Synthesis of Communicating Decentralized Supervisors for Discrete-Event Systems with Application to Communication Protocol Synthesis

Amin Mannani

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

June 2009

Your file Votre référence
ISBN: 978-0-494-63347-2
Our file Notre référence
ISBN: 978-0-494-63347-2

ABSTRACT

Synthesis of Communicating Decentralized Supervisors for Discrete-Event Systems with Application to Communication Protocol Synthesis

Amin Mannani, Ph.D.

Concordia Unviersity, 2009

A Discrete-Event Systems (DES) may be viewed as a dynamic system with a discrete state space and a discrete state-transition structure with an event-driven nature, which makes it different from the systems described by differential or difference equations. Given the desired behavior of a DES as a specification, decentralized supervisory control theory seeks to design for a (distributed) DES, consisting of a number of (geographically distant) sites, a set of supervisors, one for each site, such that the behavior of the DES always remains within the specification. If the specification is not coobservable, these supervisors need to communicate amongst each other.

This thesis proposes a mathematical framework to formally model and synthesize such communicating decentralized supervisors. The framework provides a decentralized representation of the DES's centralized supervisor and captures its observational and control-related information as mappings, which are called updating and guard functions, respectively. This leads to a polynomial dynamical system, which serves to model the required communication and synthesize its rules. The systematic synthesis, obtained through this approach, characterizes the class of distributed control problems which are solvable only with communication, comes up with a finer partition of it, and addresses practical issues. The thesis ends with the application of the theoretical results to the modeling and synthesis of a communication protocol.

To my parents, Jalil and Masoomeh

for their love and support all the way since the very first days of my life

To my wife, Elham,

for her wise decisions, patience, and honest friendship, and

To my sweet daughter, Darya,

for all the hope and joy she gifted me

ACKNOWLEDGEMENTS

"A hundred times everyday I remind myself that my inner and outer life depend on the labors of other men, living and dead, and that I must exert myself in order to give in the same measure as I have received and am still receiving."

Albert Einstein, *The World As I See It*

My life, and especially its scientific part, has been shaped by many people directly or indirectly. I have always learned from my teachers, my family, my friends, known and anonymous reviewers of my work, and historical scientific faces. They have influenced my way of critical thinking, attitude toward learning, and patience against hardships. Indeed, I am indebted forever to all of them and do not know of any ways to compensate this gifted knowledge than conveying it to the future generation.

My PhD thesis is one of the greatest achievements that I have had in my life and was impossible without others' help. Within the limited space, I will try to acknowledge those who helped me accomplish my research.

I am especially thankful to my supervisor, Dr. Peyman Gohari, who unfortunately passed nine months before I defended my thesis. Peyman was a very kind and frank friend, and a responsible, liberal, knowledgeable, and humble supervisor. His careful scientific observations and profound understanding of this work, together with his punctuality and commitment to high academic standards were what I required to open my mind and deepen my understanding. He devoted hundreds of hours of his time to supervise my research, either in the form of our weekly meetings, which sometimes happened three times a week, or in reviewing my work and giving valuable feedback on

it. His moral and financial supports provided me with an atmosphere in which I loved to do research in its most original and pleasant way. He wished to see my defense and achievements. Whereas I missed a lot his physical presence in my defense, he was always present in my mind when explaining every aspect of my work. The content of this thesis reflects only part of what we discussed during these years and I hope that I follow his way of thought and valuable suggestions in future.

In the absence of Dr. Gohari, Dr. S. Hashtrudi Zad kindly accepted to be my administrative supervisor. I would like to thank very much his support and efforts for following up my research progress and arrangements of the defense session. I am very grateful for having a wonderful doctoral committee and wish to specifically thank the external-to-the-university committee member, Dr. J. G. Thistle, and the external-to-the-department committee member, Dr. S. Klasa, for their very careful reading of the thesis and the detailed comments. I would also like to thank my internal committee members, Dr. K. Khorasani and Dr. F. Khendek, for their support in the absence of Dr. Gohari and for the valuable suggestions they have made since I presented my research proposal in 2006. I appreciate the insightful discussions I had with Dr. S. L. Ricker, Dr. W. M. Wonham, and Dr. A. Vetta.

I would like to express my sincere thanks to the present and previous staff of Electrical and Computer Engineering Department at Concordia University for their hospitality, responsibility, and patience. I would like to name Ms. Pamela J. Fox and Ms. Connie Cianciarelli for their kindness, and Ms. Kathy Kirnan and Ms. Lyne DesLauriers for their support from the early days of my arrival at Concordia.

I am very grateful to all my friends in Canada and Iran who are always a great source of support. This especially goes to Mr. Mohsen Zamani-Fekri and

Mr. Hani Khoshdel-Nikkhoo, and Mr. Nader Meskin for their kindness and support in the course of my PhD study. I wish it were possible to name all my great friends but I confine myself to the following list: B. Akbarpour, H. Assa, A. Azimi, M. Azizi, M. Benjillali, K. Bouyoucef, S. Farzaneh, M. Gharaati, O. Hassan, M. Hosseini, V. Janarthanan, A. Mohammadi, A. Momeni, M. Muniruzzaman, M. Rahnamaei, M. Sadid, M. Salimian, B. Samadi, and Y. Yang.

