between two configurations is the shortest path that is composed of straight line segments and circular arcs of minimum radius $\kappa_{\text{max}}^{-1}$. The curvature profile along with a possible $L^+S^+R^+$ RS path is depicted in Fig. 6.2. Clearly, the continuity of the curvature
profile fails to hold when the robot switches from a circular arc to a straight line segment, or vice versa. To ensure the continuity of curvature along the path, we inherit the idea of utilizing the clothoid turns and curves in [63].

A clothoid is a curve whose curvature is an affine function of the arc length \( s \) of the curve itself, i.e.,

\[
\kappa(s) = \kappa(0) + \sigma s,
\]

where \( \sigma \) is termed as the *sharpness* of the clothoid. A clothoid turn (CT) from a starting point \( q_s = (x_s, y_s, \theta_s, 0)^T \) to an end point \( q_g = (x_g, y_g, \theta_g, 0)^T \) is characterized by its *deflection* \( \delta = (\theta_g - \theta_s) \mod 2\pi \). Without loss of generality, we assume that the robot \((6.1)\) is moving along a forward and left clothoid turn from \( q_s = (0, 0, 0, 0) \) and the deflection of the CT satisfies that \( 0 \leq \delta < 2\pi \).

Let \( \delta_c = \frac{\kappa_{\text{max}}^2}{2\sigma_{\text{max}}} \). We first consider the case in which \( 2\delta_c \leq \delta \leq 2\delta_c + \pi \). In this case, the mobile robot first follows a clothoid with sharpness \( \sigma_{\text{max}} \) until the curvature \( \kappa \) increases linearly from 0 to \( \kappa_{\text{max}} \). According to \((6.1)\), the configuration \( q(s) \) of the robot at distance \( s \) from the starting point \( q_s = (0, 0, 0, 0)^T \) is given by

\[
q(s) = \begin{pmatrix}
    x(s) \\
    y(s) \\
    \theta(s) \\
    \kappa(s)
\end{pmatrix} = \begin{pmatrix}
    \sqrt{\pi/\sigma_{\text{max}}} C_f \left( s/\sqrt{\pi/\sigma_{\text{max}}} \right) \\
    \sqrt{\pi/\sigma_{\text{max}}} S_f \left( s/\sqrt{\pi/\sigma_{\text{max}}} \right) \\
    \sigma_{\text{max}} s^2/2 \\
    \sigma_{\text{max}} s
\end{pmatrix},
\]

where

\[
C_f(s) = \int_0^s \cos \frac{\pi}{2} \tau^2 d\tau, \quad S_f(s) = \int_0^s \sin \frac{\pi}{2} \tau^2 d\tau,
\]

are the Fresnel cosine and sine integrals, respectively. This clothoid ends at the intermedi-
ate configuration \( q_1 \), whose components are obtained as

\[
q_1 = \begin{pmatrix}
x_1 \\
y_1 \\
\theta_1 \\
\kappa_1
\end{pmatrix} = \begin{pmatrix}
\sqrt{\frac{\pi}{\sigma_{\max}} C_f \left( \sqrt{\frac{\kappa_{\max}^2}{(\pi \sigma_{\max})}} \right)} \\
\sqrt{\frac{\pi}{\sigma_{\max}} S_f \left( \sqrt{\frac{\kappa_{\max}^2}{(\pi \sigma_{\max})}} \right)} \\
\delta_c \\
\kappa_{\max}
\end{pmatrix}.
\] (6.13)

From \( q_1 \), the robot enters a circular arc of radius \( \kappa_{\max}^{-1} \) and of an angular value \( \theta = \delta - 2\delta_c \). The coordinates of the center \( \Omega \) of the underlying circle is given by

\[
\begin{pmatrix}
x_\Omega \\
y_\Omega
\end{pmatrix} = \begin{pmatrix}
x_1 - \kappa_{\max}^{-1} \sin \theta_1 \\
y_1 + \kappa_{\max}^{-1} \cos \theta_1
\end{pmatrix}.
\]

The circular arc ends at a second intermediate configuration \( q_2 = (x_2, y_2, \delta - \delta_c, \kappa_{\max})^T \) for some \( x_2 \) and \( y_2 \). After the robot leaves the circular arc, it follows another clothoid with sharpness \( -\sigma_{\max} \) until it reaches the goal configuration \( q_g = (x_g, y_g, \theta_g, 0)^T \) while the curvature reduces to zero. The clothoid turn \( CT \) of deflection \( 2\delta_c \leq \delta \leq 2\delta_c + \pi \) is then formed by:

(i) a clothoid starting at \( q_0 \) with sharpness \( \kappa_{\max} \) and length \( \kappa_{\max}/\sigma_{\max} \);
(ii) a circular arc of radius \( \kappa_{\max}^{-1} \) and of angular value \( \delta - 2\delta_c \);
(iii) a second clothoid with sharpness \( -\kappa_{\max} \) and length \( \kappa_{\max}/\sigma_{\max} \).

The locus of the starting and the goal configurations forms a circle whose center is at \( (x_\Omega, y_\Omega)^T \). This circle, which is named as a CC Circle of a left and forward CT starting from \( q_s \), is written as \( C_i^+(q_s) \). The CC circles corresponding to CTs with different directions are also depicted in Fig. 6.3.
Furthermore, the radius $R_\Omega$ of the CC Circle $C_i^+(q_s)$ and the angle $\mu$ between the orientation of $q_s$ and the tangent of $C_i^+(q_s)$ at $q_g$ are obtained as follows, respectively:

$$
\begin{pmatrix}
R_\Omega \\
\mu
\end{pmatrix} = \begin{pmatrix}
\sqrt{x_\Omega^2 + y_\Omega^2} \\
\arctan(x_\Omega/y_\Omega)
\end{pmatrix}.
$$

(6.14)

Note that the parameters of the CC Circle can be computed independent of the goal configuration $q_g$.

Similarly, when $2\delta_c + \pi \leq \delta < 2\pi$, we allow backward motion of the mobile robot within a single $CT$ so that the length of the clothoid turn can be reduced. In particular, the desired clothoid turn is composed of

(i) a clothoid starting at initial configuration $q_i$ with sharpness $\kappa_{max}$ and end at $q_1$:

Figure 6.3. CC Circles $C_i^+(q_s)$, $C_r^+(q_s)$, $C_i^-(q_s)$ and $C_r^-(q_s)$ [63].
(ii) a backward circular arc of radius $\kappa_{\max}^{-1}$ and of angle $\delta - 2\delta_c - 2\pi$ starting at configuration $q_1$ and ending at some configuration $q_2$;

(iii) a second clothoid starting at configuration $q_2$ with sharpness $-\kappa_{\max}$ and length $\kappa_{\max}/\sigma_{\max}$.

It can be verified that the starting configuration $q_s$ and the goal configuration $q_g$ are both located on the CC Circle $C_i^+(q_s)$ in this case.

Next, in the case where $0 < \delta < 2\delta_c$, the angular value of the circular arc in the CT reduces to 0 and hence the CT is composed of

(i) a clothoid of sharpness $\sigma \leq \sigma_{\max}$ starting from $q_s$;

(ii) a symmetric clothoid of sharpness $-\sigma$ ending at $q_g$.

To pursue computationally convenient path planning in the following sections, we require that $(x_g, y_g)^T$ be also located at $C_i^+(q_s)$. Toward this end, the sharpness $\sigma$ is computed as

$$\sigma = \frac{\pi \left[ \cos(\delta/2)C_f(\sqrt{\delta/\pi}) + \sin(\delta/2)S_f(\sqrt{\delta/\pi}) \right]^2}{R_{\Omega}^2 \sin^2(\delta/2 + \mu)}, \quad (6.15)$$

where $R_{\Omega}$ and $\mu$ are the same in (6.14), and the arc length of each clothoid arc is $\sqrt{\delta/\sigma}$.

Finally, for the case $\delta = 0$, the clothoid turn reduces to a straight line segment of length $2R_{\Omega} \sin \mu$ that starts from $q_s$, so as to ensure that $q_g$ belongs to $C_i^+(q_s)$.

6.3.3 The $\mu$-Tangency Conditions for Clothoid Turns

To explore the existence conditions for CC paths that are composed of CTs and straight line segments, the $\mu$-tangency conditions among clothoid turns and straight line segments are studied in this subsection. Recall that in the RS path planning paradigm, a connection between a circular arc and a line segment is needed at a configuration $q$ if and only if the line segment is tangent to the circle associated with $q$. When a straight line segment is connected to a CT, the line segment must cross the CC circle associated with the configuration $q$ and make an angle of $\mu$. We deem this condition as a $\mu$-tangency condition.
rather than the conventional tangency condition in forming RS paths. For instance, for an \( L^+S^+ \) motion starting from some configuration \( q \) of the mobile robot with model (6.1), the \( \mu \)-tangency between a straight line segment and a \( CT \) is illustrated in Fig. 6.4, where the circle denotes the CC Circle \( C_i^+(q) \).

![Figure 6.4. \( \mu \)-tangency between line segments and CC circles.](image)

In addition, the class of the RS paths also admits the case in which two circles with radius \( \kappa_{\text{max}}^{-1} \) corresponding to the two configurations can be connected by a third circle that is tangent to both of them. Nevertheless, in our proposed approach of constructing CC paths, the \( \mu \)-tangency conditions between two consecutive \( CTs \) are presented in a different way. We first investigate the case in which the direction of the robot’s motion remains the same, i.e., there is no cusp in the path. We take a path of the form \( L^+R^+ \) that connects two configurations \( q_1 \) and \( q_2 \) as an example to explain the \( \mu \)-tangency conditions. As shown in Fig. 6.5, the \( \mu \)-tangency conditions suggest that the CC circle \( C_i^+(q_1) \) be tangent to \( C_r^-(q_2) \). The intermediate configuration \( q_3 \) serves not only the final configuration of the left and forward \( CT \) from \( q_1 \) but the initial configuration of the right and forward \( CT \) that ends at \( q_2 \) as well. Specifically, angles between the orientation angle of \( q_3 \) and tangent vectors of both \( C_i^+(q_1) \) and \( C_r^-(q_2) \) are \( \mu \).
As shown in Fig. 6.5, the $\mu$-tangency condition for a $CC$ path pattern suggests that the distance between the center $\Omega_1$ of the CC Circle $C^+_l(q_1)$ and the center $\Omega_2$ of the CC Circle $C^-_r(q_2)$ be given by

$$L(\Omega_1\Omega_2) = 2R_\Omega.$$

(6.16)

Finally, we take a path of the form $L^+R^-$ that connects two configurations $q_1$ and $q_2$ as an example to explain $\mu$-tangency conditions when the direction of motion changes. This case is illustrated by Fig. 6.6, where a cusp emerges on the path. Let $q_3$ be the configuration of the cusp, and $q_3$ must be located at one of the two intersecting points of the CC Circles $C^+_l(q_1)$ and $C^+_r(q_2)$. The $\mu$-tangency condition requires that the orientation of $q_3$ make an angle of $\mu$ with respect to both CC Circles. This type of $\mu$-tangency condition validates if and only if the orientation of the configuration $q_3$ is vertical to $\Omega_1\Omega_2$.

As shown in Fig. 6.6, the $\mu$-tangency condition for a $CC$ path pattern suggests that the distance between the center $\Omega_1$ of the CC Circle $C^+_l(q_1)$ and the center $\Omega_2$ of the CC Circle $C^-_r(q_2)$ be given by

$$L(\Omega_1\Omega_2) = 2R_\Omega \cos \mu.$$

(6.17)
Figure 6.6. $\mu$-tangency between CC circles (with a cusp).

6.4 Existence Conditions for CC Paths with RS Path Patterns

This section derives existence conditions of the CC paths that admit the RS path classes in Table 6.1 by exploiting geometric analysis. For brevity of presentation, we only consider the CC paths starting with a left and forward CT $L^+$ that connect the null-curvature initial and final configurations $q_0 = (0, 0, 0, 0)$ and $q_f = (x_f, y_f, \theta_f, 0)$ (henceforth deemed as canonical paths); whereas the existence conditions for the rest members shown in Table 6.1 can be derived in a similar manner.

Furthermore, it is worth pointing out that the computing a $CSC_{\pi}^+ | C - 1$ or $CSC_{\pi}^+ | C - 2$ path from $q_0$ to $q_f$ is equivalent to constructing a $C|C_{\pi}^+ SC - 1$ or $C|C_{\pi}^+ SC - 2$ path from $q_f$ to $q_0$, respectively. Therefore, in this section, we present the existence conditions for the other 10 path classes listed in Table 6.1: $CSC - 1$, $CSC - 2$, $C|C|C$, $C|CC$, $CC|C$, $CC_u|C_uC$, $C|C_uC_u|C$, $C|C_{\pi}^+ SC - 1$, $C|C_{\pi}^+ SC - 2$ and $C|C_{\pi}^+ SC_{\pi}^+ | C$. For convenience of presentation, we introduce the validity of paths to simplify computation.

**Definition 6.1 (Valid Paths).** A canonical continuous-curvature path is said to be valid if:

(i) for all CT's that are admitted by the CC path, $0 \leq \delta < 2\delta_c + \pi$ holds for a positive deflection and $-2\delta_c - \pi \leq \delta < 0$ for a negative deflection;
(ii) the direction of the mobile robot’s motion does not change in any CTs that are included in the path;

(iii) the total length of the path $L > 0$.

6.4.1 The $CSC - 1$ Paths

We start our study of CC path planning by proposing a geometric reasoning of an $L^{+}S^{+}L^{+}$ canonical path that connects $q_0$ and $q_f$. To this regard, we first use the CC Circles $C_{i}^{+}(q_0)$ for $q_0$ and $C_{i}^{-}(q_f)$ for $q_f$ to characterize the final configuration $q_1$ of the first left clothoid turn, and the initial configuration $q_2$ of the second left CT, respectively. The path planning scenario is shown in Fig. 6.7. Let $\theta$ denote the angle between the straight line $\Omega_1\Omega_2$ and the positive $x$-axis.

![Diagram of geometric planning of $CSC - 1$ paths.](image)

Figure 6.7. Geometric planning of $CSC - 1$ paths.

To assure the $\mu$-tangency between $C_{i}^{+}(q_0)$ and $q_1q_2$, and between $q_1q_2$ and $C_{i}^{-}(q_f)$, a straightforward geometric analysis shows that the straight line $q_1q_2$ is parallel to $\Omega_1\Omega_2$;
therefore, the canonical path that connects $q_0$ and $q_f$ can be written as $L^+_{\delta_1}S^+L^+_{\delta_2}$, and is composed of:

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1$: since $q_1q_2$ is parallel to $\Omega_1\Omega_2$, the orientation of the final configuration $q_1$ of $CT_1$ is identical to $\theta$, we have the deflection $\delta_1 = \theta$;

(ii) a straight line segment $q_1q_2$ of length

$$l = L(q_1q_2) = L(\Omega_1\Omega_2) - 2R_\Omega \sin \mu; \quad (6.18)$$

(iii) a second left and forward clothoid turn $CT_2$ with deflection $\delta_2$: since the orientation of the initial configuration $q_2$ of $CT_2$ is also $\theta$, the deflection of $CT_2$ shall be $\delta_2 = \theta_f - \theta$.

**Existence Conditions** It is clear that such an $L^+S^+L^+$ path is valid if and only if: (i) $L(\Omega_1\Omega_2) \geq l = 2R_\Omega \sin \mu$; (ii) $\theta_f \geq \theta$.