I gratefully acknowledge my debt to my family members for their incredible passion, love, and dedication. My wife and my best friend, Elham, has been a source of hope, love, and enthusiasm in my life. She has helped me overcome the difficulties of life and managed to provide a lovely atmosphere at home. Her presence and sense of responsibility made me confident in pursuing my study and her wise decisions and passion for life helped me pass the most difficult moments successfully and calmly. My special gratitude goes to my parents for their continuous dedication to their children's life and for the wisdom and love they have gifted them. I hope I can do the same when it comes to my turn. I wish to thank my brother, Ali, and my sister, Leila, and their families, and my family in-law for their love and support. I would like to gratify the memory of my uncle, Keramat, who always was willing to hear from my achievements and encourage me for further steps. I would like to sweeten this acknowledgement by thanking Darya, my beloved pearl, whose presence in my life is a source of endless hope and joy and a sign of God's blessing.

To myself I am only a child playing on the beach, while vast oceans of truth lie undiscovered before me.

Isaac Newton (1642 - 1727)

TABLE OF CONTENTS

List of Figures xiv

List of Tables xxi

List of Symbols and Abbreviations xxiv

1 Introduction **1**

1.1 Discrete-event systems and supervisory control theory 1

1.2 Distributed discrete-event systems 4

1.3 Communicating supervisors 8

1.3.1 The issue of time in communication 15

1.4 Inferencing supervisors 16

1.5 The perspective of this research on SCDS 20

1.6 EFSM and SDES frameworks 24

1.7 Symbolic systems and PDSs 29

1.8 Other formalisms to study decentralized control 31

1.9 Problem statement and basic assumptions 32

1.10 Research achievements 35

1.11 Synthesis of communication protocols 38

1.11.1 The application-oriented contribution of the thesis . . . 40

1.12 Outline of the thesis 44

1.12.1 Some conventions used in mathematical proofs 46

2 Background **48**

2.1 Introduction 48

2.2 Model of a discrete-event system 48

2.3 Extended finite-state machines 52

2.3.1 Preliminaries 52

2.3.2 Implementation of supervisory control map by an EFSM 53
2.4 Decentralized embedded supervisory control 55
2.4.1 Plant model and problem definition 55
2.4.2 Solution: coobservable specifications 57
2.5 Conclusion 61

3 Decentralized Supervisory Control of Discrete-Event Systems over Communication Networks 62

3.1 Introduction 62
3.2 Extended finite-state machines 65
3.3 Problem statement 66
3.3.1 Processes 67
3.3.2 Channels 69
3.3.3 Control problem 70
3.4 DECCN solution—special case 71
3.4.1 ABP: Problem formulation in EFSM framework 72
3.4.2 Solution 75
3.5 Towards the general problem in the presence of ideal channels 81
3.6 DECCN solution—unreliable channels 98
3.7 Conclusion 103

4 The Framework of Supervised Discrete-Event Systems 106

4.1 Introduction 106
4.2 Supervised DESs 109
4.2.1 The formalism of supervised DESs 109
4.2.2 SDESs and centralized supervision 110
4.3 Distributed SDESs and decentralized supervisors 113

4.3.1 Distributed SDESs and synthesis of decentralized supervisors . . . 114
4.3.2 Agent-wise labeling maps and design of updating and guard functions . . . 116
4.4 Guard and updating functions as polynomials over a finite field 120
4.4.1 Finite field representation of a DSDES . . . 122
4.4.2 EFSM implementation of a DSDES . . . 129
4.5 A note on computational complexity . . . 135
4.6 Conclusion . . . 137

5 Modeling and Synthesis of Communication within SDES Framework 138

5.1 Introduction . . . 138
5.2 Communication as reevaluation of guard and updating functions 139
5.2.1 Communication-related events . . . 140
5.2.2 Communication policies . . . 144
5.2.3 Two simple communication policies . . . 146
5.2.4 Communication policies which prescribe less communication . . . 150
5.2.5 Communication policies which prescribe communication of encoded events . . . 165
5.2.6 Synthesis of communicating supervisors using DSDESs: a summary . . . 174
5.3 A communication-based classification of DSDESs . . . 175
5.3.1 Communication for observation and communication for control . . . 175
5.3.2 PDSs with independent guard functions . . . 179
5.3.3 PDSs with independent updating functions . . . 185

5.4 IUFs, DSDES behavior, and state representations 188
5.4.1 Weak joint observability 188
5.4.2 Undecidability of verification of weak joint observability 191
5.4.3 Weak joint observability as a necessary condition for the existence of IUFs 193
5.4.4 Some state representations which lead to IUFs 194
5.4.5 Construction of IUF-yielding ALMs 198
5.5 Conclusion 206

6 Case Study **208**
6.1 Introduction 208
6.2 An overview of RMTP 210
6.2.1 Terminology 211
6.2.2 Basic functions of RMTP 212
6.3 Modeling of RMTP's basic functions for a simple network . . . 214
6.3.1 Network topology and control nodes 215
6.3.2 Definition of events 217
6.3.3 Modeling the control nodes 222
6.3.4 RMTP control specifications 230
6.4 Synthesis of the RMTP-like information policies for the simple network 234
6.4.1 Centralized supervisors for RMTP control specifications 235
6.4.2 DSDES model of the modular supervisors 238
6.4.3 PDS representation of the RMTP network 240
6.4.4 RMTP-like state-transferring information policy for the RMTP network 244
6.4.5 RMTP-like event-transferring information policy for the RMTP network 250

6.5 Conclusion 256

7 Conclusions and future work **259**

7.1 Conclusions 260

7.2 Future work 263

References 266

A Missing proofs of the claims of Chapter 4 **278**

B Missing proofs of the claims of Chapter 5 **284**

C Details of computations in Chapter 6 **315**

C.0.1 Labeling of events 315

C.0.2 Network modeling in TCT 317

C.0.3 Application of supervisory control theory using TCT . 317

List of Figures

1.1 Traditional approach to protocol synthesis. 41

1.2 A first attempt at protocol synthesis using supervisory control theory based on coobservability. 43

1.3 The proposed approach based on supervisory control theory and DSDES framework. Dotted lines indicate a possibly required communication between two supervisors. 45

2.1 (a) Decentralized embedded supervisory control with a coobservable specification: component plants and specification. (b) Plants $\mathbf{G}$, $\tilde{\mathbf{G}}_\mathbf{1}$ and $\tilde{\mathbf{G}}_\mathbf{2}$. 59

2.2 (a) Supervisors $\tilde{\mathbf{S}}_\mathbf{1}$ and $\tilde{\mathbf{S}}_\mathbf{2}$, and embedded controllers $\tilde{\mathbf{G}}_{\mathbf{1x}}$ and $\tilde{\mathbf{G}}_{\mathbf{2x}}$. (b) Overall controlled system $\mathbf{G}_{\mathbf{1x}} \parallel \mathbf{G}_{\mathbf{2x}}$. 60

2.3 A specification that is not coobservable. 61

3.1 A network of n communicating parallel processes ($n = 4$). Each process $\mathbf{P}_{\mathbf{ix}}$ has a number of observable, unobservable and communication-related events. A process may be connected to others through ideal (bold arrows) or potentially unreliable (regular arrows) channels. 67

3.2 An unreliable channel. 70

3.3 Two processes $\mathbf{P}_\mathbf{1}$ and $\mathbf{P}_\mathbf{2}$ communicating over a channel. . . 73

3.4 Schematic of the plant. 74

3.5 Plant FSMs and the requirement specification. $\Sigma_1 = \{\alpha_1, \beta_{12}^s, \beta_{21}^e, \beta_{21}^r\}$ and $\Sigma_2 = \{\alpha_2, \beta_{21}^s, \beta_{12}^e, \beta_{12}^r\}$. 74

3.6 The centralized supervisor $\mathbf{E}_\mathbf{1}$ is state-independent while $\mathbf{E}_\mathbf{2}$ is not. 80

3.7 The centralized supervisor **E** of the ABP is state-independent. 80

3.8 The centralized supervisor **S**, with $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $\Sigma_{uo,i} = \emptyset$, $i = 1, 2$, is not state-independent when one Boolean variable is used to count α_1, while it becomes state-independent when two Boolean variables are used to count α_1. 82

3.9 An example of an I-connected automaton. 84

3.10 A Latin cube with $n = 3$ (the number of axes) and $m = 3$ (the number of objects): There exists exactly one copy of each object in every direction. 85

3.11 (a) A centralized supervisor and (b) its unfolded version. (c) Graphical representation of a finite ALM. (d) The encoded supervisor. 87

3.12 $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $i = 1, 2$. (a) A language that is not weakly jointly observable. (b) A state-independent centralized supervisor yielding independent actions. 93

3.13 (a) A centralized supervisor and (b) its labels assigned by an ALM. (c) The estimator structure (without communication) for part (a) (reprinted from Fig. 3 in [1]). 98

3.14 The centralized supervisor of Example 3.9. 100

3.15 Four processes in a deterministic network and the centralized supervisor **S**. 103

3.16 The complete design for the system of Fig. 3.15. 104

3.17 ABP design in EFSM framework. 104

4.1 (a) Plant **G**, (b) specification **E**, and (c) supervisor **S** with labeled states. 113

4.2 (a) The plant's model. (b) The specification. (c) The modified centralized supervisor. (d) The ALM-assigned labels; a point $\mathbf{v} \in \mathbb{N}^3$ is labeled with $r \in R$ if and only if $\mathbf{v} \in \ell(r)$; thus, for example, $\ell(r_4) = \{(201), (111), (021)\}$. 119

5.1 (a) Plant $\mathbf{G}$, (b) specification and centralized supervisor $\mathbf{S}$, and (c) a graphical representation of the ALM for $\mathbf{S}$. 152

5.2 Graphical representation of $f(x_1, x_2) = 2x_1[x_1 + 1 + 2(x_1 + 2)(x_2 + 1)^2]$ and equivalent classes. 154

5.3 Graphical representation of a function f with symmetric pattern with respect to v_2 axis. 162

5.4 (a) A plant $\mathbf{G}$. (b) A specification and centralized supervisor $\mathbf{S}$. 182

5.5 (a) Recognizers $\mathbf{A}_1$ and $\mathbf{A}_2$. (b) The modified recognizers $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ whose states are labeled. (c) The meet of the modified recognizers whose states are labeled by the joint labeling of the ALMs for $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$. 196

5.6 (a), (b) Two Latin squares of side $m = 5$ with $\Gamma = \{r_0, r_1, r_2, r_3, r_4\}$. v_1 and v_2 denote the horizontal and vertical components of a vector v assigned by an ALM to a state r_i, $(i = 0, 1, \cdots, m-1)$, respectively. (c) An example of an automaton whose states can be labeled by the Latin squares in parts (a) and (b) ($\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$). 199

5.7 (a) An automaton with $\Sigma_{o,1} = \{\alpha_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) A Latin square which defines an ALM yielding IUFs for α_1 and α_2. . . 203

5.8 (a) An automaton with $\Sigma_{o,1} = \{\alpha_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) A Latin square which defines an ALM yielding IUF for α_1, but not for α_2. 203

5.9 (a) An automaton with $\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b), (c) Two Latin squares which defines ALMs yielding IUFs for α_1, and β_1, but not for α_2. (d) A Latin square which embeds the Latin squares in parts (b) and (c) and satisfies the conditions in Problem 5.3. copies of the square corresponding to $\eta_{\alpha_1}^{1,1}$ are enclosed with dashed lines, and symbols of the two copies corresponding to $\eta_{\beta_1}^{1,1}$ are marked with underlines and overlines. . 205