6.4.2 The $CSC - 2$ Paths

![Diagram](image)

Figure 6.8. Geometric planning of $CSC - 2$ paths.
Similar geometric analysis can be utilized to generate an $L^+S^+R^+$ canonical path that connects $q_0$ and $q_f$, except that we use the CC Circle $C^-_r(q_f)$ rather than $C^-_l(q_f)$ to characterize the intermediate configuration $q_2$. In this cases, the straight line $q_1q_2$ crosses $\Omega_1\Omega_2$, as shown in Fig. 6.8.

Let $\alpha$ denote the angle between $q_1q_2$ and $\Omega_1\Omega_2$. According to the $\mu$-tangency condition between a straight line segment and a $CT$, it is clear to see that $\alpha$ can be determined by

$$\sin \alpha = \frac{R_{\Omega} \cos \mu}{L(\Omega_1\Omega_2)/2}.$$ (6.19)

The desired canonical path $L^+S^+R^+$ can therefore be generated with the help of (6.19). In fact, the canonical path that connects $q_0$ and $q_f$ can be written as $L_1^+S_1^+R_1^+$, and is composed of:

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1 = \theta + \alpha$;

(ii) a straight line segment $q_1q_2$ of length

$$l = L(q_1q_2) = \sqrt{L^2(\Omega_1\Omega_2) - (2R_{\Omega} \cos \mu)^2} - 2R_{\Omega} \sin \mu;$$ (6.20)

(ii) a second right and forward clothoid turn $CT_2$ with deflection $\delta_2 = \theta_f - \theta - \alpha$.

**Existence Conditions** It is clear that such an $L^+S^+R^+$ path is valid if and only if: (i) $\alpha$ is well-defined, i.e., $L(\Omega_1\Omega_2) \geq 2R_{\Omega}$; (ii) both $\delta_1$ and $\delta_2$ are valid deflections according to Definition 6.1.

6.4.3 The $C|C|C$ Paths

We now investigate the conditions under which an $L^+R^-L^+$ path exists to connect $q_0$ and $q_f$. The geometric scenario of this case is demonstrated in Fig. 6.9.
Figure 6.9. Geometric planning of $C|C|C$ paths.

The CC Circles $C^+_l(q_0)$ for $q_0$ and $C^-_l(q_f)$ for $q_f$ are employed to characterize the final configuration $q_1$ of the first left and forward $CT$, and the initial configuration $q_2$ of the second left and forward $CT$. As shown in Fig. 6.9, the CC Circle $C^-_r(q_1)$ must coincide with $C^+_r(q_2)$ in order to achieve the validity of the $L^+R^-L^+$ path. Let $\Omega_3$ denote the center of the this CC Circle, and let $\alpha$ denote the angle between $\Omega_1\Omega_2$ and $\Omega_1\Omega_3$. The $\mu$-tangency conditions (6.17) between $C^+_l(q_0)$ and $C^-_r(q_1)$ and between $C^-_r(q_1)$ and $C^-_l(q_f)$ suggest that

$$L(\Omega_1\Omega_3) = L(\Omega_2\Omega_3) = 2R_\Omega \cos \mu. \quad (6.21)$$

Thus, applying law of cosines in the triangle $\Omega_1\Omega_2\Omega_3$ yields

$$\cos \alpha = \frac{L^2(\Omega_1\Omega_2) + (2R_\Omega \cos \mu)^2 - (2R_\Omega \cos \mu)^2}{2 \cdot L(\Omega_1\Omega_2) \cdot 2R_\Omega \cos \mu}$$

$$= \frac{L(\Omega_1\Omega_2)}{4R_\Omega \cos \mu}. \quad (6.22)$$
It then can be inferred from the auxiliary angle $\alpha$ (6.22) that the coordinate of $\Omega_3$ can be given by

$$\Omega_3 = \begin{pmatrix} x_{\Omega_3} \\ y_{\Omega_3} \end{pmatrix} = \begin{pmatrix} x_{\Omega_1} + 2R_{\Omega} \cos \mu \cos(\theta + \alpha) \\ y_{\Omega_1} + 2R_{\Omega} \cos \mu \sin(\theta + \alpha) \end{pmatrix}.$$ \hspace{1cm} (6.23)

Let $\theta_1$ denote the angle between $\Omega_2 \Omega_3$ and the positive $x$-axis, which can be determined by the coordinates of $\Omega_2$ and $\Omega_3$. The path $L_{\delta_1}^{+} R_{\delta_2}^{-} L_{\delta_3}^{+}$ can then be computed according to Fig. 6.9 and is composed of

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1$: the $\mu$-tangency condition (6.17) between $C_{l}^{+}(q_0)$ and $C_{r}^{-}(q_1)$ asserts that the orientation of $q_1$ is $\delta_1 = \theta + \alpha + \frac{\pi}{2}$;

(ii) a right and backward clothoid turn $CT_2$ with deflection $\delta_2$: since the orientation of $q_2$ is $\theta_1 - \frac{\pi}{2}$ according to the $\mu$-tangency between $C_{l}^{-}(q_f)$ and $C_{r}^{-}(q_1)$, the deflection of $CT_2$ is hence given by $\delta_2 = \theta_1 - \theta - \alpha - \pi$;

(iii) a second left and forward clothoid turn $CT_3$ with deflection $\delta_3 = \theta_f - \theta_1 + \frac{\pi}{2}$.

**Existence Conditions** (i) $\delta_1$, $\delta_2$ and $\delta_3$ are all valid with respect to Definition 6.1; (ii) the triangle $\Omega_1 \Omega_2 \Omega_3$ should be valid, i.e.,

$$0 \leq L(\Omega_1 \Omega_2) \leq 4R_{\Omega} \cos \mu.$$ \hspace{1cm} (6.24)

### 6.4.4 The $C|CC$ Paths

The canonical $C|CC$ path, $L^+ R^- L^-$, can be processed in the same way as the $C|C|C$ path. As shown in Fig. 6.10, after computing the center $\Omega_1$ of $C_{l}^{+}(q_0)$ and $\Omega_2$ of $C_{l}^{+}(q_f)$, we aim to determine the intermediate configurations $q_1$, $q_2$, and the center, denoted as $\Omega_3$, of the CC Circle $C_{r}^{-}(q_1)$.

The $\mu$-tangency (6.17) between $C_{l}^{+}(q_0)$ and $C_{r}^{-}(q_1)$ implies that

$$L(\Omega_1 \Omega_3) = 2R_{\Omega} \cos \mu,$$ \hspace{1cm} (6.25)
whereas the $\mu$-tangency (6.16) between $C_r^-(q_1)$ and $C_l^+(q_f)$ implies that

$$L(\Omega_2\Omega_3) = 2R_\Omega.$$  \hfill (6.26)

Let $\alpha$ denote the angle between $\Omega_1\Omega_2$ and $\Omega_1\Omega_3$. The law of cosines suggests that, in the triangle $\Omega_1\Omega_2\Omega_3$,

$$\cos \alpha = \frac{L^2(\Omega_1\Omega_2) + (2R_\Omega \cos \mu)^2 - (2R_\Omega)^2}{2 \cdot L(\Omega_1\Omega_2) \cdot 2R_\Omega \cos \mu} = \frac{L^2(\Omega_1\Omega_2) - (2R_\Omega \sin \mu)^2}{4L(\Omega_1\Omega_2)R_\Omega \cos \mu}. \hfill (6.27)$$

As shown in Fig. 6.10, the coordinates of $\Omega_3$ can thus be represented by (6.23), with $\alpha$ computed by (6.27). Let $\theta_1$ denote the angle between $\Omega_2\Omega_3$ and the positive $x$-axis of the global frame. The desired canonical path $L_{\delta_1}^+ R_{\delta_2}^- L_{\delta_3}^+$ consists of

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1 = \theta + \alpha + \frac{\pi}{2}$;

(ii) a right and backward clothoid turn $CT_2$ with deflection $\delta_2$: since the orientation of
$q_2$ is $\theta_1 - \frac{\pi}{2} - \mu$, the deflection of $CT_2$ is $\delta_2 = \theta_1 - \theta - \alpha - \mu - \pi$;

(iii) a left and backward clothoid turn $CT_3$ with deflection $\delta_3$: $\delta_3 = \theta_f - \theta_1 + \frac{\pi}{2} + \mu$, which guarantees that the final configuration of $CT_3$ is identical to $q_f$.

**Existence Conditions** (i) $\delta_1$, $\delta_2$ and $\delta_3$ satisfy Definition 6.1; (ii) the triangle $\Omega_1\Omega_2\Omega_3$ should be valid, i.e.,

$$2R_{\Omega 1}(1 - \cos \mu) \leq L(\Omega_1\Omega_2) \leq 2R_{\Omega 1}(1 + \cos \mu).$$

(6.28)

6.4.5 The $CC|C$ Paths

We slightly modify the computation procedure of $C|CC$ paths to adapt for the planing of $CC|C$ paths. The geometric setup is illustrated in Fig. 6.11, where the center of $C_i^r(q_0)$ is $\Omega_1$ and the center of $C_i^r(q_f)$ is $\Omega_2$.

Figure 6.11. Geometric planning of $CC|C$ paths.
Let \( q_1 \) and \( q_2 \) be the two intermediate configurations along with the path. To determine the center \( \Omega_3 \) of the CC Circle \( C_r^+(q_1) \), we apply law of cosines in the triangle \( \Omega_1 \Omega_3 \Omega_2 \) to obtain the angle \( \alpha \):

\[
\cos \alpha = \frac{L^2(\Omega_1 \Omega_2) + (2R_\Omega)^2 - (2R_\Omega \cos \mu)^2}{2 \cdot L(\Omega_1 \Omega_2) \cdot 2R_\Omega} = \frac{L^2(\Omega_1 \Omega_2) + (2R_\Omega \sin \mu)^2}{4L(\Omega_1 \Omega_2)R_\Omega},
\]

which yields the coordinate of \( \Omega_3 \)

\[
\Omega_3 = \begin{pmatrix} x_{\Omega_3} \\ y_{\Omega_3} \end{pmatrix} = \begin{pmatrix} x_{\Omega_1} + 2R_\Omega \cos(\theta - \alpha) \\ y_{\Omega_1} + 2R_\Omega \sin(\theta - \alpha) \end{pmatrix}.
\]

Let \( \theta_1 \) denote the angle between \( \Omega_2 \Omega_3 \) and the positive \( x \)-axis of the global frame. The path \( L_{\delta_1}^+ R_{\delta_2}^+ L_{\delta_3}^- \) can be computed as the concatenation of

(i) a left and forward clothoid turn \( CT_1 \) with deflection \( \delta_1 \): the \( \mu \)-tangency (6.16) between \( C_I^+(q_0) \) and \( C_r^+(q_1) \) implies that the orientation of \( q_1 \) is \( \theta - \alpha + \frac{\pi}{2} - \mu \), which implies that \( \delta_1 = \theta - \alpha + \frac{\pi}{2} - \mu \);

(ii) a right and forward clothoid turn \( CT_2 \) with deflection \( \delta_2 \): since the \( \mu \)-tangency (6.17) between \( C_r^+(q_1) \) and \( C_I^+(q_f) \) implies that the orientation of \( q_2 \) is \( \theta_1 - \frac{\pi}{2} \), thus \( \delta_2 = \theta_1 - \theta + \alpha - \mu - \pi \);

(iii) A left and backward clothoid turn \( CT_3 \) with deflection \( \delta_3 = \theta_f - \theta_1 + \frac{\pi}{2} \).

**Existence Conditions** (i) \( \delta_1, \delta_2 \) and \( \delta_3 \) all satisfy Definition 6.1; (ii) the triangle \( \Omega_1 \Omega_3 \Omega_2 \) be valid, i.e.,

\[
2R_\Omega(1 - \cos \mu) \leq L(\Omega_1 \Omega_2) \leq 2R_\Omega(1 + \cos \mu).
\]

6.4.6 The \( CC_u|C_u \) Paths

Here we concentrate on the geometric computation of canonical \( CC_u|C_u \) paths, namely, \( L^+ R_u^+ L_u^- R^- \) paths, where two extra CC Circles are taken into consideration. Taking into account the symmetry of \( CC_u|C_u \) and the \( \mu \)-tangency conditions between \( CTs \), we
can derive a total of four feasible geometric scenarios of \(CC_u|C_uC\) paths. Due to space limitation, Fig. 6.12 illustrates one possible geometric scenario to solve the \(CC_u|C_uC\) path planning problem. We can see from Fig. 6.12 that, the computation of such a \(CC_u|C_uC\) is reduced to the determination of the intermediate configurations \(q_1, q_2, q_3\) and the coordinates of the centers of the associated CC Circles.

Let \(\Omega_1\) and \(\Omega_2\) denote the centers of the CC Circles \(C_r^+(q_f)\) and \(C_-^+(q_f)\), respectively; and let \(\theta\) denote the angle between \(\Omega_1, \Omega_2\) and the positive \(x\)-axis. We let the centers of \(C_r^+(q_1)\) and \(C_-^+(q_2)\) be \(\Omega_3\) and \(\Omega_4\), respectively. The \(\mu\)-tangency condition (6.16) with
respect to $C^+_t(q_0)$ and $C^+_r(q_1)$ implies that

$$L(\Omega_1\Omega_3) = L(\Omega_2\Omega_4) = 2R_\Omega; \quad (6.32)$$

whereas the $\mu$-tangency condition (6.17) with respect to $C^+_r(q_1)$ and $C^-_t(q_2)$ asserts that

$$L(\Omega_3\Omega_4) = 2R_\Omega \cos \mu. \quad (6.33)$$

To determine the coordinates of $\Omega_3$ and $\Omega_4$, we exploit the symmetry of $CC_u|C_uC$ to establish that $\Omega_3\Omega_4$ must be parallel to $\Omega_1\Omega_2$. This fact implies that $\Omega_1\Omega_3\Omega_4\Omega_2$ forms an isosceles trapezoid. Let $\alpha$ denote the angle between $\Omega_1\Omega_2$ and $\Omega_1\Omega_3$, a straightforward geometric analysis yields that

$$\cos \alpha = \frac{\frac{1}{2}(L(\Omega_1\Omega_2) - L(\Omega_3\Omega_4))}{L(\Omega_1\Omega_3)} = \frac{L(\Omega_1\Omega_2) - 2R_\Omega \cos \mu}{4R_\Omega}. \quad (6.34)$$

Based on (6.34) and Fig. 6.12, the coordinates of $\Omega_3$ and $\Omega_4$ are respectively given by

\[
\begin{align*}
\Omega_3 &= \begin{pmatrix} x_{\Omega_3} \\ y_{\Omega_3} \end{pmatrix} = \begin{pmatrix} x_{\Omega_1} + 2R_\Omega \cos(\theta - \alpha) \\ y_{\Omega_1} + 2R_\Omega \sin(\theta - \alpha) \end{pmatrix}, \\
\Omega_4 &= \begin{pmatrix} x_{\Omega_4} \\ y_{\Omega_4} \end{pmatrix} = \begin{pmatrix} x_{\Omega_3} + 2R_\Omega \cos \mu \cos \theta \\ y_{\Omega_3} + 2R_\Omega \cos \mu \sin \theta \end{pmatrix}. 
\end{align*}
\]

(6.35)

Let $\theta_1$ represent the angle between $\Omega_2\Omega_4$ and the positive $x$-axis, the $L^+_\delta_1 R^+u L^-u R^-\delta_2$ path can then be formed by sequentially composing

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1$: the $\mu$-tangency condition (6.16) between $C^+_t(q_0)$ and $C^+_r(q_1)$ implies that the orientation of $q_1$ is $\theta - \alpha - \mu + \frac{\pi}{2}$, thus the deflection is $\delta_1 = \theta - \alpha - \mu + \frac{\pi}{2}$;

(ii) a right and forward clothoid turn $CT_2$ with deflection $u$: the $\mu$-tangency condition
between $C_i^+(q_1)$ and $C_i^-(q_2)$ implies that the orientation of $q_2$ is $\theta - \frac{\pi}{2}$, thus the deflection of $CT_2$ is $u = \alpha + \mu - \pi$;

(iii) a left and backward clothoid turn $CT_3$ with the same deflection $u = \alpha + \mu - \pi$;

(iv) a right and backward clothoid turn $CT_4$ with deflection $\delta_2$: the $\mu$-tangency condition (6.16) between $C_i^-(q_2)$ and $C_r^+(q_f)$ implies that the orientation of $q_3$ is $\theta_1 - \frac{\pi}{2} + \mu$, thus the deflection of $CT_4$ is $\delta_2 = \theta_f - \theta_1 - \mu + \frac{\pi}{2}$.

Existence Conditions (i) $\delta_1$, $\delta_2$ and $u$ all satisfy Definition 6.1; (ii) the angle $\alpha$ is well-defined, thus

$$2R_{\Omega} \leq L(\Omega_1\Omega_2) \leq 2R_{\Omega} \cos \mu + 4R_{\Omega}. \quad (6.36)$$

6.4.7 The $C|C_uC_u|C$ Paths

To appropriately generate the canonical $C|C_uC_u|C$ path, namely $L^+ R_u^- L_u^- R^+$, we need to determine three intermediate configurations $q_1$, $q_2$, $q_3$, as shown in Fig. 6.13.

Let $\Omega_1$ and $\Omega_2$ denote the centers of the CC Circles $C_i^+(q_0)$ and $C_r^-(q_f)$ associated with $q_0$ and $q_f$, respectively. Let $\theta$ be the angle between $\Omega_1\Omega_2$ and the positive $x$-axis of the global frame. As shown in Fig. 6.13, $\Omega_3$ and $\Omega_4$ are the centers of the CC Circles $C_r^-(q_1)$ and $C_i^-(q_2)$, respectively. The $\mu$-tangency condition (6.17) between $C_i^+(q_0)$ and $C_r^-(q_1)$, and between $C_i^-(q_2)$ and $C_r^-(q_f)$ implies that

$$L(\Omega_1\Omega_3) = L(\Omega_2\Omega_4) = 2R_{\Omega} \cos \mu. \quad (6.37)$$

In addition, the $\mu$-tangency condition (6.16) between $C_r^-(q_1)$ and $C_i^-(q_2)$ implies that

$$L(\Omega_3\Omega_4) = 2R_{\Omega}. \quad (6.38)$$

The following proposition presents the relative positions of $\Omega_3$ and $\Omega_4$ with respect to $\Omega_1$ and $\Omega_2$ as shown in Fig. 6.13.

**Proposition 6.1.** The straight line $\Omega_1\Omega_3$ is parallel to $\Omega_2\Omega_4$. 

208
Figure 6.13. Geometric planning of $C|C_uC_u|C$ paths

Proof. Let $\theta_1$ denote the angle between $\Omega_1\Omega_3$ and the positive $x$-axis, and let $\theta_2$ denote the angle between $\Omega_4\Omega_2$ and the positive $x$-axis. The $\mu$-tangency between $C_r^+(q_0)$ and $C_r^-(q_1)$ enforces the orientation of $q_1$ to be $\theta_1 + \frac{\pi}{2}$. Thus, after a backward and right clothoid turn with deflection $u$, the orientation of $q_2$ is $\theta_1 + \frac{\pi}{2} + u$ and similarly, the orientation of $q_3$ is $\theta_1 + \frac{\pi}{2} + u - u = \theta_1 + \frac{\pi}{2}$.

On the other hand, serving as the initial configuration of the final backward and right clothoid turn connecting $q_3$ and $q_f$, the $\mu$-tangency between $C_r^-(q_2)$ and $C_r^-(q_f)$ assures the orientation of $q_3$ to be $\theta_2 + \frac{\pi}{2}$. Thus $\theta_1 = \theta_2$, which suffices to prove that the straight line $\Omega_1\Omega_3$ is parallel to $\Omega_2\Omega_4$.

It follows immediately from (6.37) and Proposition 6.1 that $\Omega_1\Omega_4\Omega_2\Omega_3$ forms a parallelogram. Let $\alpha$ denote the angle between $\Omega_1\Omega_2$ and $\Omega_1\Omega_3$. Applying law of cosines within
the triangle $\Omega_1q_2\Omega_3$ yields

$$
\cos \alpha = \frac{(\frac{1}{2}L(\Omega_1\Omega_2))^2 + L^2(\Omega_1\Omega_3)) - (\frac{1}{2}L(\Omega_3\Omega_4))^2}{2L(\Omega_1\Omega_3)L(\Omega_1\Omega_2)}
= \frac{1}{4}L^2(\Omega_1\Omega_2) + (2R_\Omega \cos \mu)^2 - \frac{R_\Omega^2}{2L(\Omega_1\Omega_2)R_\Omega \cos \mu},
\tag{6.39}
$$

The coordinates of $\Omega_3$ and $\Omega_4$ can therefore be obtained in terms of $\theta$ and $\alpha$:

$$
\Omega_3 = \begin{pmatrix} x_{\Omega_3} \\ y_{\Omega_3} \end{pmatrix} = \begin{pmatrix} x_{\Omega_1} + 2R_\Omega \cos \mu \cos(\theta + \alpha) \\ y_{\Omega_1} + 2R_\Omega \cos \mu \sin(\theta + \alpha) \end{pmatrix},
$$

$$
\Omega_4 = \begin{pmatrix} x_{\Omega_4} \\ y_{\Omega_4} \end{pmatrix} = \begin{pmatrix} x_{\Omega_1} + x_{\Omega_2} - x_{\Omega_3} \\ y_{\Omega_1} + y_{\Omega_2} - y_{\Omega_3} \end{pmatrix}.
\tag{6.40}
$$

As illustrated in Fig. 6.13 and Proposition 6.1, the $L^+R_u^-L_u^-R_{\delta_2}^+$ CC path turns to be the sequential concatenation of

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1 = \theta + \alpha + \frac{\pi}{2}$;

(ii) a right and backward clothoid turn $CT_2$ with deflection $u = \theta_1 - \theta - \alpha - \mu$;

(iii) a left and backward clothoid turn $CT_3$ with deflection $-u$.

(iv) a right and forward clothoid turn $CT_4$ with deflection $\delta_2 = \theta_f - \theta - \alpha - \frac{\pi}{2}$.

**Existence Conditions** (i) $\delta_1$, $\delta_2$ and $u$ all satisfy Definition 6.1; (ii) the triangle $\Omega_1q_2\Omega_3$ should be valid, i.e.,

$$
|2R_\Omega - 4R_\Omega \cos \mu| \leq L(\Omega_1\Omega_2) \leq 2R_\Omega + 4R_\Omega \cos \mu.
\tag{6.41}
$$

6.4.8 The $C|C_{\frac{\pi}{2}}^{\frac{\pi}{2}}SC - 1$ Paths

Construction of an $L^+R_{\frac{\pi}{2}}^-S^-R^-$ canonical path that connects the given pair of initial and final configurations $q_0$ and $q_f$ can be viewed as the construction of a $L^+R_{\frac{\pi}{2}}^-$ path pattern followed by a $CSC - 1$ path $R_{\frac{\pi}{2}}^-S^-R^-$. For such a pursuit, we need to determine the
intermediate configurations \( q_1 \), \( q_2 \) and \( q_3 \) which represent the initial configuration of the first right and backward \( CT \), the line segment and the second right and backward \( CT \), respectively. In this subsection, we use the CC Circles \( C^+_r(q_0) \) with center \( \Omega_1 \) for \( q_0 \) and \( C^+_r(q_f) \) with center \( \Omega_2 \) for \( q_f \). Let \( \theta \) denote the angle between \( \Omega_1 \Omega_2 \) and the positive \( x \)-axis of the global frame. The geometric scenario is depicted in Fig. 6.14, where \( \Omega_3 \) is the center of the CC Circle \( C^-_r(q_1) \) associated with the intermediate configuration \( q_1 \). The coordinates of \( \Omega_3 \) can be determined according to the following proposition.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.14.png}
\caption{Geometric planning of \( C|C_2^\pi \) \( SC - 1 \) paths.}
\end{figure}

**Proposition 6.2.** \( \Omega_1 \), \( \Omega_2 \) and \( \Omega_3 \) are collinear.

**Proof.** Let \( \theta_1 \) denote the angle between \( \Omega_1 \Omega_3 \) and the positive \( x \)-axis of the global frame,
and $\theta_2$ denote the angle between $\Omega_3\Omega_2$ and the positive $x$-axis. The $\mu$-tangency condition (6.17) between $C^+_r(q_0)$ and $C^-_r(q_1)$ indicates that the orientation of the intermediate configuration $q_1$ is $\theta_1 + \frac{\pi}{2}$, thus the orientation of $q_2$ is $\theta_1 + \pi$ due to the fact that the deflection of the right and backward $CT$ is set to be $\frac{\pi}{2}$.

On the other hand, based on the $\mu$-tangency condition in the construction of a $CSC - 1$ path between the straight line segment $q_2q_3$ and $C^+_r(q_f)$, $q_2q_3$ is parallel to $\Omega_2\Omega_3$, and thus the orientation of the configuration $q_3$ is $\theta_2 + \pi$. Since $q_2q_3$ is a straight line segment, the orientation of $q_2$ and $q_3$ should be identical; therefore, we have that

$$\theta_1 + \pi = \theta_2 + \pi \iff \theta_1 = \theta_2,$$

indicating the collinearity of $\Omega_1$, $\Omega_2$ and $\Omega_3$. The proof is completed.

Based on the conclusion of Proposition 6.2, the $L^+_{\delta_1}R^\frac{\pi}{2}S^-R^-_{\delta_2}$ path can be computed immediately as follows.

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1 = \theta + \frac{\pi}{2}$;
(ii) a right and backward clothoid turn $CT_2$ with deflection $\frac{\pi}{2}$;
(iii) a straight line segment $q_2q_3$: the length of the straight line segment $q_2q_3$ can be determined in the same way as the $CSC - 1$ path, i.e.,

$$L(q_2q_3) = L(\Omega_2\Omega_3) - 2R_\Omega \sin \mu = L(\Omega_1\Omega_2) - 2R_\Omega \cos \mu - 2R_\Omega \sin \mu,$$

where the second equation is derived by exploiting the $\mu$-tangency condition (6.17);
(iv) a second right and backward clothoid turn $CT_3$ with deflection $\delta_3 = \theta_f - \theta - \pi$.

**Existence Conditions** (i) $\delta_1$, $\delta_2$ and $\delta_3$ satisfy Definition 6.1; (ii) $L(q_2q_3) \geq 0$, that is

$$L(\Omega_1\Omega_2) \geq 2R_\Omega (\cos \mu + \sin \mu).$$
The computation of $C \mid C_{\frac{\pi}{2}} SC - 2$ class of the CC paths requires a slightly different geometric analysis procedure. The construction of the canonical $L^+ R_{\frac{\pi}{2}} S^- L^-$ path involves two CC Circles $C_i^+(q_0)$ and $C_i^+(q_f)$, with centers at $\Omega_1$ and $\Omega_2$, respectively. Let $\theta$ denote the angle between $\Omega_1 \Omega_2$ and the positive $x$-axis. The geometric scenario of this path is illustrated in Fig. 6.15.

Building up an $L^+ R_{\frac{\pi}{2}} S^- L^-$ path is equivalent to computing an $L^+ R_{\frac{\pi}{2}}$ path pattern followed by a $CSC - 2$ path $R_{\frac{\pi}{2}} S^- L^-$. For this purpose, we let $q_1$, $q_2$ and $q_3$ denote the initial configurations of the first right and backward $CT$, the line segment and the second

Figure 6.15. Geometric planning of $C \mid C_{\frac{\pi}{2}} SC - 2$ paths
right and backward CT, respectively. Let $\Omega_3$ be the center of the CC Circle $C_r^-(q_1)$, and let $\alpha$ be the angle between $\Omega_1 \Omega_3$ and $\Omega_1 \Omega_2$. We extend $\Omega_1 \Omega_3$ to $\Omega_0$ in such a way that $\Omega_2 \Omega_0$ is perpendicular to $\Omega_1 \Omega_0$. Let $\Omega_3A$ be perpendicular to $q_2q_3$, and let $B$ denote the intersecting point of $\Omega_2 \Omega_0$ and $q_2q_3$. Based on the $\mu$-tangency condition (6.17) between $C_i^+(q_0)$ and $C_r^-(q_1)$, the orientation of both $q_2$ and $q_3$ is $\theta + \alpha + \pi$, which implies that $q_2q_3$ is parallel to $\Omega_1 \Omega_0$ and is perpendicular to $\Omega_0 \Omega_2$, indicating that $\Omega_3AB\Omega_0$ forms a rectangle. The $\mu$-tangency condition between $q_2q_3$ and $C_r^-(q_1)$, and between $q_2q_3$ and $C_i^+(q_f)$ suggests that

$$L(\Omega_3A) = L(\Omega_2B) = L(\Omega_0B) = R_{\Omega} \cos \mu,$$  \hspace{1cm} (6.45)

which implies that, in the right triangle $\Omega_0\Omega_1\Omega_2$, it holds that

$$\sin \alpha = \frac{L(\Omega_0\Omega_2)}{L(\Omega_1\Omega_2)} = \frac{2R_{\Omega} \cos \mu}{L(\Omega_1\Omega_2)}.$$ \hspace{1cm} (6.46)

It can be inferred immediately from (6.46) and Fig. 6.15 that the $L^+R^-S^-L^-$ path should consist of

(i) a left and forward clothoid turn $CT_1$ with deflection $\delta_1 = \theta + \alpha + \frac{\pi}{2}$;

(ii) a right and backward clothoid turn $CT_2$ with deflection $\delta_2 = \frac{\pi}{2}$;

(iii) a straight line segment $q_2q_3$ of length

$$L(q_2q_3) = L(AB) - 2R_{\Omega} \sin \mu
= L(\Omega_3\Omega_0) - 2R_{\Omega} \sin \mu
= L(\Omega_1\Omega_2) \cos \alpha - 2R_{\Omega} \cos \mu - 2R_{\Omega} \sin \mu;$$  \hspace{1cm} (6.47)

(iv) a second left and backward clothoid turn $CT_3$ with deflection $\delta_3 = \theta_f - \theta - \pi$.