5.10 (a) An automaton with $\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) The columns of the left Latin square, which defines an ALM yielding IUF for β_1, are suitably mapped to columns of the second square, also yielding IUF for β_1, to define the part of the ALM which yields IUF for α_1. The top and bottom arrows represent β_1- and α_1-related transitions, respectively. 206

6.1 An RMTP tree (reprinted from [2]). Arrows specify the direction of data and control signals flow. 213

6.2 An asymmetrical network consisting of 7 control nodes. 216

6.3 The automaton model of sender **SD**. 225

6.4 The automaton model of receiver $\mathbf{RN}_i$, where $i \in \{1, 2, 3\}$. . . 226

6.5 Automaton model of $\mathbf{DR}_2$. A selfloop labeled with R_{DR2}^e should be added to every state of the automaton. 227

6.6 Automaton model of $\mathbf{DR}_1$. A selfloop labeled with R_{DR1}^e should be added to every state of the automaton. 228

6.7 The automaton model of top node **TN**. 229

6.8 (a), (b) The control specifications for the joint behavior of $\mathbf{DR}_1$ and its children, $\mathbf{RN}_1$ and $\mathbf{RN}_2$, respectively. All of the events in $\Sigma \setminus \{A_{RN1}, A^e_{RN1}, RA_{RN1}, RA^e_{RN1}\}$ and every event in $\Sigma \setminus \{A_{RN2}, A^e_{RN2}, RA_{RN2}, RA^e_{RN2}\}$ are selflooped at all states of $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. These selfloops are not shown to keep the figures clear. (c) Two simple requirements for the alternation of $\mathbf{RN}_1$'s two types of acknowledgements and $\mathbf{DR}_1$'s corresponding acknowledgement-received events. (d) Synchronous application of the simple requirements in part (c) allows the serial issuance of multiple acknowledgements by $\mathbf{RN}_1$ without processing them by $\mathbf{DR}_1$, hence justifying the introduction of specification $\mathbf{S}_1$ in part (a). 232

6.9 The control specification for the joint behavior of $\mathbf{DR}_2$ and its child. All of the events in $\Sigma \setminus \{A_{RN3}, A^e_{RN3}, RA_{RN3}, RA^e_{RN3}\}$ are selflooped at all states, but not shown. 232

6.10 The control specifications for the joint behavior of **TN** and its children $\mathbf{DR}_1$ and $\mathbf{DR}_2$, respectively. All of the events in $\Sigma \setminus \{A_{DR1}, A^e_{DR1}, RA_{DR1}, RA^e_{DR1}\}$ and $\Sigma \setminus \{A_{DR2}, A^e_{DR2}, RA_{DR2}, RA^e_{DR2}\}$ are selflooped at all states of the automata in parts (a) and (b), respectively. These selfloops are not shown to keep the figures clear. 233

6.11 The control specification for the joint behavior of **SD** and **TN**. All of the events in $\Sigma \setminus \{A_{TN}, A^e_{TN}, RA_{TN}, RA^e_{TN}\}$ are selflooped at all states, but not shown. 234

6.12 (a) General model of the modular supervisors, where $*$ denotes the events in the set $\Sigma \setminus \{\alpha_i, \beta_i, \alpha_j, \beta_j\}$. (b) One arrangement of the Latin squares of side 3 and dimension 2, which is suitable for defining an ALM for each modular supervisor. 239

B.1 Latin square LS_1 corresponds to $\eta_\alpha^{i,1}$ and is used to construct an ALM which yields IUF for $\alpha \in \Sigma_{o,1}$. 310

B.2 Construction of an ALM yielding IUFs from Latin squares corresponding to disjoint tuples. (a) Construction of a rectangle of length 6 from Latin squares of sides 3 and 2. (b) Forming the rectangle of size 5 by 6 by putting one rectangle on top of the other one. (c) A compact representation of rectangle 3. (d) Distributing the rows of rectangle 1. (e) Distributing the rows of rectangle 2. (f) The general form of the rectangle used to build an ALM yielding IUF for α. (g) A reduced design. . . . 311

C.1 TCT model of control node **SD** 330

C.2 TCT model of control node **TN** 330

C.3 TCT model of control node **DR1** 331

C.4 TCT model of control node **DR2** 331

C.5 TCT model of control node **RN1** 332

C.6 TCT model of control node **RN2** 332

C.7 TCT model of control node **RN3** 332

C.8 TCT model of specification $\mathbf{S}_1$ 333

C.9 TCT model of specification $\mathbf{S}_2$ 335

C.10 TCT model of specification $\mathbf{S}_3$ 336

C.11 TCT model of specification $\mathbf{S}_4$ 337

C.12 TCT model of specification $\mathbf{S}_5$ 338

C.13 TCT model of specification $\mathbf{S}_6$ 339

List of Tables

3.1 System events. 73

4.1 Updating functions ($a, b \in \{0, 1, 2\}$, $c \in \{0, 1\}$) for the DSDES in Example 4.2 121

4.2 Computed polynomials for updating and guard functions the DSDES in Example 4.2 127

4.3 Actions and guards in EFSM framework 135

5.1 Communication amongst supervisors based on policy 1 148

5.2 Communication amongst supervisors based on policy 2 149

5.3 Updating functions for the DSDES in Example 5.2 152

5.4 Computed polynomials for updating and guard functions of the DSDES in Example 5.2 153

5.5 Actions and guards in EFSM framework for supervisors in Example 5.2 159

5.6 Communication amongst supervisors in Example 5.2 based on policy 2 160

5.7 Communication amongst supervisors in Example 5.2 based on policy 3 161

5.8 Encoding of Events for Example 5.6 172

5.9 Communication amongst supervisors in Example 5.6 based on policy 5 173

5.10 Updating functions for the DSDES in Example 5.7 182

5.11 Computed polynomials for updating and guard functions of the DSDES in Example 5.7 183

5.12 Actions and guards for Example 5.7 in EFSM framework . . . 185

5.13 Communication amongst supervisors in Example 5.7 based on policy 6 185
5.14 Updating functions for $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ of Example 5.9. 195