**Existence Conditions** (i) $\delta_1$, $\delta_2$ and $\delta_3$ satisfy Definition 6.1; (ii) $L(q_2q_3) \geq 0$, which is equivalent to

$$L(\Omega_1\Omega_2) \geq 2R_{\Omega} \cos \mu,$$  \hspace{1cm} (6.48)

$$\sqrt{L^2(\Omega_1\Omega_2) - (2R_{\Omega} \cos \mu)^2} \geq 2R_{\Omega}(\cos \mu + \sin \mu).$$
6.4.10 The $C|CSC|C$ Paths

We conclude this section by developing conditions under which a canonical $C|CSC|C$ path, $L^+R_\frac{\pi}{2}S^-L_\frac{\pi}{2}R^+$, exists. Let $C'^+(q_0)$ and $C'^-(q_f)$ be the CC circles associated with $q_0$ and $q_f$, whose centers are $\Omega_1$ and $\Omega_2$, respectively. Let $\theta$ denote the angle between $\Omega_1\Omega_2$ and the positive $x$-axis. The geometric setup of the $L^+R_\frac{\pi}{2}S^-L_\frac{\pi}{2}R^+$ path planning is depicted in Fig. 6.16.

![Figure 6.16. Geometric planning of $C|CSC|C$ paths](image-url)
As shown in Fig. 6.16, to appropriately determine the intermediate configurations \( q_1, q_2, q_3 \) and \( q_4 \), one needs to take advantage of the CC Circles \( C_r^{-}(q_1) \) and \( C_l^{-}(q_3) \), whose centers are denoted as \( \Omega_3 \) and \( \Omega_4 \), respectively. We first introduce the following proposition, which states the positions of \( \Omega_3 \) and \( \Omega_4 \).

**Proposition 6.3.** The straight line \( \Omega_1 \Omega_3 \) is parallel to \( \Omega_4 \Omega_2 \).

**Proof.** Let \( \theta_1 \) and \( \theta_2 \) be the angle between \( \Omega_1 \Omega_3 \), \( \Omega_4 \Omega_2 \) and the positive \( x \)-axis, respectively. The \( \mu \)-tangency condition (6.17) between the CC Circles \( C_l^{+}(q_0) \) and \( C_r^{-}(q_1) \) enforces the orientation of \( q_1 \) to be \( \theta_1 + \frac{\pi}{2} \), and the \( \frac{\pi}{2} \) deflection of the right and backward CT suggests that the orientation of \( q_2 \) and \( q_3 \) be \( \theta_1 + \pi \). Therefore, after the left and backward CT with deflection \( -\frac{\pi}{2} \), the orientation of \( q_4 \) shall be \( \theta_1 + \frac{\pi}{2} \). On the other hand, the \( \mu \)-tangency condition (6.17) between \( C_l^{-}(q_3) \) and \( C_r^{-}(q_f) \) implies that the orientation of \( q_4 \) is \( \theta_2 + \frac{\pi}{2} \). Thus \( \theta_1 = \theta_2 \), i.e., \( \Omega_1 \Omega_3 \) is parallel to \( \Omega_4 \Omega_2 \). The proof is thus completed. \( \square \)

Let \( \Omega_4 \Omega_0 \) be perpendicular to \( \Omega_1 \Omega_3 \). Since \( q_0 q_1 q_2 q_3 q_4 \) forms a \( L^+ R^- S^- L^- \) path from \( q_0 \) to \( q_4 \), it follows from the previous reasoning that \( L(\Omega_4 \Omega_0) = 2R_\Omega \cos \mu \). Moreover, let \( \Omega_5 \) be a point on the straight line \( \Omega_1 \Omega_3 \) such that \( \Omega_2 \Omega_5 \) is perpendicular to \( \Omega_1 \Omega_5 \). From Proposition 6.3, \( \Omega_4 \Omega_2 = 2R_\Omega \cos \mu \) and is parallel to \( \Omega_1 \Omega_5 \), making \( \Omega_0 \Omega_4 \Omega_2 \Omega_5 \) a square. Therefore, the auxiliary angle \( \alpha \) can be determined in the right triangle \( \Omega_1 \Omega_2 \Omega_5 \) as follows:

\[
\sin \alpha = \frac{L(\Omega_2 \Omega_5)}{L(\Omega_1 \Omega_2)} = \frac{2R_\Omega \cos \mu}{L(\Omega_1 \Omega_2)}. \tag{6.49}
\]

It follows immediately from (6.49) that the \( L_{\delta_1}^+ R_{\frac{\pi}{2}}^- S^- L_{\frac{\pi}{2}}^- R_{\delta_2}^+ \) CC path can be formed by composing

(i) a left and forward clothoid turn \( CT_1 \) with deflection \( \delta_1 = \theta + \alpha + \frac{\pi}{2} \);

(ii) a right and backward clothoid turn \( CT_2 \) with deflection \( \frac{\pi}{2} \);

216
(iii) a backward line segment $q_2q_3$ of length

$$L(q_2q_3) = L(\Omega_3\Omega_0) - 2R_\Omega \sin \mu$$

$$= \sqrt{L^2(\Omega_1\Omega_2) - (2R_\Omega \cos \mu)^2} - 4R_\Omega \cos \mu - 2R_\Omega \sin \mu;$$

(iv) a left and backward clothoid turn $CT_3$ with deflection $-\frac{\pi}{2}$;

(v) a right and forward clothoid turn $CT_4$ with deflection $\delta_2 = \theta_f - \theta - \alpha - \frac{\pi}{2}$.

**Existence Conditions** (i) $\delta_1$ and $\delta_4$ satisfy Definition 6.1; (ii) $L(q_2q_3) \geq 0$, which is equivalent to

$$L(\Omega_1\Omega_2) \geq 2R_\Omega \cos \mu,$$

$$\sqrt{L^2(\Omega_1\Omega_2) - (2R_\Omega \cos \mu)^2} \geq 4R_\Omega \cos \mu + 2R_\Omega \sin \mu.$$

6.5 Simulation Results

The effectiveness of existence conditions and the resultant CC path planning algorithm are validated through simulation. We check the feasibility of constructing an RS and a CC path connecting $q_0 = (0, 0, 0, 0)$ and 1000 different $q_f$'s, where each $q_f$ is randomly generated as $q_f \in [-4, 4] \times [-4, 4] \times (-\pi, \pi) \times \{0\}$. Table 6.2 shows that the resultant CC steering algorithm can find feasible CC paths for all cases. In terms of computation efficiency, the construction of the shortest CC path takes about 20 times longer than that of the RS path. For the 1000 path planning problems, an average of 24.3200 classes of the RS paths is feasible, therefore, our three-step planning paradigm admits a 49.33% more efficiency than directly planning 48 types of the CC paths. Moreover, as shown in Table 6.2, generating a feasible CC path requires less computation resource than checking the feasibility of the path; thus, planning RS paths before proceeding to CC reduces extra computational burden.
TABLE 6.2

RS VS. CC FEASIBILITY

<table>
<thead>
<tr>
<th>Paths</th>
<th>RS Paths</th>
<th>CC Paths</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasible Classes (Avg.)</td>
<td>24.3200</td>
<td>8.6280</td>
<td>2.8187</td>
</tr>
<tr>
<td>Planning Time (Avg.)</td>
<td>0.5579 ms</td>
<td>10.4220 ms</td>
<td>25.7414</td>
</tr>
<tr>
<td>Computation Time(Avg.)</td>
<td>0.1260 ms</td>
<td>2.8193 ms</td>
<td>21.9234</td>
</tr>
</tbody>
</table>

Figure 6.17. Reeds-Shepp’s and continuous-curvature paths.

Next, we show that as the $\sigma_{max}$ increases, CC paths converge to corresponding RS paths. This is illustrated by designing an $L^+R^-L^-$ path from $(0, 0, 0, 0)^T$ to $(-2, -2, -0, 0)^T$ with $\sigma_{max}$ varying from 0.5 to 50. As shown in Fig. 6.17 and Fig.-6.18, the length of the CC path converges to the length of the RS path, as $\sigma_{max}$ increases, while the path itself converges to the RS path.
6.6 Conclusion

This chapter explores methods to efficiently construct a class of sub-optimal continuous-curvature paths for mobile robots with car-like dynamics. Geometric analyses are carried out to generate continuous-curvature sub-optimal paths that are composed of straight line segments and clothoid turns for robots which are subject to constraints on velocity, curvature and derivative of the curvature. Based on the established $\mu$–tangency conditions, existence conditions of continuous-curvature paths, having the same patterns as the Reeds-Shepp paths, are derived. The effectiveness of the proposed path planning approach is demonstrated by simulation.
CHAPTER 7
ENFORCEMENT OF DECENTRALIZED AND JOINT OPACITY PROPERTIES OF DISCRETE EVENT SYSTEMS

7.1 Introduction

The information and data among different subsystems of an autonomous CPS are commonly transmitted through open communication channels and therefore are likely to be acquired and/or corrupted by unauthorized adversaries, jeopardizing safe and reliable operation of the system [118, 175]. Such vulnerability highlights the essential need for a large research effort to strengthen the resilience and security of large-scale and highly confidential engineering systems against intentional intrusions.

Motivated by the increasing reliance on cyber-infrastructures in many civilian and industrial applications (ranging from personal banking systems to power grids), various notions of privacy and cybersecurity properties have been extensively proposed in the literature. Using the acronym CIA in [15], cybersecurity properties are generally classified into the three categories.

(i) **Confidentiality**, which guarantees the non-disclosure of information regarding to unauthorized parties

(ii) **Integrity**, which requires that a user never perform illegal actions to change the contents and properties of the information.

(iii) **Availability**, which ensures the timely access to the critical information.

Among the prior studies of confidentiality properties in cyber and cyber-physical systems, there exists a vast literature in the inter-disciplinary study in control theory and computer
science that focuses on the properties that characterize the information flow from the system to an external observer, including anonymity [93, 185], trace-based non-interference [71], secrecy [106, 140] and opacity [20, 75]. In this dissertation, we focus our investigation on the property of opacity, which is a central notion that was first introduced in the computer science community [117] to analyze whether or not the key used in a cryptographic protocol can be inferred by an external observer. Essentially, opacity formalizes the impossibility for an external observer potentially with malicious intentions (termed as an “intruder”) to infer the truth of a predicate representing the confidential information (denoted as a “secret”). In other words, a system is said to be opaque with respect to the given secret if the intruder can never determine unambiguously that the secret has occurred based on its (partial) observation of the system’s behaviors.

Opacity has recently become an active research topic in the realm of DESs, as this type of dynamic systems provides both formal models and analytical methods for investigation. Two fundamental problems, namely the verification and enforcement of opacity properties in DESs, have drawn a considerable amount of research interest in the recent decade. In [20], the opacity notion was formulated for systems that were modeled as Petri nets, where the secret was defined in terms of the predicates over the markings of the underlying Petri net; the results were further extended to labeled transition systems in [21]. A related notion named secrecy was also investigated over labeled transition systems in [1]. Opacity properties of DESs that can be modeled as finite automata were first introduced in [148]. The secret of interest can be defined as either states or languages of the systems. Based on different representations of the secret, a variety of opacity notions have been proposed in the literature, examples include, but not limited to, language-based opacity [6, 27, 55, 109], current-state opacity [151], initial-state opacity [153], $K$-step opacity [150] and infinite-step opacity [152]. A vast majority of research effort has been devoted to the formal verification of the aforementioned opacity notions, see [150, 152, 153] and the references therein. In [192], the authors proved that there existed a polynomial-time trans-
formation between several notions of opacity, including language-based and state-based ones. Furthermore, verification of opacity of Petri nets was also investigated in [177]. Motivated by the absence of likelihood information in the earlier work, opacity notions have also been extended to probabilistic settings in recent years. Current-state opacity was introduced for probabilistic finite automata in [154] by deriving appropriate measures to quantify opacity; whereas corresponding initial-state opacity notion in probabilistic finite automata was developed in [85]. Probabilistic opacity for Markov decision processes was also studied in [12].

In addition to the aforementioned prior work on DES opacity in the presence of a single intruder, recent advances in communication and network technologies have made decentralized monitoring and control of engineering systems feasible, resulting in essential need for the study of opacity problems for DES with decentralized observation structures. For instance, a cryptosystem that can be accessed by users of multiple security levels shall satisfy that: (i) users with lower security level can never infer any information which can only be accessed by users of high security level [71]; (ii) even if a user has a high security level, it is still not able to infer any private information that is possessed by a user with low security level [6]. In this case, both of the requirements can be formulated as opacity enforcement problems. Badouel et al. [6] considered multiple intruders with independent observation mappings and secrets of interest. The system therein was said to be concurrently opaque if all secrets can be kept confidential. The notion of “joint opacity” was proposed in [192], in which a team of intruders collaborated through a coordinator to infer the secret of common interest. Paoli and Lin [129] studied decentralized opacity issues with and without coordination among the intruders. Nevertheless, most of the prior work was derived to the verification of opacity notions in the presence of multiple intruders, while enforcement of these opacity notions were not discussed in detail.

Despite the fruitful accomplishment in the verification of opacity properties, relatively less prior work has been devoted to enforcing the opacity properties via formal model-
based methods in case that the system fails to be opaque. In this chapter, we are hence motivated to investigate how to enforce the secret of a system to be opaque such that an intruder can never infer the occurrence of the secret. In the case that the underlying system fails to ensure opacity of the secret, formal approaches have also been developed to synthesize an opaque DES by means of *enforcement* mechanisms. Among the enforcement methods, supervisory control theory [142, 143] has been utilized for the construction of maximally permissive and non-blocking opacity-enforcing supervisors for DESs. To enforce language-based opacity, the authors of [11] proposed formulas for the computation of the maximal opaque sublanguage of a given specification language, whereas Dubreil et al. [55] developed existence conditions as well as synthesis algorithms of the maximally permissive opacity-enforcing supervisor. In [199], the authors proposed algorithmic procedures for synthesizing maximally permissive supervisors to enforce current-state opacity via “All Enforcement Structures”. Opacity-enforcing supervisors were also designed in [151] to enforce initial-state opacity. Despite their success in ensuring opacity properties, supervisory control approaches cannot be applied for situations where the system must execute its full behavior.

Other than supervisory control approaches, *opacity-enforcement* strategies, which were first introduced in [161], aimed at ensuring the opacity of the system by manipulating its output information, while the full behavior of the system was not altered.

Figure 7.1. Monitoring and opacity-enforcement of discrete event systems.

A typical enforcement architecture is shown in Fig. 7.1, in which the enforcer lies be-
between the system and the observation interface of an observer (possibly with malicious intentions). The enforcer monitors the output of the system and makes modifications whenever necessary. A runtime mechanism was developed in [59] to enforce $K$-step opacity based on delaying the output; however, this method only ensured the opacity of secrets whose time duration was of concern. The authors of [27] enforced the language-based opacity by using dynamic observers, which dynamically modified the observability of every event by activating and/or deactivating the corresponding sensors so that the intruder would never infer the secret; the optimized dynamic observer was also implemented in [202] (termed as maximum information release therein). Different from an opacity-enforcing supervisor, a dynamic observer allows the full system behavior; nonetheless, a dynamic observer also erases some information that ought to be output, and the intermittent loss of observability of certain events may provide clues for the intruder to determine the existence of the opacity-preserving policies. Wu and Lafortune [193] developed an enforcement mechanism based on implementation of insertion functions. An insertion function serves as a monitoring interface at the system’s output, receives actual output behaviors of the system and modifies the output whenever necessary to preserve opacity; the inserted events are observationally equivalent to the system’s genuine observable events from the intruder’s perspective, preventing the intruder from unambiguously inferring the secret. Based on such an opacity-enforcement mechanism, Wu and Lafortune considered the synthesis of an optimal insertion function in [194, 195] for conventional and stochastic DESs, respectively, by assigning a cost value to each inserted event, and the optimal insertion function was synthesized to minimize the total cost and the average cost.

It is worth pointing out that all the aforementioned prior work on opacity-enforcement was derived for the single-intruder case, whereas to the best of our knowledge, very limited work has been devoted to studying opacity-enforcement problems in the presence of multiple intruders. Due to the connection [109] between opacity and other cybersecurity notions such as secrecy and anonymity, we are motivated to study opacity-enforcement
problems of DESs that can be observed by multiple intruders in this chapter. By modeling the system as a finite automaton, we assume that each intruder has full prior knowledge of the system model but can only partially observe the behavior of the system. We investigate opacity problems in two cases, one assuming no coordination among the intruders and the other assuming that the intruders may coordinate with each other. We adopt the enforcement mechanism based on insertion functions to assure decentralized opacity when no coordination exists among the intruders. Furthermore, we study the enforcement of joint opacity when the intruders may coordinate via an intersection-based protocol. Facing the coordinated intruders, we propose a centralized coordination and refinement procedure to construct local insertion functions associated with each intruder’s observation capabilities such that joint opacity can be guaranteed.

The remainder of this chapter is organized as follows. We present the system model and relevant concepts of opacity problems in the context of DESs in Section 7.2. The insertion-based opacity-enforcement mechanism of DESs is introduced in Section 7.3. Appropriate insertion functions are computed in Section 7.4 corresponding to an individual non-coordinating intruder to assure the decentralized opacity property. Under the assumption that the intruders may coordinate via an intersection-based protocol, we introduce the notion of joint opacity in Section 7.5 and develop enforcement schemes for joint opacity by incorporating the synthesis of local insertion functions with centralized coordination. Finally, we end this chapter with concluding remarks in Section 7.6.

7.2 Opacity Notions in Discrete Event Systems

7.2.1 Models of Partially Observed Discrete Event Systems

In this chapter, we consider the DES that is modeled as a NFA $G = (Q, \Sigma, \delta, Q_0)$, where $Q$ is the finite set of states, $\Sigma$ is the finite set of events, $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$ is the (partial) transition function, $Q_0 \subseteq Q$ is the set of initial states. The transition function
δ can be extended to $Q \times \Sigma^*$ in the natural way [26]. In opacity problems, the initial state in $Q_0$ need not be known \textit{a priori} by the intruder and thus it is reasonable to assume a set of possible initial states $Q_0$ in $G$. Given a set $Q' \subseteq Q$ of states, the language generated by $G$ from $Q'$ is defined by $L(G, Q') = \{ s \in \Sigma^* | (\exists q' \in Q')[\delta(q', s)] \}$. The generated language of $G$ is then given by $L(G, Q_0)$. We write $L(G)$ for simplicity if $Q_0$ is clear from the context.

In general, the system $G$ can only be partially observed. Towards this end, $\Sigma$ is partitioned into an observable set $\Sigma_o$ and an unobservable set $\Sigma_{uo}$. The presence of the partial observation is captured by the natural projection $P_o$ from $\Sigma^*$ to $\Sigma_o^*$. To effectively estimate the state of $G$ when $Q_0$ turns to be a singleton $q_0$, we define the following \textit{observer automaton}.

**Definition 7.1 (Observer Automaton).** [26] Consider $G = (Q, \Sigma, \delta, q_0)$ with $\Sigma = \Sigma_o \cup \Sigma_{uo}$.

The observer automaton of $G$ is a finite automaton

$$\text{Obs}(G) = (Q_{obs}, \Sigma_o, \delta_{obs}, q_{0,obs})$$

(7.1)

that is constructed as follows.

(i) Define $q_{0,obs} := UR(q_0)$, and set $Q_{obs} = \{q_{0,obs}\}$.

(ii) For each $B \in Q_{obs}$ and $\sigma \in \Sigma_o$, define

$$\delta_{obs}(B, \sigma) := UR(\{q \in Q | (\exists q_e \in B)[q \in \delta(q_e, \sigma)]\}) .$$

If $\delta(q_e, \sigma)$ is defined for some $q_e \in B$, add the state $\delta_{obs}(B, \sigma)$ to $Q_{obs}$. If $\delta(q_e, \sigma)$ is not defined for any $q_e \in B$, then $\delta_{obs}(B, \sigma)$ is not defined in $\text{Obs}(G)$.

(iii) Repeat Step (ii) until the entire accessible part of $\text{Obs}(G)$ has been constructed.

Note that $UR(\cdot)$ denotes the unobservable reach of a state $q$ (cf. Subsection 3.2.5).
We now proceed to the opacity issues in DESs that can be modeled as finite automata. In the basic opacity problems, it is assumed that the system can be observed by only one single intruder. The ingredients of such problems include: (i) the system $G$ possesses a secret; (ii) the intruder is an observer of $G$ with full prior knowledge of $G$; (iii) the intruder can only partially observe the output behavior of $G$, i.e., the intruder’s observations fall in the set $P_o[L(G)]$. With the prior knowledge of $G$, the intruder is capable of inferring the system’s real behavior by constructing estimates on the basis of observations. Opacity of the secret with respect to both the system and the partial observation holds if for any behavior that may reveal the secret (termed as a secret behavior), there exists at least one behavior that does not reveal the secret (termed as a non-secret behavior) sharing the same observation of the secret behaviors from the intruder’s perspective; thus, the intruder can never be sure whether or not the secret has occurred.

Depending on how the secret is defined, various notions of opacity have been extensively studied in literature, such as language-based opacity (LBO) [109], initial-state opacity (ISO) [153] and current-state opacity (CSO) [148]. For brevity of presentation, we only introduce LBO and CSO in this chapter. Language-based opacity was first defined in [55], in which the secret characterization under consideration was a sublanguage $L_S$ of the language $L(G)$ generated by the system. A system is language-based opaque if an intruder can never infer whether or not the currently observed string generated by the system is contained in the secret language. Formally, the definition of language-based opacity is (termed as “strong opacity” in [109]) stated as follows.

**Definition 7.2 (Language-based Opacity).** [109] Given the system $G = (Q, \Sigma, \delta, Q_0)$, the observation projection $P_o$, the secret language $L_S \subseteq L(G, Q_0)$, and the non-secret language $L_{NS} \subseteq L(G, Q_0)$, $G$ is said to be language-based opaque with respect to $P_o$, $L_S$ and $L_{NS}$ if for every string $t \in L_S$, there exists another string $t' \in L_{NS}$ such that $P_o(t) = P_o(t')$; or equivalently, $L_S \subseteq P_o^{-1}[P_o(L_{NS})]$. 


The system $G$ is LBO if for every string $t \in L_S$, there exists at least one other string $t' \in L_{NS}$ with the same projection. Therefore, given the observation $s = P_o(t)$, the intruder cannot conclude whether the secret string $t$ or the non-secret string $t'$ has occurred.

On the other hand, the secret is defined in terms of a set of states in state-based notions of opacity. In this chapter, we focus our study on the notion of current-state opacity (CSO), which was first introduced as “final opacity” in Petri nets in [20]. The definition was then extended to labeled transition systems [21] and finite automata [148]. By current-state opacity we mean that the intruder can never infer whether or not the current state of the system is a secret state according to its observations.

**Definition 7.3 (Current-state Opacity).** [192] Given the system $G = (Q, \Sigma, \delta, Q_0)$, the observation projection $P_o$, the set of secret states $Q_S \subseteq Q$, and the set of non-secret states $Q_{NS} \subseteq Q$, $G$ is said to be current-state opaque with respect to $P_o$, $Q_S$ and $Q_{NS}$ if

\[
\forall q_i \in Q_0 \text{ and } \forall t \in L(G, q_i) \text{ such that } \delta(q_i, t) \in Q_S, \exists q_j \in Q_0, \exists t' \in L(G, q_j) \text{ such that: (i) } \delta(q_j, t') \in Q_{NS} \text{ and (ii) } P_o(t) = P_o(t').
\]

Intuitively, the notion of CSO states that for every string $t \in L(G)$ that leads to secret states only, there exists another string $t'$ going to a non-secret state that shares the same projection value of $t$. As a result, the intruder can never determine that the system’s current state belongs to the set of $Q_S$.

In fact, it can be shown that a CSO problem can always be transformed into an equivalent LBO problem, and vice versa.

**Theorem 7.1.** Given the system $G = (Q, \Sigma, \delta, Q_0)$, the natural projection $P_o$, the set of secret states $Q_S \subseteq Q$ and the set of non-secret states $Q_{NS} \subseteq Q$, define $L_S(G) = \{s \in L(G, Q_0) | (\exists q_0 \in Q_0)[\delta(q_0, s) \cap Q_S \neq \emptyset]\}$ and $L_{NS}(G) = \{s \in L(G, Q_0) | (\exists q_0 \in Q_0)[\delta(q_0, s) \cap Q_{NS} \neq \emptyset]\}$, respectively. The system $G$ is CSO with respect to $P_o$, $Q_S$ and $Q_{NS}$ if and only if $G$ is LBO with respect to $P_o$, $L_S(G)$ and $L_{NS}(G)$.

**Proof.** $(\Leftarrow)$ According to Definition 7.3, the current-state opacity of the system $G$ implies
that \( \forall q_i \in Q_0 \) and \( \forall t \in L(G, q_i) \) such that \( \delta(q_i, t) \subseteq Q_S \), \( \exists q_j \in Q_0 \), \( \exists t' \in L(G, q_j) \) such that \( \delta(q_j, t') \subseteq Q_{NS} \). Thus from the construction of \( L_S(G) \) and \( L_{NS}(G) \), it can come to a conclusion that in this case \( t \in L_S(G) \) and \( t' \in L_{NS}(G) \). Furthermore, since \( P_o(t) = P_o(t') \) if system \( G \) is CSO with respect to \( P_o, Q_S \) and \( Q_{NS} \), then it follows from Definition 7.2 that system \( G \) is also LBO with respect to \( P_o, L_S(G) \) and \( L_{NS}(G) \).

\((\Rightarrow)\) If \( G \) is LBO with respect to \( P_o, L_S(G) \) and \( L_{NS}(G) \), then Definition 7.2 states that for all \( t \in L_S(G) \), there exists \( t' \in L_{NS} \) such that \( P_o(t) = P_o(t') \). It then follows from the definitions of \( L_S(G) \) and \( L_{NS}(G) \) that there exists \( q_i \in Q_0 \) such that \( \delta(q_i, t) \cap Q_S \neq \emptyset \), while there exists another \( q_j \in Q_0 \) such that \( \delta(q_j, t') \cap Q_{NS} \neq \emptyset \). Since \( P_o(t) = P_o(t') \), it then follows from Definition 7.3 that \( G \) is CSO with respect to \( P_o, Q_S \) and \( Q_{NS} \). \( \square \)

**Remark 7.1.** In the sequel, we will write \( L_S \) for \( L_S(G) \) and \( L_{NS} \) for \( L_{NS}(G) \), respectively, when \( G \) is clearly specified from context.

From Theorem 7.1, enforcing one notion of opacity for the underlying discrete event system is sufficient for the derivation of the results to ensure the other notion of opacity. Therefore, in the rest of this chapter, we use CSO and LBO interchangeably for the demonstration of our results.

The notions of CSO and LBO can be verified by the standard observer automaton in Definition 7.1; whereas ISO can be verified by the trellis-based initial-state estimator introduced in [153]. It turns out in [27] that checking whether or not a system is opaque is PSPACE-complete.

### 7.3 Enforcement of Opacity via Insertion Functions

In this section, we first present the statement of the opacity-enforcement problem for the given DES. Then, we briefly review the opacity enforcement mechanism presented in [193], which is implemented based on the insertion functions.
7.3.1 Opacity-enforcement Problem

We now study the opacity-enforcement problem in case that the system is potentially non-opaque. We require that the opacity of the system $G$ be guaranteed in an online fashion, i.e., once the secret is inferred by the intruder, the opacity of $G$ is violated and cannot be recovered. Recall from Theorem 7.1 that we can verify the given opacity notion by mapping it to LBO and checking if $P_\omega[L(G)] \subseteq P_\omega(L_{NS})$ holds. For such a pursuit, we define the safe language, denoted by $L_{safe}$, as the supremal prefix-closed sublanguage of $P_\omega(L_{NS})$ and can be computed as follows according to [91]

$$L_{safe} := P_\omega[L(G)] - (P_\omega[L(G)] - P_\omega(L_{NS})) \Sigma_\omega^*.$$  (7.2)

Therefore, the opacity of $G$ holds if the intruder’s observation is always contained in $L_{safe}$, i.e., $P_\omega[L(G)] \subseteq L_{safe}$.

Based on the definition of $L_{safe}$, we now formulate the opacity-enforcement problem as follows.

**Problem 7.1** (Opacity-enforcement problem). *Given a non-opaque discrete event system $G = (Q, \Sigma, \delta, Q_0)$, the observation projection $P_\omega$, the set of pre-defined secret states $Q_S \subseteq Q$, the set of pre-defined non-secret states $Q_{NS} \subseteq Q$ with corresponding $L_S$ and $L_{NS}$, modify (if necessary) the output behavior $P_\omega[L(G)]$ such that all observations are contained in $L_{safe}$."

7.3.2 Inserted Events and Relevant Operations

In [193], the authors proposed an opacity-enforcement mechanism based on implementation of insertion functions if the system $G$ fails to be CSO. As shown in Fig. 7.2, an insertion function serves as a special monitoring interface between the system and the intruder. The insertion function receives an output behavior in $s \in P_\omega[L(G)]$ and inserts fictitious observable events before $s$ is observed whenever the intruder may infer the oc-
currence of the secret from $s$. It is worth pointing out that an inserted event looks identical to a genuine observable event generated by the system, therefore the intruder cannot distinguish from one to another. For the purpose of clear presentation, we define an inserted event $\sigma_I$ by associating the observable event $\sigma \in \Sigma_o$ with an “insertion label” $I$, and the set of all inserted events is hence denoted by $\Sigma_I = \{\sigma_I | \sigma \in \Sigma_o\}$.

![Figure 7.2. Opacity-enforcement based on event insertion.](image)

In practice, it is desirable that both the system and the insertion function can clearly distinguish an inserted event from an observable event. Toward this end, we define $P_{und} : (\Sigma_I \cup \Sigma_o) \rightarrow \Sigma_o$ to be a projection that “filters” inserted events from the output of the insertion function: (i) $P_{und}(\sigma_I) = \epsilon, \forall \sigma_I \in \Sigma_I$ and (ii) $P_{und}(\sigma) = \sigma, \forall \sigma \in \Sigma_o$. Clearly, $P_{und}$ can be extended to a projection from $(\Sigma_I \cup \Sigma_o)^*$ to $\Sigma_o^*$ in a recursive manner. On the other hand, from the intruder’s perspective, its observation of the output behaviors of the system is captured by the observation mask $M_I : (\Sigma_o \cup \Sigma_I) \rightarrow \Sigma_o$ instead of the natural projection $P_o$: (i) $M_I(\sigma_I) = \sigma, \forall \sigma_I \in \Sigma_I$ such that $\sigma \in \Sigma_o$ and (ii) $M_I(\sigma) = \sigma, \forall \sigma \in \Sigma_o$. Note that $M_I$ can also be extended to the domain $(\Sigma_I \cup \Sigma_o)^*$ in the usual manner.