6.1 Summary of the events used to model the RMTP network. Here $i \in \{1,2,3\}$ and $j \in \{1,2\}$. 223
6.2 Event subalphabets for control nodes ($i \in \{1,2,3\}$). 224
6.3 Updating functions associated with $\mathbf{S}_k$ and ℓ 240
6.4 Participation of control nodes in the six modular supervisors . 243
6.5 Private variables owned by control nodes when participating in different modular supervisors 243
6.6 The PDS representation of the RMTP network 245
6.7 Private Boolean variables owned by control nodes for state-transferring information policy 246
6.8 Actions of the control nodes of the RMTP network 248
6.9 Guards of the control nodes of the RMTP network 249
6.10 Communication amongst supervisors based on policy 2 251
6.11 Communication amongst supervisors based on policy 2 (Part 2) 252
6.12 Communication amongst supervisors based on policy 2 (Part 3) 253
6.13 Details of the encoding of the events for communication policy 5 255
6.14 Communication amongst supervisors based on policy 5 (Part 1) 257
6.15 Communication amongst supervisors based on policy 5 (Part 2) 258

C.1 Event numbering used in TCT 316
C.2 The code to generate the whole network. 318
C.3 The code to verify the existence of a controller for each specification. 320

C.4 The code to verify that each specification is nonblocking and controllable. 321
C.5 The code to verify that each specification is controllable. . . . 322
C.6 The code to verify that each specification is nonconflicting with respect to the network. 324
C.7 The code to verify that supervisors are mutually nonconflicting (Part 1). 327
C.8 The code to verify that supervisors are mutually nonconflicting (Part 2). 328
C.9 The code to verify that supervisors are mutually nonconflicting (Part 3). 329

List of Symbols and Abbreviations

ABP	Alternating Bit Protocol
ALM	Agent-wise Labeling Map
ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
CNF	Conjunctive Normal Form
DECCN	Discrete-Event Control over Communication Networks
DES	Discrete-Event System
DSDES	Distributed Supervised Discrete-Event System
EFSM	Extended Finite-State Machine
EGPZT	Embedded Global Problem with Zero Tolerance
FSM	Finite State Machine
GLM	Global Labeling Map
GPZT	Global Problem with Zero Tolerance
IDD	Integer Decision Diagram
PDS	Polynomial Dynamical System
PLC	Programmable Logic Controller
RMTP	Relible Multicast Transport Protocol
RW	Ramadge-Wonham (supervisory control theory)
SCDS	Synthesis of Communicating Decentralized Supervisors
SCT	Supervisory Control Theory
SDES	Supervised Discrete-Event System
TCP-IP	Transmission Control Protocol-Internet Protocol

Chapter 1

Introduction

1.1 Discrete-event systems and supervisory control theory

Consider the daily operation of an airport with aircraft, vehicles, and other logistics providing different services for passengers and luggage. As the second example, in an automated manufacturing plant there are plenty of jobs, each bearing the notion of an event, done by robotic manipulators on some input lines and using some common plant resources. Similarly, in a software program the resources of a computer are used by each module of the software to perform various operations or provide services to environmental agents again. A no less important example is the functioning of many worldwide telecommunication or computer networks making possible easy phone calls and online access to various economic, health, and other data within a short time. All these "jobs" require different kinds of "services," each being distinguished by a sequence of events done by a number of plants (agents), utilizing a multitude of "resources" that interact with each other continuously and dynamically. These are among the endless numbers of "Discrete-Event Systems" (DESs).

DESs may be considered as discrete-time dynamic systems with (often discrete) state spaces and discrete state-transition structures. Their evolution is event-driven (or asynchronous) by means of events (sometimes including tick of a clock) in a nondeterministic fashion, i.e. the system may choose among its different available transitions using an internal mechanism or one which is not modeled by the system analyst. Being primarily modeled by rules of logic, DESs are inherently different from time-driven systems which are modeled and described by differential or difference equations [3].

Such multidisciplinary systems are interesting to researchers in control and industrial engineering, computer and communication sciences and operation research. They can happen in a variety of application domains such as manufacturing and software systems and communication networks. Other applications include the abstract (logical) models of physical systems.

For a long time, the informaticians and computer scientists have been investigating the "analysis" of DESs, which they call "verification", and their attempts have resulted in industrial size verification tools especially for hardware [4]. The synthesis problem, which is usually called "control" by control community, on the other hand, has not been inspected much perhaps due to the inherent complexity and lack of suitable mathematical tools [5].

To deal with "synthesis" problems, P.J. Ramadge and W.M. Wonham introduced an abstract model of controlled DES in 1982, which is referred to as RW framework. Their model, which is automaton- or (dually) language-based [6], represents a DES as the generator of a formal language, some of whose events (or transitions) can be prevented by an external agent. The generator, its corresponding formal language, and the events that can be prevented are called "plant," the "behavior" of the system, and the "controllable events," respectively. This model captures the basic control-theoretic notions

such as controllability and observability and provides an approach to the control problem of DES, known as *supervisory control theory* (SCT). The approach is optimal in the sense that it synthesizes another DES, called the "controller" or "supervisor,"[1] which "minimally restricts" the behavior of the plant by disabling some of its transitions, which are labeled by controllable events, at different states of the plant's automaton. This is done in such a way that the language generated by the synchronous operations of the plant and the controller respects a "given specification, " which might itself be stated as desired safety or liveness properties on the alphabet of the plant. In other words, the behavior of the synchronous product of the plant and the controller, which is called the "closed-loop language," is a subset of the specification language. Therefore the closed-loop system is correct by construction, eliminating the need for after-design verification. Accordingly, a string is legal if it is entirely within the legal behavior. A transition that takes a string from legal to illegal behavior is referred to as (one step) *illegal move*. Thus a controller essentially prohibits the illegal behavior through avoiding the illegal moves.

The events of a system might be observable by a controller or not, i.e. a controller can or cannot disambiguate for itself if a particular event has happened. By supervisory control theory, in the case that all events are observable, a controller guaranteeing the satisfaction of a given specification, always exists if and only if the specification is *controllable* with respect to the plant[2] [6]. In simple terms, controllability requires that no one-step continuation of the

[1]These two terms are used interchangeably in this thesis.

[2]To guarantee that the closed-loop behavior is deadlock-free (or nonblocking), another condition, called L-closure with L being the plant's language, is also necessary and sufficient. This condition requires that all the prefix-closed strings in the given specification, which are marked by the plant, are marked in the specification, too. This issue is not directly related to the supervisor's observation and communication, which are the central topics of the discussion here and, this is not emphasized in this chapter. To justify this neglect, it can be safely assumed, for now, that the given specifications are prefix-closed. This assumption guarantees that L-closure is automatically satisfied (see [3]). In later chapters, the issue of nonblocking is addressed, too.

plant's legal behavior by uncontrollable events violate the specification. It has been shown that this property can be verified efficiently. Controllability is closed under the union operation [6], therefore, for a given specification one can always efficiently compute the *supremal* controllable sublanguage of the specification language (or supervisor) [6]. When some events are unobservable, a controller enforcing the specification exists if and only if the specification is controllable with respect to plant and observable with respect to the plant and the corresponding observational map (modeled as natural projection) [7], [8]. Observability necessitates that any two lookalike legal strings have the same one-step continuation by any event, in the sense that both lead to either legal or illegal behavior with respect to the specification. Checking observability can also be done in polynomial time [9]. It is only closed under intersection but not under union [8], therefore, for a general (controllable) specification, only its *infimal* observable sublanguage is guaranteed to exist, and in general there may be incomparable *maximal* closed-loop languages.

1.2 Distributed discrete-event systems

Since the introduction of supervisory control theory in the 80's, continuous research has been done to extend the basic theory to various "realistic" cases such as decentralized, hierarchical, and timed structures, and to improve its associated computational efficiency [3]. In particular, the decentralized supervisory control, which deals with the controller synthesis for "distributed" systems, has been the focus of attention since the beginning of the 1990's.

A distributed DES may consist of several, say[3] $n \in \mathbb{N}$, "component plants" or "agents", each associated with part of the tasks to be performed

[3] $\mathbb{N}$ is the set of natural numbers.

by the system. The language generated by the overall system is the synchronous product of the languages of component plants. Such an architecture is sometimes referred to as a "multi-agent" architecture in the literature.

For such a distributed system if a specification is given for each component, the control problem is reduced to n separate control problems, one for each component plant and its corresponding "local" specification. However, to apply this approach, which is usually called *modular supervisory control*, one has to be aware of the fact that these local specifications might be *conflicting*, i.e. their corresponding separately designed controllers might result in a blocking behavior or deadlock [10].

In general, specifications of a distributed DES determine the legal behavior of the whole system. These so called "global specifications" determine the tasks of each component and the interactions among them. If such a global specification can be decomposed into "local" specifications for different agents, the problem again is reduced to the modular supervisory control problem explained above [11]. However if no such decomposition can be envisaged, the control problem is called *decentralized supervisory control* in which a set of controllers, one for each component plant, should be designed such that the closed-loop behavior of the overall system, consisting of the component plants and their controllers, meet the specification [12].

The decentralized supervisory control problem usually arises in cases where the component plants are geographically far from each other. Apart from the lack of possibility of decomposing the global specification into local ones, the more important characteristics of a decentralized supervisory control is the agents' partial observation of each others' behaviors. While in modular supervisory control it is assumed that each component plant can observe other components' events, in decentralized supervisory control each component has

only a limited observation over the whole system's behavior and can only see part of the events. This imposes further complications on decision making for a decentralized controller, as it might observe lookalike strings whose one-step continuations by an event lead to violation of the specification for some and remains inside the legal behavior for others. Following Cieslak *et al*'s initial work in [13], Rudie and Wonham showed in [12] that there exists a set of decentralized supervisors, whose induced closed-loop language satisfies the language of the specification if and only if the specification is controllable with respect to the plant and *coobservable* with respect to the plant and the controllers' observational maps. In simple words coobservability means that for any two strings whose one-step continuations lead to conflicting behaviors, one legal and the other one illegal, there exists one supervisor which can distinguish them and exercise control over the event in question. Unfortunately coobservability is only closed under intersection and not union, therefore for a general global (controllable) specification there might be different sets of controllers each yielding incomparable maximal closed-loop languages. It was then shown that coobservability can be verified in polynomial time [14].

Another aspect of the decentralized supervisory control is how to "fuse" different decisions, made by distinct controllers about disabling (or enabling) a common event at a state of the system. Coobservability was originally formulated under the "conjunctive" fusion rule in [12] in the sense that each supervisor locally disables an event only if it is sure that the possible execution of that event, after all observationally equivalent strings, leads to a violation of the specification; otherwise it just "passes the buck" [12] to the other controllers. Following this, all such "local" decisions of the controllers are conjuncted with each other to decide about the disablement of that event.