In addition, another projection $P_{oil} : (\Sigma \cup \Sigma_I) \rightarrow (\Sigma_I \cup \Sigma_o)$ is used to exclude events
which are neither observable nor inserted, i.e.,

\[ \forall \sigma \in \Sigma \cup \Sigma_I : P_oI(\sigma) = \begin{cases} 
\sigma, & \text{if } \sigma \in \Sigma_o \cup \Sigma_I, \\
\epsilon, & \text{if } \sigma \in \Sigma_{uo} \cup \{\epsilon\}. 
\end{cases} \] (7.3)

Again, \( P_oI \) can be extended to a mapping from \((\Sigma \cup \Sigma_I)^*\) to \((\Sigma_I \cup \Sigma_o)^*\) in the usual manner.

7.3.3 Insertion Functions

An insertion function monitors the output of the system and manipulates the output strings whenever necessary. Specifically, as shown in Fig. 7.2, the insertion function receives an observed event from the system, inserts an extra string of inserted events before the given event, and finally outputs the resulting string over \( \Sigma_I \cup \Sigma_o \). On the one hand, the intruder, observing through the mask \( M_I \), cannot tell whether or not the observed string includes any inserted events. On the other hand, the system can still reconstruct the genuine output string by applying the projection \( P_{und} \) to it. However, it is worth pointing out that although the insertion functions are not available to the intruder, their existence can be detected by the intruder whenever the output string is not consistent with the system knowledge \( L(G) \) [193].

Formally, the basic structure of an insertion function is defined as a (possibly partial) mapping \( f_I : \Sigma_o^* \times \Sigma_o \rightarrow \Sigma_I^* \Sigma_o \), which outputs a string with necessarily inserted events based on the system’s historical and current output behavior. Given a string \( s\sigma_o \in P_o[L(G)] \) where \( s \) stands for the historical observation and \( \sigma_o \in \Sigma_o \) is the current observed event, the output behavior of the insertion function before the occurrence of \( \sigma_o \) is defined as \( f_I(s, \sigma_o) = s_I\sigma_o \), where \( s_I \in \Sigma_I^* \) is the inserted string before outputting \( \sigma_o \). In the sequel, we assume additionally that length of \( s_I \) is bounded from above. The function \( f_I \) defines the instantaneous insertion for every pair \((s, \sigma_o)\). To determine the complete modified string generated by the insertion function, we define an
induced string-based insertion function $f_{I}^{str}$ from $f_{I}$ recursively: (i) $f_{I}^{str}(\epsilon) = \epsilon$ and (ii) $f_{I}^{str}(s_{n}) = f_{I}(\epsilon, \sigma_{1})f_{I}(\sigma_{1}, \sigma_{2}) \cdots f_{I}(\sigma_{1}\sigma_{2} \cdots \sigma_{n-1}, \sigma_{n})$ where $s_{n} = \sigma_{1}\sigma_{2} \cdots \sigma_{n} \in \Sigma_{o}^{*}$ is the observation of a string generated by $G$. Given the system $G$, the modified language output from the insertion function is

$$f_{I}^{str}(P_{o}[L(G)]) = \{ \tilde{s} \in (\Sigma_{I}\Sigma_{o})^{*} | \tilde{s} = f_{I}^{str}(s_{n}) \land s_{n} \in P_{o}[L(G)] \},$$

which is denoted by $L_{out}$ hereafter.

![Figure 7.3. Output of an insertion function.](image)

To pursue succinct notations, we use $f_{I}$ and $f_{I}^{str}$ interchangeably in the sequel. As demonstrated in the Venn’s diagram in Fig. 7.3, a satisfactory insertion function $f_{I}$ for $G$ should be well-defined such that for any observed string $s \in P_{o}[L(G)] - L_{safe}$, $f_{I}^{str}(s) = t$, where $M_{I}(t) \subseteq L_{safe}$ from the intruder’s observation. The system is said to be privately enforceable if such an insertion function exists. Specifically, private enforceability holds if an insertion function is both safe and admissible. Safety of an insertion function requires that every modified behavior from the insertion function be contained in the safe language.
If the system is monitored with a safe insertion function, no output reveals the secret.

**Definition 7.4 (Safe Insertion Functions).** Given the system \( G = (Q, \Sigma, \delta, Q_0) \), the observation projection \( P_o \), the set of pre-defined secret states \( Q_S \subseteq Q \), the set of pre-defined non-secret states \( Q_{NS} \subseteq Q \) with corresponding \( L_S \) and \( L_{NS} \), an insertion function \( f_I \) is safe if the modified output under the observation mask \( M_I \) is within the safe language; that is, \( M_I(L_{out}) \subseteq L_{safe} \).

In addition to safety, admissibility suggests that an insertion function be well-defined to every possible input in \( P_o[L(G)] \).

**Definition 7.5 (Admissible Insertion Functions).** Given a discrete event system \( G = (Q, \Sigma, \delta, Q_0) \), the observation projection \( P_o \), the set of pre-defined secret states \( Q_S \subseteq Q \), the set of pre-defined non-secret states \( Q_{NS} \subseteq Q \), an insertion function \( f_I \) is admissible if it inserts (possibly \( \epsilon \)) on every observable string from \( G \); that is, \( f_I(s, \sigma_o) \) is defined for all \( s\sigma_o \in P_o[L(G)] \).

The private enforceability is then defined as the combination of safety and admissibility

**Definition 7.6 (Private Enforceability).** Given a discrete event system \( G = (Q, \Sigma, \delta, Q_0) \), the observation projection \( P_o \), the set of pre-defined secret states \( Q_S \subseteq Q \), the set of pre-defined non-secret states \( Q_{NS} \subseteq Q \) with corresponding \( L_S \) and \( L_{NS} \), an insertion function \( f_I \) is privately enforcing if it is both safe and admissible. Furthermore, the considered opacity notion is called privately enforceable if there exists a privately enforcing insertion function.

By satisfying both safety and admissibility properties, a privately enforcing insertion function can always modify the system output to a non-secret behavior (as shown in Fig. 7.3) regardless of what the system indeed outputs. Therefore, the opacity can be enforced by the insertion function.
7.4 Synthesis of Insertion Functions that Enforce Decentralized Opacity

7.4.1 Opacity Properties in Decentralized Observation Architecture

In this section, we focus our study on opacity problems of DES with a decentralized observation architecture. Specifically, we extend the investigation of opacity-enforcement strategies to the case in which the system $G$ can be observed by multiple intruders as shown in Fig. 7.4.

![Figure 7.4. A DES $G$ observed by non-coordinating intruders.](image)

The opacity problem for the case in which the intruders do not communicate with each other and there is no coordination among intruders is considered at this point. Let $I_i$, $i \in \mathcal{N} := \{1, 2, \ldots, N\}$ denote a team of $N$ intruders. Similar to the opacity problem in the presence of one single intruder, each intruder here is assumed to have the complete knowledge of the system $G$ a priori. Intruder $I_i$ is associated with the set of local events $\Sigma_i \subseteq \Sigma$ ($i \in \mathcal{N}$). To characterize each intruder’s observation capability, $\Sigma_i$ is further partitioned into the set of locally observable events $\Sigma_{i,o}$ and the set of locally unobservable events $\Sigma_{i,uo}$. The partial observation for the intruder $I_i$ ($i \in \mathcal{N}$) is captured by the projection $P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$ when no (local) insertion function is implemented.
Based on the aforementioned notations, we are now able to extend the notions of LBO and CSO to the decentralized observation architecture shown in Fig. 7.4. Formally, the notion of *decentralized language-based opacity* (D-LBO) is defined as follows.

**Definition 7.7** (Decentralized Language-based Opacity). *Given the system G, the local observation projections* $P_i$ ($i \in \mathcal{N}$), *the secret language* $L_S \subseteq L(G, Q_0)$, *and the non-secret language* $L_{NS} \subseteq L(G, Q_0)$, *G is said to be decentralized language-based opaque with respect to* $P_i$ ($i \in \mathcal{N}$), $L_S$ and $L_{NS}$ if

$$
(\forall i \in \mathcal{N})(\forall t \in L_S)(\exists t_i \in L_{NS})[P_i(t) = P_i(t_i)].
$$

(7.5)

It is worth pointing out that the non-secret string $t_i$ in (7.5) can be different for each intruder $I_i$ ($i \in \mathcal{N}$). Intuitively, by D-LBO we mean that for each intruder and for any string generated by the system that may reveal the secret, there always exists another non-secret string that is observationally equivalent to this intruder. On the other hand, by defining the secret as a set of states of $G$, the property of *decentralized current-state opacity* (D-CSO) is defined as follows.

**Definition 7.8** (Decentralized Current-state Opacity). *Given the system G, the local observation projections* $P_i$ ($i \in \mathcal{N}$), *the set of secret states* $Q_S$, *and the set of non-secret states* $Q_{NS}$, *G is said to be decentralized current-state opaque with respect to* $P_i$ ($i \in \mathcal{N}$), $Q_S$ and $Q_{NS}$ if

$$
(\forall i \in \mathcal{N})(\forall q_{i,0} \in Q_0)(\forall t_i \in L(G, q_{i,0}) : \delta(q_{i,0}, t_i) \in Q_S) \Rightarrow

(\exists q'_{i,0} \in Q_0)(\exists t'_i \in L(G, q'_{i,0}) \mid [\delta(q'_{i,0}, t'_i) \in Q_{NS}) \land (P_i(t'_i) = P_i(t_i))].
$$

(7.6)

The D-CSO of the system $G$ suggests that any secret state in $Q_S$ be not inferred by any one of the intruders based on local observations. The following theorem suggests that a D-CSO problem can always be transformed into an equivalent D-LBO problem, and vice versa.
Theorem 7.2. Given the system $G = (Q, \Sigma, \delta, Q_0)$, the local projection $P_i \ (i \in N)$, the set of secret states $Q_S \subseteq Q$ and the set of non-secret states $Q_{NS} \subseteq Q$, define $L_S(G) = \{s \in L(G, Q_0) | (\exists q_0 \in Q_0)[\delta(q_0, s) \cap Q_S \neq \emptyset] \}$ and $L_{NS}(G) = \{s \in L(G, Q_0) | (\exists q_0 \in Q_0)[\delta(q_0, s) \cap Q_{NS} \neq \emptyset] \}$, respectively. The system $G$ is D-CSO with respect to $P_i \ (i \in N)$, $Q_S$ and $Q_{NS}$ if and only if $G$ is D-LBO with respect to $P_i \ (i \in N)$, $L_S(G)$ and $L_{NS}(G)$.

Proof. The theorem can be proved as the same way of proving Theorem 7.1, by checking the CSO-LBO equivalence with respect to each observation projection $P_i \ (i \in N)$. \qed

The conclusion of Theorem 7.2 states that enforcing one notion of decentralized opacity for the underlying discrete event system is sufficient for the derivation of the results to ensure the other notion of opacity. Therefore, in the sequel, we focus the investigation on the D-CSO property.

7.4.2 Enforcing Decentralized Opacity via Insertion Functions

It follows from Definitions 7.7 and 7.8 that D-LBO and D-CSO can be viewed as a decentralized counterpart of LBO and CSO with respect to each individual intruder $I_i \ (i \in N)$, respectively. This fact implies that the enforcement of D-CSO for $G$ in the presence of multiple non-coordinating intruders is equivalent to the enforcement of local CSO with respect to each individual intruder $I_i \ (i \in N)$. As a result, we can synthesize local opacity-enforcing insertion function $f_{I,i}$ for the intruder $I_i \ (i \in N)$ independently so that the D-CSO of the system can be ensured.

The synthesis procedure via “All Insertion Structure” (AIS) that was proposed in [193] and [194] is inherited at this point to synthesize appropriate local insertion functions for the intruder $I_i \ (i \in N)$. For clarity of presentation, we illustrate the synthesis procedure by the following example.

Example 7.1. Consider the system $G = (Q, \Sigma, \delta, Q_0)$ shown in Fig. 7.5, where the event set $\Sigma = \{a, b, c, d\}$. The set of secret states is given by $Q_S = \{1, 2, 6\}$, which are the
shaded states in Fig. 7.5. We assume that $G$ is observed by two intruders with different sets of observable events, namely, $\Sigma_{1,o} = \{b, c, d\}$ and $\Sigma_{2,o} = \{a, c, d\}$, respectively.

The observer automata $\text{Obs}_1(G)$ and $\text{Obs}_2(G)$ can be constructed according to Definition 7.1, and are shown in Fig. 7.6 and Fig. 7.7, respectively.
For \( i \in \{1, 2\} \), each state in \( \text{Obs}_i(G) \) contains the current state estimation from the perspective of the intruder \( I_i \). From Fig. 7.6 and Fig. 7.7, it can be verified that both of the observer automata reveal some secrets (the shaded states therein) without the opacity-enforcement mechanism.

Since we assume that the intruders do not communicate with each other and there is no coordination between the intruders, we can follow the procedures presented in [194] to construct the AIS that encodes all the valid system and insertion function moves for each intruder individually. An appropriate insertion function for each intruder can then be extracted from the corresponding AIS.

![Figure 7.8. The AIS \( AIS_1 \) for intruder \( I_1 \).]
Given a system that is not opaque, the AIS is a bipartite graph that enumerates all valid insertion functions. The construction procedure of an AIS is presented in [194] and is composed of the following three stages:

(i) Constructing the \( i \)-Verifier;

(ii) Constructing the Unfolded \( i \)-Verifier;

(iii) Pruning and obtaining the All Insertion Structure (AIS).

Formally, an AIS is a tuple

\[
AIS = (X, \Sigma_o \cup \{\epsilon\}, f, x_0),
\]

which can be viewed as a 2-player game structure between the “system player” and the “insertion function player”. Towards this end, we assume that \( X = X_S \cup X_I \), where \( X_S \) denotes the set of the states of the system player and \( X_I \) denotes the set of the states of the insertion function player. In the game structure (7.7), the system player moves at \( X_S \) states; the insertion function player moves at \( X_I \) states. A given system state \( x \in X_S \) is a state in the \( i \)-verifier and can be represented as a pair of state estimates \( x = (m_d, m_f) \), where \( m_d \) is the intruder’s estimate (which could be wrong due to the inserted events), and \( m_f \) is the real state estimate of the system; whereas each action at \( x \) is an output \( \sigma \in \Sigma_o \). On the other hand, a given insertion function state, say \( z = (x, \sigma) \in X_I \), consists of not only its predecessor state \( x \), but the action of observable event \( \sigma \in \Sigma_o \) that the system player has just made. The transition function \( f \) is written as \( f = f_{SI} \cup f_{IS} \), where \( f_{SI} : X_S \times \Sigma_o \to X_I \) denotes the transition function from \( X_S \) to \( X_I \), and \( f_{IS} : X_I \times X_S \to X_S \) denotes the transition function from \( X_I \) to \( X_S \). As in the game structure of the AIS, the system is the first player, the initial state is defined to be \( x_0 \in X_S \). The following theorem establishes the relationship between the private enforceability and the non-emptiness of the obtained AIS.
**Theorem 7.3.** [193] *The current-state opacity is privately enforceable if and only if the obtained AIS is nonempty.*

In other words, if an insertion function can be synthesized from the AIS, it must be privately enforcing.

Figure 7.9. The AIS $AIS_2$ for intruder $I_2$.