Extensions of this work to other fusion rules was later started with introducing the notion of *D&A-coobservability*, comparing to the original notion, which was renamed *C&P-coobservability* in [15], with the hope of enlarging the class of problems solvable by employing other fusion rules. As shown in [16], the classes of solvable problems holding these two notions of coobservability are not comparable with each other. The largest class of solvable problems is shown to be *C&P∨D&A-coobservable* specifications, which enjoy both notions of coobservability with respect to two associated disjoint sets of events. Obviously the classes of solvable problems holding any such coobervability properties are always subsets of the class of problems solvable by a *centralized* supervisor, i.e. the one whose observable set of events is the union of all events which can be observed by at least one agent.

So far there are two fundamental properties which a specification must hold in order to be enforceable by a set of decentralized supervisors, i.e. controllability and (co)observability. While the former captures the idea of how far a supervisor can prohibit the plant's illegal move to limit its behavior within the given specification, the latter explores the limit of available information for the supervisor. These two correspond to "control" and "estimation" in system theory [17]. There is yet a third element which can alter a controller's estimation (or observation) of the plant and that is *communication*, by which the controllers may share their information, i.e. observations, with each other. In this way, one may expect to enlarge the class of solvable problems in decentralized supervisory control by disambiguating for supervisors the suspicious lookalike strings.

1.3 Communicating supervisors

In the absence of coobservability, the local supervisors need to communicate to meet a global specification. The problem of *Synthesis of Communicating Decentralized Supervisors* (SCDS) seeks to design a set of supervisors, one for each component plant (or agent), and a set of communication rules, which prescribe the procedure for controllers to share their information (or observation) among each other, in order to meet given global specifications. This problem has been the focus of attention for the past few years.

Before proceeding more, a note about the terminology is in order. In control community, the term "distributed control" is used for the case where controllers communicate amongst them *all* their information, and "decentralized control" usually refers to the case where no information is exchanged among controllers. Accordingly, the term "semi-decentralized" might be a better choice to name "communicating supervisors," which is the subject of this work to emphasize on the exchange of *part* of the controllers' information and distinguish it from the two extreme cases of distributed control and decentralized control. In DES community, "decentralized control" was first used in [13] and formally defined in its current meaning in [12] and was proved to exist if and only if the given specification is coobservable. On the other hand, "fully decentralized control" was later coined in [18] to refer to a more special case of the decentralized control. On the other hand, the part of DES community, including control theorists, who work on communication between supervisors, has not adopted a fixed and distinct term. They often use "decentralized control" and assume that adding communication to decentralized control should be understood from the context. In other words, there is not a general agreement between the researchers in this regard. This thesis chooses the term "communicating decentralized supervisors," which is essentially meant to be

equivalent to the term "semi-decentralized," as explained above and usually used by control community. Furthermore, it has the advantage of being understood by the DES community.

Decentralized supervisory control with communicating supervisors is motivated by the control of networks such as communication or traffic networks [19]. In [20] Wong and Van Schuppen, for the case of two controllers, formulated a necessary and sufficient solvability condition as a refinement relation between observation and control maps. This relation was proved to hold for *alternating bit protocol* (ABP), a prototype protocol which was used for communication networks. The authors then show that under certain conditions, if the problem can be solved by two communication policies each sharing the observation of a set of events, there exists another solution whose corresponding communicated event set is the intersection of those two event sets, hence establishing a first result on "minimality of the content of communication."

Teneketzis [21] views the problem in the framework of *nonsequential systems* for which the control actions cannot be ordered, *a priori*, independent of the set of control laws that determines these actions. He considers the following issues fundamental to the study of these systems (paraphrased from [21]):

1. "Who should know what and when?

2. Who should communicate with whom and when?

3. Given that communication must be limited, either because channels have limited capacity or because stations have limited memory to store data and limited processing capability, what information must be exchanged in *real time* among stations so that they can improve the quality of their actions?

4. What information should be available to each control station so that the system is deadlock-free?

5. Given that the design of highly concurrent systems is desirable, but concurrency can only increase by increasing the complexity of the system's information gathering sources, what are the fundamental tradeoffs between system concurrency and the complexity of the system's information gathering sources?

6. How does one optimize the performance of nonsequential stochastic controlled systems?"

The above issues take a key role in the direction of this research as well as many other works on SCDS.

Van Schuppen define a *decentralized supervisory control with information structure* as a supervisory control in which controllers' decisions depend only on the language of the information structure [22]. In this general framework he considers three information structures based on subsets of controllers, subsets of observable events, and subsets of observable strings, respectively, and formulates for each case the conditions under which a decentralized control with an information structure can implement a centralized controller, hence enforcing a global controllable specification. Whereas this work touches the subject in an abstract algebraic level, the idea of how control, observation, and communication are fundamentally limited to the available information structures, seems to address many relevant issues when it is implemented using detailed algebraic structures.

Barrett and Lafortune suggest a more detailed model for communication among the supervisors [1]. Assuming a two-way broadcast, the supervisor who observes the execution of an event sends the set of states it might be possibly

in, i.e. *state estimates*, to the other supervisor(s). The minimality of communication among supervisors is formulated in this work based on the rule that nothing should be communicated until it is absolutely necessary, i.e. at *the latest safe point.* Accordingly, using their model they first distinguish the *conflict states*, i.e. the states reached by lookalike strings with conflicting legality status upon the occurrence of some controllable event. Then, starting from the states which are one-step before the conflicting states, they iteratively verify if communication at preceding states can resolve the ambiguity or not. They introduce anticipatory controllers, which use full communication among them and hence can solve the problems which are solvable by centralized controllers, and show that regardless of what is communicated, state estimates or observed events, the control solution would be the same. However, their model is inherently complicated mainly due to the fact that it is a fully state-based model and besides that, it uses a tree structure to find the state estimates, which is not a convenient way to handle loops. Moreover, the "latest-safe-point" policy in communication is not robust to communication delays and it is not necessarily optimal in terms of the volume of the exchanged information, i.e. the number of bits that actually have to be communicated.