The main differences between the AIS proposed in this chapter and the AIS defined in [194] are two folds. The first is that we unfold the moves of the insertion function as well as the intruder’s state estimate *event by event*, while in [194], the insertion function’s move is from $\Sigma^*_o \cup \{\epsilon\}$, which could denote the whole string that has been inserted. For example, in our AIS definition, if we have a transition $x_0 \xrightarrow{a} x_1 \xrightarrow{b} x_2 \xrightarrow{c} x_3$, where $x_0, x_1, x_2 \in X_I$...
and $x_3 \in X_S$, in [194]'s definition, the same transition would be simplified to $x_0 \xrightarrow{abc} x_3$.

The second is that, if the system $G$ contains loops (for the simplest case, imagine there is a self-loop in some state), it could be the case that the inserted string contains $s^*$ for some $s \in \Sigma_o^*$ and becomes arbitrarily long (for example, Fig. 7 in [194]). In our case, we restrict the inserted strings to be $\ast$-free, that is, the insertions cannot be arbitrarily long and we replace $s^*$ with $\epsilon$. Our definition with unfolding and $\ast$-free insertions is to facilitate the analysis of joint opacity enforcement in Section 7.5.

The AISs for the intruders $I_1$ and $I_2$ in Example 7.1 are depicted in Fig. 7.8 and Fig. 7.9, respectively, where the system states are represented by rectangles and the insertion function states in $X_I$ are represented by ellipses. All the transitions with events originated from the system states are deemed as “uncontrollable” transitions, while all the transitions from insertion function states are insertion function moves that the intruder actually observes. In the two AISs, we have $m_0 = \{0, 1\}, m_1 = \{2\}, m_2 = \{3, 4\}, m_3 = \{5, 6\}, m_4 = \{7\}, m_5 = \{8, 9\}, m_6 = \{10\}, c_0 = \{0, 2\}, c_1 = \{1\}, c_2 = \{3, 5\}, c_3 = \{4, 6\}, c_4 = \{7\}, c_5 = \{8, 10\}, c_6 = \{9\}$. For instance, in $AIS_1$ shown in Fig. 7.8, starting from the initial state where the intruder and the system’s estimates are $(m_0, m_0)$, if the event $b$ occurs in the system, $AIS_1$ transits to the insertion function state $((m_0, m_1), b)$ since the system observer sees the event $b$ and the intruder observer observes nothing as the insertion has not been decided yet. Then if the insertion function decides to insert $c$, the system transits to the insertion function state $((m_2, m_1), b)$ as the intruder observer observes $c$ and the system observer will ignore the insertion function outputs. Then the real system output $b$ is appended and consequently $AIS_1$ transits to the system state $(m_3, m_1)$.

For a more concise representation, we encode the AIS into a corresponding Nondeterministic Finite-state Mealy Automaton (NFM), which is defined as follows.

**Definition 7.9** (Non-deterministic Mealy Automaton). A non-deterministic finite-state Mealy automaton is a 5-tuple

$$\mathcal{M} = (Q, \Sigma_I, \Sigma_{Out}, q_0, f),$$

(7.8)

242
where $Q$ is the set of states, $\Sigma_{In}$ and $\Sigma_{Out}$ are the sets of input and output symbols, respectively, $q_0 \in Q$ is the initial state, $f(q, \sigma) = (q', o)$ defines the transition and input output relation for $q, q' \in Q, \sigma \in \Sigma_{In}, o \in \Sigma_{Out}$.

The non-determinism of an NFM comes from the fact that in general $|f(q, \sigma)| \geq 1$, which implies that the same input on the same state may result in non-unique insertions and transit to different states. Our NFM formulation is similar to the insertion automaton in [194] but we allow non-deterministic choices of insertions upon observing a system output $\sigma \in \Sigma_o$. The procedure to convert an AIS into an NFM $M = (Q, \Sigma_{In}, \Sigma_{Out}, q_0, f)$ is stated as follows. $Q$ is the set of all the system states of the given AIS, $\Sigma_{In}$ is the set of all the events originated from the system states and $\Sigma_{Out}$ is the set of all the possible insertion strings. The transition function of the NFM is defined as $f(q, \sigma) = (q', o)$, where $o = o'\sigma, o' \in \Sigma_i^*, \sigma \in \Sigma_o, o'$ is the inserted string, $\sigma$ is the system input and $o$ denotes the output from the state $q$ when the system input is $\sigma$. We revisit Example 7.1 to study the NFM representation of an insertion function.

**Example 7.2.** Fig. 7.10 and Fig. 7.11 illustrate the NFMs $M_1$ and $M_2$ for the intruders $I_1$ and $I_2$ in Example 7.1, respectively, where $m_0 = \{0, 1\}, m_1 = \{2\}, m_2 = \{3, 4\}, m_3 = \{5, 6\}, m_4 = \{7\}, m_5 = \{8, 9\}, m_6 = \{10\}, c_0 = \{0, 2\}, c_1 = \{1\}, c_2 = \{3, 5\}, c_3 = \{4, 6\}, c_4 = \{7\}, c_5 = \{8, 10\}, c_6 = \{9\}$. Note that, different from the corresponding AIS representations, in the NFM formulation, upon observing an event $\sigma$, the state directly jumps from $q$ to $q'$ while outputting the string $o$. However, what really happens is that the intruder’s estimation is updated event by event for each output of the insertion function. Such estimation evolution is omitted in the NFM formulation for conciseness.
Figure 7.10. The NFM $M_1$ for AIS$_1$.

Figure 7.11. The NFM $M_2$ for AIS$_2$.

We conclude this section with the following theorem, which presents the necessary and sufficient conditions for the private enforceability of the D-CSO property.
Theorem 7.4. Given the system $G$, the local observation projections $P_i (i \in \mathcal{N})$, the set of secret states $Q_S$ and the set of non-secret states $Q_{NS}$, the D-CSO property of $G$ is privately enforceable if and only if $AIS_i$ is nonempty for all $i \in \mathcal{N}$.

Proof. According to Definition 7.8, D-CSO holds if and only if local CSO holds for all individual intruder $I_i (i \in \mathcal{N})$. Thus, the D-CSO property of $G$ is privately enforceable if and only if for each intruder $I_i (i \in \mathcal{N})$, the local CSO is privately enforceable (with respect to $P_i$) is privately enforceable. Furthermore, from Theorem 7.3, this condition holds if and only if the AIS corresponding to $I_i (i \in \mathcal{N})$, namely $AIS_i$, is not empty. Therefore we complete the proof of the theorem. 

7.5 Enforcement of Joint Opacity in the Presence of Coordinated Intruders

In this section, we aim to extend the study of state-based opacity notions (e.g., CSO and ISO) to the case in which multiple intruders may work as a team to jointly infer the secret. Rather than observing the same system without coordination, in many applications, intruders do coordinate among themselves by exchanging their estimates of the system’s states. For these applications, decentralized opacity notions of coordinating intruders need further investigation. Specifically, we study a simplified coordinated observation architecture [51] where intruders may coordinate with one coordinator.

7.5.1 Intersection-based Coordination Protocol

As shown in Fig. 7.12, we assume that each intruder $I_i (i \in \mathcal{N})$ not only observes the system through the individual observation projection and generates local state estimates, but reports the estimates to a coordinator as well. The coordinator has no prior knowledge about the system. It forms the so-called coordinated estimate by following an intersection-based coordination protocol [198], i.e., by taking the intersection of the local estimates that it receives. The communication from the local intruders to the coordinator is assumed to
ideal and no delay takes place. The collaboration is restricted by the following rules: (i) intruders have no knowledge of the projections of one another; (ii) intruders do not directly communicate with each other about their individual state estimates; (iii) the only collaboration between the intruders is through the coordinator, whose only available memory is to store the most recent coordinated estimate. Because of the restricted collaboration among the intruders, the coordinated state estimate is no finer than the estimate of a single “system intruder” that would observe the union of the events that are observable to each individual intruder. Such coordinated structures capture the situations where a system intruder does not exist and where the coordination among local intruders is restricted. It should be noted that other collaboration rules also exist from the fault diagnosis literature [86], a dual problem of opacity.

```
G = (Q, Σ, δ, Q₀)
```

Figure 7.12. The DES $G$ observed by intruders with intersection-based coordination protocols.
We first formally introduce the intersection-based coordination protocol that is embedded in the coordinated observation architecture shown in Fig. 7.12. For the intruder \( I_i (i \in \mathcal{N}) \), a string-based local estimation map \( \psi^i : P_i[L(G)] \rightarrow 2^Q \) is defined as follows: for \( s \in L(G) \) and \( s_i := P_i(s) \),

\[
\psi^i(s_i) = \delta \left( Q_0, P_i^{-1}(s_i) \cap L(G) \right) .
\]  

(7.9)

Then, we define an intersection-based coordination protocol \( \Psi : \Pi_{i \in \mathcal{N}} P_i[L(G)] \rightarrow 2^Q \) as

\[
\Psi(s_1, s_2, \cdots, s_N) = \bigcap_{i \in \mathcal{N}} \psi^i(s_i) .
\]  

(7.10)

In other words, the intersection-based coordination protocol \( \Psi \) takes the intersection of the local estimates reported by the intruders and forms a coordinated estimate accordingly.

7.5.2 Enforcement Scheme of Joint Opacity in Discrete-event Systems

We now consider opacity issues of DES that can be observed by intruders following the intersection-based coordination protocol in (7.10). Roughly speaking, the system is said to be jointly current-state opaque if no coordinated estimate ever reveals the secret information.

**Definition 7.10 (Joint Current-state Opacity (J-CSO)).** Given the system \( G \), the local observation projections \( P_i \ (i \in \mathcal{N}) \), the set of secret states \( Q_S \) and the set of non-secret states \( Q_{NS} \), \( G \) is said to be jointly current-state opaque under the coordinated architecture shown in Fig. 7.12 if

\[
(\forall q_0 \in Q_0)(\forall t \in L(G, q_0) : \exists q \in Q_S \land q \in \delta(q_0, t)) \Rightarrow \]

\[
(\forall i \in \mathcal{N})(\exists q^i_0 \in Q_0 \land t_i \in L(G, q^i_0)) [\exists q' \in Q_S : (q' \in \delta(q^i_0, t_i)) \land (P_i(t) = P_i(t_i) = s_i)] .
\]  

(7.11)
The intuition behind Definition 7.10 states that the system $G$ is jointly-current-state opaque if for every string $t$ that reaches to a secret state $q$, there are $N$ other strings $t_i (i \in \mathcal{N})$ reaching to a common non-secret state $q'$ such that the intruder $I_i$ cannot distinguish $t$ from the string $t_i$. In Definition 7.10, the strings $t_i (i \in \mathcal{N})$ are not necessarily required to start from the same initial state, but they need to reach to a common non-secret state so that the secret state will not be revealed in the coordinated estimate. Alternatively, the property of J-CSO can also be defined in terms of the intersection-based coordination protocol in (7.10).

**Definition 7.11 (J-CSO, alternative).** Given the system $G$, the local observation projections $P_i (i \in \mathcal{N})$, the set of secret states $Q_S$, the set of non-secret states $Q_{NS}$, and the intersection-based coordination protocol $\Psi$, the system $G$ is said to be jointly current-state opaque in the presence of $\Psi$ if

(i) $G$ is current-state opaque with respect to each individual intruder $I_i (i \in \mathcal{N})$;

(ii) for all the strings $s_i (i \in \mathcal{N})$ that are defined in Definition 7.10, it holds that

$$\Psi(s_1, s_2, \ldots, s_N) \cap Q_S \neq \emptyset \land \Psi(s_1, s_2, \ldots, s_N) \cap Q_{NS} \neq \emptyset. \quad (7.12)$$

It can be inferred from Definition 7.11 that D-CSO is a necessary condition for the satisfaction of J-CSO, i.e., a system $G$ that is J-CSO implies that $G$ is also D-CSO. Nevertheless, the converse of this assertion does not hold in general, as explained in the following example.

**Example 7.3.** With the AIS$_1$ and AIS$_2$ obtained in Fig. 7.8 and Fig. 7.9, respectively, D-CSO is guaranteed in Example 7.1. However, if the two intruders can send their estimates to the intersection-based coordinator, joint opacity may be violated. For instance, if the system outputs a string cab, the string will result in the projected string cb for the intruder $I_1$ and ca for $I_2$, respectively. If both insertion functions choose to insert the empty string $\epsilon$, which is a valid move according to the admissibility, the resulting estimates reported by intruders $I_1$ and $I_2$, after observing cb and ca, are $\{5, 6\}$ and $\{4, 6\}$, respectively, which
yields a coordinated state estimate \( \{6\} \in Q \) according to (7.10). In this case, one secret state 6 is revealed and the J-CSO property is violated.

Example 7.3 implies that insertion functions which can enforce the D-CSO property may not necessarily guarantee the J-CSO property as well. Therefore, after synthesizing appropriate insertion functions for each intruder that ensure the accomplishment of D-CSO, the obtained local insertion functions need to be specifically coordinated and refined to enforce the J-CSO. In the sequel, we present a centralized approach to synthesize proper insertion functions for each intruder so that the J-CSO property of the system can be assured.

Since D-CSO is a necessary condition for J-CSO, thus in our approach, we first enumerate all the appropriate local insertion functions for each intruder to enforce the D-CSO property, which can be computed by following the steps in Section 7.4 and the obtained insertion functions can be represented by either the AISs \( AIS_i \) or equivalently the individual insertion NFMs \( M_i \ (i \in \mathcal{N}) \).

Next, the local insertion NFMs should be further coordinated and the set of appropriate local insertion functions should be pruned in order to satisfy J-CSO in the presence of the intersection-based coordination protocol. It should be noticed that the insertion NFMs associated with each intruder should be synchronized with the original system \( G \) so that no extra secret information can be revealed due to coordination of asynchronous state estimates from each intruder. For such a pursuit, we construct another system observer \( \text{Obs}(G) = (Q_{\text{Obs}}, \Sigma_{\text{Obs},o}, \delta_{\text{Obs}}, q_{0,\text{Obs}}) \), where

\[
\Sigma_{\text{Obs},o} = \bigcup_{i \in \mathcal{N}} \Sigma_{i,o}. \tag{7.13}
\]

In other words, \( \text{Obs}(G) \) is constructed as if the system \( G \) can be observed by a system intruder, which can observe the occurrences of any events as long as it can be observed by at least one individual intruder. For instance, for the system shown in Fig. 7.5, \( \Sigma_{\text{Obs},o} = \Sigma \)
and the observer thus has the identical structure as the system $G$.

Note that the system observer automaton $\text{Obs}(G)$ can also be viewed as an NFM that outputs the empty string $\epsilon$ regardless of the inputs, i.e., $\Sigma_{Out} = \{\epsilon\}$. Therefore, given $N$ NFMs $M_i = (Q_i, \Sigma_{In}^i, \Sigma_{Out}^i, q_0^i, f_i)$ ($i \in \mathcal{N}$) and the system observer automaton $\text{Obs}(G)$, we can build up a composed NFM

$$G = (Q_G, \Sigma_{In}, \Sigma_{Out}, q_0, f)$$

(7.14)

that describes all the possible combined insertion behaviors, where

- $Q_G = Q_1 \times Q_2 \times \ldots \times Q_N \times Q_{\text{Obs}}$;
- $\Sigma_{In} = \Sigma_{In}^1 \times \Sigma_{In}^2 \times \ldots \times \Sigma_{In}^N \times \Sigma_{\text{Obs},o}$;
- $\Sigma_{Out} = \Sigma_{Out}^1 \times \Sigma_{Out}^2 \times \ldots \times \Sigma_{Out}^N \times \{\epsilon\}$;
- and the transition relation $f((q_1, q_2, \ldots, q_N, q_{\text{obs}}), \sigma) = ((q'_1, q'_2, \ldots, q'_N, q'_{\text{obs}}), o)$ is given by
  
  (i) $o = (o_1, \ldots, o_N, \epsilon)$, where $f_i(q_i, \sigma) = (q'_i, o_i)$ ($i \in \mathcal{N}$) if $\sigma \in \Sigma_{i,o}$; otherwise $(q_i, \epsilon) := (q'_i, o_i)$;
  
  (ii) $\delta_{\text{Obs}}(q_{\text{obs}}, \sigma) = (q'_{\text{obs}}, \epsilon)$.

By the aforementioned construction procedure of the composed NFM $G$, we assume that when an event $\sigma \in \Sigma_o$ is output by the system, it is guaranteed that for every intruder $I_i$ ($i \in \mathcal{N}$) such that $\sigma \in \Sigma_{i,o}$ holds, its corresponding local insertion function will finish outputting the modified string $o_i$ before the next system event $\sigma' \in \Sigma_u$ is generated. It is always possible since we restrict the output of the insertion functions to be $*$-free. In this regard, every insertion function is synchronized with the system inputs.

We illustrate the construction procedure of $G$ by using the following example.

**Example 7.4.** Fig. 7.13 illustrates the $G$ that can be obtained from the system $G$ in Example 7.3, along with the corresponding NFMs $M_1$ and $M_2$ depicted in Fig. 7.10 and Fig. 7.11, respectively. For simplicity, in this figure we omit the constant output $\epsilon$ from the system observer as well as each individual observer’s state estimation.
Thanks to the obtained local AISs and NFMs for each intruder, for a given transition $f((q_1, ..., q_N, q_{obs}), \sigma) = ((q'_1, ..., q'_N, q'_{obs}), o)$, it is then possible to check whether the secrets will be revealed and joint opacity could be violated during this transition. Note that the event by event evolution of the state estimation for each intruder upon observing a modified string is omitted in the NFM but not AIS.

For instance, in the composed NFM $G$ shown in Fig. 7.13, starting from the initial state, when the event $c$ is output and the two local insertion functions decide to insert $d$ and $\epsilon$, respectively, the transition is $(m_0, c_0, 0) \xrightarrow{c/(dc,c)} (m_5, c_2, 3)$. From the AISs depicted in Fig. 7.8 and Fig. 7.9, the evolution of each intruder’s estimation can be viewed as a two step transition $(m_0, c_0, 0) \xrightarrow{d,c} (m_4, c_2, 3) \xrightarrow{c,\epsilon} (m_5, c_2, 3)$. Note that there is an intermediate state $(m_4, c_2, 3)$ that is not shown in $G$. In the first step, upon observing the system event $c$, the first insertion function outputs $d$ and the second insertion function decides to insert nothing, the system event $c$ is directly outputted. Therefore, the estimations evolve from $m_0$ to $m_4$, $c_0$ to $c_2$, and the system observer’s estimation changes from 0 to 3. In the
second step, the first insertion function outputs the system event $c$ and the second insertion function outputs $\epsilon$. It can be seen that our assumption is that the event output (including $\epsilon$) for each intruder is synchronized.

To determine if a transition $(q, \sigma) \rightarrow (q', o)$ in $G$ is safe, we examine every intermediate state from the AISs that evolves with each output event to see if the joint estimation reveals a secret. By definition, $q_{obs}$ encodes the “true” set of states that the system is currently in. The first element of $AIS_i$ state, denote as $est_i$, represents each intruder’s estimation of the current state of the system. According to the intersection-based coordination protocol (7.10), the coordinated estimation, written as $J_{est}$, is obtained by taking the intersection among $q_{obs}$ and $est_i (i \in \mathcal{N})$.

$$J_{est} = q_{obs} \cap est_1 \cap \ldots \cap est_N.$$  \hspace{1cm} (7.15)

In this case, the following proposition can be drawn as an immediate corollary of Definition 7.11.

**Proposition 7.1.** $J_{est}$ does not reveal a secret if $J_{est} \cap Q_S \neq \emptyset \implies J_{est} \cap Q_{NS} \neq \emptyset$

Based on Proposition 7.1, we now re-define the J-CSO property in the presence of synthesized insertion functions as follows.

**Definition 7.12.** Given the locally unobservable event sets $\Sigma_{i,u_o}$ and the synthesized insertion functions $f_{I,i}$ corresponding to the intruder $I_i (i \in \mathcal{N})$, the system $G$ is J-CSO if

- For each individual intruder $I_i (i \in \mathcal{N})$, the insertion function $f_{I,i}$ enforces local CSO.
- The $J_{est}$ never reveals the secret.

Furthermore, we define J-CSO to be jointly privately enforceable if all individual insertion functions are locally privately enforcing and $J_{est}$ never reveals the secret.
An intermediate state is unsafe if its $J_{est}$ reveals a secret. A transition $(q, \sigma) \rightarrow (q', o)$ in $G$ is unsafe if any of the intermediate states between $q$ and $q'$ is unsafe. Similarly, any state $q$ of $G$ is unsafe if its $J_{est}$ reveals a secret. If a transition is found to be unsafe, it will be pruned. If a state $q$ is found to be unsafe, this state, together with all its incoming and outgoing transitions, will be pruned. If after the pruning, at some state $q' \in Q$, there is no incoming transition (except the initial state) or there is no outgoing transition defined on an event $e$ that could happen in this state, which implies that the system blocks when $\sigma$ is output at $q'$ since there is no insertion function available, then such state is also unsafe and all its incoming and outgoing transitions will be pruned. Again, such pruning may trigger new deadlocks and create unsafe states. Therefore, this is an iterative process until no unsafe state is found or the initial state is pruned.

For example, as shown in Fig. 7.13, the state $(m_3, c_3, 6)$ is an unsafe state that reveals the secret, since $m_3 \cap c_3 \cap \{6\} = \{5, 6\} \cap \{4, 6\} \cap \{6\} = \{6\} \in Q_S$. Therefore it has to be pruned, which results in the states $(m_2, c_3, 4)$ and $(m_3, c_2, 5)$ being unsafe since they would become blocking states after the pruning. Consequently, pruning $(m_2, c_3, 4)$ and $(m_3, c_2, 5)$ leads $(m_2, c_2, 3)$ to be unsafe. In this case, by removing $(m_2, c_2, 3)$ from $G$, the pruning process terminates. It can be verified that no more state or transitions is found to be unsafe, thus the resulting $G$ can be found in Fig. 7.13, excluding the states in the dashed box.

We conclude this subsection with the following theorem that states the necessary and sufficient conditions for the jointly private enforceability of the J-CSO property in the presence of coordinating intruders.

**Theorem 7.5.** Given the system $G$ and the observation projections $P_i$ for the intruder $I_i$ ($i \in \mathcal{N}$), J-CSO is jointly privately enforceable if and only if $G$ is nonempty after pruning.

**Proof.** ($\Rightarrow$) If J-CSO is jointly privately enforceable, in our definition, it implies that for each individual intruder $I_i$ ($i \in \mathcal{N}$), local CSO is privately enforceable and thus $AIS_i$ is nonempty by Theorem 7.3. Since $AIS_i$ enumerates all the possible local insertion func-
tions that are privately enforcing for the intruder $I_i \ (\in \mathcal{N})$, the composed NFM $\mathcal{G}$, which serves as the product of all AISs, should enumerate all the possible joint insertion strategies that are privately enforcing. Since J-CSO is jointly privately enforceable, there exists at least one local insertion function for each intruder $I_i$ that is privately enforcing and the joint estimate never reveals the secret. Thus the joint insertion strategy is nonempty, which implies that $\mathcal{G}$ is nonempty.

$(\Leftarrow)$ Conversely, the non-emptiness of $\mathcal{G}$ implies that AIS$_i$ is nonempty for any $i$. Thus, the local opacity is guaranteed. Furthermore, since $\mathcal{G}$, after pruning, encodes all the valid privately enforcing insertion functions for each intruder such that the joint state estimate never reveals the secret, J-CSO is guaranteed.

7.5.3 Complexity Analysis

Given the system $\mathcal{G}$ with $|\mathcal{Q}|$ states, the space and time complexity to construct each AIS is polynomial with $|Q_{CE}|$ [194], where $|Q_{CE}| = |Q_{Obs}| = 2^{|\mathcal{Q}|}$ denotes the total number of states of the (forward) state estimator. In addition, for each intruder $I_i \ (i \in \mathcal{N})$, it follows from the construction procedure that the size of the state space of the insertion NFM will be no more than the state space of its corresponding AIS. Therefore, the space complexity to construct $\mathcal{G}$ is polynomial in $|Q_{CE}|$ and exponential in $|\mathcal{N}|$. The pruning process, in the worst case, goes through all the states in $\mathcal{G}$ as well as the intermediate states, which is also polynomial in $|Q_{CE}|$ and exponential in $|\mathcal{N}|$. In summary, the space and time complexity in our proposed centralized synthesis approach is both polynomial with $|Q_{CE}|$ and exponential in $|\mathcal{N}|$.

7.6 Conclusion

In this chapter, we investigate the opacity-enforcement problem for discrete event systems that can be observed by multiple intruders. The major contribution of this chapter are summarized as follows. First, we introduce two types of opacity notions for DESs
in the presence of multiple intruders, one with coordination and the other without coor-
dination. Next, we adopt the event insertion mechanism in the literature to ensure de-
centralized opacity for intruders without coordination; the synthesized insertion functions
are further refined to enforce joint opacity of the DES when intruders can coordinate vi-
a an intersection-based protocol. Finally, the computational complexity of the proposed
opacity-enforcement mechanisms is studied.
CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this dissertation, we have investigated the formal synthesis problem of coordination and control policies for large-scale engineering systems from an autonomous Cyber-Physical System perspective. On the one hand, by capturing the higher-level and logical behaviors of the underlying autonomous cyber-physical systems as Discrete Event Systems, we have developed novel approaches for solving the coordination and control problems for systems with distributed architectures. On the other hand, facing the low-level and physical evolutions of the systems, we have presented a path planning scheme for systems whose dynamics are subject to nonholonomic constraints. The results developed in this dissertation provide not only new theoretical foundations and computationally efficient algorithms for synthesizing correct-by-construction logical supervisors in the cyber part that accomplish a set of qualitative requirements, but also dynamics-based controllers that drive the physical part of the systems to fulfill prescribed performance objectives in a provably correct manner.

More specifically, for the coordination and control problem of distributed CPSs whose logical behaviors can be modeled as finite automata, a uniform coordination and control framework has been presented to synthesize local supervisors as well as coordination policies among various system components. The proposed uniform framework is carried out based on the integration of supervisory control theory and compositional verification in the model checking literature. Furthermore, to deal with potential uncertainties from the
system models and/or from the environment in which the systems perform high-level behaviors, we have modified the $L^*$ learning algorithm to synthesize satisfactory local supervisors even in the absence of prior model knowledge of the uncontrolled subsystems. We have also taken advantage of the modified $L^*$ learning algorithm in the compositional verification procedure, and by following an assume-guarantee paradigm, the coordination of different subsystems can be achieved.

Furthermore, the resilience and security properties of the CPSs have been investigated regarding to the proposed framework. On the one hand, by introducing automata-theoretic methods to characterize the behaviors of each subsystem that is affected by sensor and/or actuator faults, appropriate post-fault supervisors are synthesized such that local safety can be ensured; and an assume-guarantee post-fault coordination scheme is exploited to maintain the global specification. On the other hand, we have also investigated the opacity-enforcement problem for DESs that may be observed by multiple intruders. When the given opacity notion fails to hold, we adopt the event insertion mechanism to ensure decentralized opacity for intruders without coordination; the synthesized insertion functions are further refined to enforce joint opacity of the DESs when intruders can coordinate via an intersection-based protocol.

Finally, regarding to the nonholonomic dynamics of the subsystems, we have considered schemes to enable real-time computation of sub-optimal continuous-curvature paths for steering car-like robots with boundedness on both path curvature and its time derivative. Based on the $\mu$–tangency conditions [63], we have established geometric conditions for the existence of the sub-optimal continuous-curvature paths. The proposed existence conditions can be used to determine whether a sub-optimal path with a specific driving pattern exists, upon which computationally efficient CC paths can be extracted.
8.2 Future Work

The research results presented in this dissertation in fact open several research directions for future work. First, the frameworks proposed in Chapter 4 and Chapter 5, which aim at achieving resilient coordination and control of distributed CPSs whose high-level behaviors could be modeled as finite automata, are developed for (prefix-closed) regular language specifications for the generated behaviors of the underlying (sub)systems. One interesting research direction may be to consider the marked behaviors of the systems and take non-blockingness into consideration. Furthermore, in some applications, different marked states of a system may represent different types of performance objectives and one may be interested in synthesizing a supervisor such that different marked states can be visited in an orderly manner. Although similar problems have been studied in the context of multitasking supervisory control [50] by assuming that all the events are (locally) observable, how to apply the results for partially-observed systems still remains elusive.

Despite the proposed resilient coordination and control framework for regular language specifications, our future work is also interested in the exploration of supervisor synthesis algorithms and coordination techniques that drive a distributed CPS to satisfy non-terminating specifications such as LTL formulas [7]. This future work is motivated by the fact that infinite-length behaviors also play an important role in supervisory control problems, especially characterizing progress or liveness properties of a system, i.e., those properties which state that some desired conditions must occur infinitely often, which generally are not able to be captured by finite-length behaviors. Temporal logic provides us an effective means of describing such types of specifications, and in most cases, the translation of a simple natural language specification into temporal logic one is quite straightforward. Although the problem of synthesizing a controller that fulfills certain class of LTL specifications such as safety [94] and reactivity (GR(1)) [137] has been investigated in the literature, how to coordinate the controlled subsystems in such a way that a global specification can be accomplished collaboratively is also an interesting future direction.
Thirdly, with regarding to the opacity-enforcement mechanisms of CPSs under observations of multiple intruders that are presented in Chapter 7, it would also be interesting to study the opacity notions for systems with distributed architectures. More specifically, the system $G$ is assumed to be composed of multiple subsystems, i.e., $G = G_1 \parallel G_2 \parallel \cdots \parallel G_n$. Each subsystem $G_i$ ($i \in I$) possesses its own set $Q_{i,s}$ of local secret states, while a global secret state of $G$ can be deemed as a composed state $q = (q_1, q_2, \ldots, q_n)$ in which for each $i \in I$, $q_i \in Q_{i,s}$. In this case, even if each subsystem $G_i$ ($i \in I$) can be individually opaque, the composed system $G$ may still reveal secret behaviors due to the fact that some non-secret behaviors are disabled after the parallel composition. The distributed opacity is similar to the notion of concurrent opacity presented in [6], where in the latter global secret was not studied. The authors of [149] developed verification algorithms for the distributed opacity notions; however, how to enforce the distributed opacity when the system fails to be opaque is a challenging problem in the future.

Finally, we need to investigate more practical case studies in the future. It would be of great interest to apply the results and techniques developed in this thesis to construction of distributed CPSs that achieve certain resilience and security objectives. Examples include, but not limited to, intelligent transportation systems, flexible manufacturing systems and autonomous vehicle teams.