Ricker and Rudie choose the *knowledge* framework of [23] to explore the information in a DES [24]. This framework has a simple syntax to represent the decentralized information in the system as shown by the authors in reformulating the coobservability as *Kripke-coobservability* [25]. Kripke-cobservability, which is later called know-coobservability in [26], requires that at all states of a specification, for each (controllable) event either it is the case that it does not happen or it is legal otherwise or there is an agent which has control over that event and knows that it is illegal. The authors show that know-coobservability

is a necessary and sufficient condition for solving a decentralized control problem [25]. In [24] the supervisor which observes the execution of an event starts communicating its state estimates, called *possible worlds*, to the supervisor which needs this information. The authors divide the communication into two parts; "communication for control," which is done to widen the observability of the supervisors, and "communication for consistency," which is done at the states which are (from the viewpoint of the communicating supervisor) indistinguishable from the states at which communication for control is necessary. However, the communication policy is *as early as possible* in this work. Accordingly, first a *maximal-P pair* is defined as two strings which are indistinguishable to a centralized controller. Communication takes place at the first distinguishable state (to the communicating supervisor) following such a pair. It is shown that the observability of the specification from the viewpoint of the centralized controller guarantees the existence of such *communication states.* The authors suggest a greedy algorithm to find the communication rules with no proof. Again, this algorithm uses a tree-like structure whose extension to general automata with loops is not convenient.

In [27], Rudie *et al* suggest an algorithm which communicates the observed *events*, i.e. transitions, and prove that it yields a minimal communication among two supervisors in the sense that the cardinality of the set of communicated events is minimal. Their algorithm satisfies their proposed characterization of the solutions of a communication problem, which is formalized in terms of three properties of *validity*, *feasibility*, and *implementability.* Validity requires that a supervisor identify which state it is in and feasibility necessitates that the communication policy of the communicating supervisor(s) be consistent for any two lookalike strings. Implementability, however, limits the communication policy to the given state structure of the supervisors. The

algorithm starts by picking up the two smallest required sets of transitions for communication associated with the two supervisors. Next, it adds or subtracts iteratively other transitions to satisfy feasibility and ends up with a minimal set, in the sense stated above.

It is reasonably argued in [27] that the inherent difficulty of the problem of "decentralized supervisory control with communication" stems from the chain-wise dependence of control, estimation, and communication on each other. While implementation of any control is limited by how much information the controller can estimate (i.e. observe) about the plant's behavior, estimation of the plant's behavior is changed by the controller, once part of this behavior is prohibited. Needless to say, estimation also depends directly on communication, whereas the amount and the time of communication vary as the controller chooses a different policy. Therefore, in general to deal with this coupled problem, we may seek a *suboptimal* solution, i.e. one in which a fixed strategy is taken for one element, thereby reducing the complexity of the whole problem and solving for the two other unknowns subject to the predetermined third element. As an example we may assume a fixed observation (i.e. no communication) and derive solutions to the decentralized supervisory control under the (fixed) partial observation.

On the other hand, one may forget about control for the moment (for example by assuming a fixed control strategy), and focus on the observational properties. Taking this viewpoint, Tripakis introduces the notion of *joint observability* [28] for investigating the observational properties of a given global specification, regardless of control. A specification is called jointly observable with respect to the plant's language and a tuple of subalphabets, associated with a number of component plants, if for every two strings, one

legal and the other one illegal, there exists at least one supervisor, not necessarily being able to exercise control over the event in question, which can tell them apart. Joint observability is stronger than observability, incomparable to $C\&P$-coobservability, and closed under the union and intersection of languages [28]. This notion, which was later called *unbounded-memory joint observability* in [29], essentially reveals the (maximum) amount of information available to the component plants (or agents) which may be represented as (possibly infinite state) automata.

Tripakis' work has roots in *Trace Theory* [30], which provides mathematical means to characterize traces. By Trace Theory a language is called *trace-closed* if for every sequence of events within it, it includes all permutations of the events from disjoint alphabets forming the sequence. Tripakis shows that for disjoint subalphabets, each belonging to a component plant, *finite-memory* and infinite-memory joint observability are equivalent to each other if the plant language is trace-closed [29]. Moreover, he shows that if the plant language is trace-closed, a specification is jointly observable if and only if it is observable and trace-closed. Along the same viewpoint, he calls a specification *locally observable* with respect to the plant language, components' subalphabets, and a Boolean function if the Boolean function, acting as a logical operator to fuse the local observations, disambiguates any two conflicting strings, one legal and the other one illegal, in plant's behavior [31]. Local observability implies joint observability and moreover, its finite-memory version is equivalent to infinite-memory version under the conjunctive or disjunctive fusion rules. Although Tripakis' main focus is on decidability issues, which are associated with decentralized observation, his approach seems promising for formal investigation of observation, and in particular, communication. This is mainly due to the fact that it provides a characterization of the problem in

the absence of any coobservability.

Minimality of communication has been studied using abstract frameworks in some recent papers, too. In [32] "Essential transitions" are introduced as transitions which "constitute the initial required communications," subject to which an algorithm can find the minimal communication between two supervisors. The authors in [33] extend the model of [27] to any finite number of supervisors, a more general communication structure, and a more efficient algorithm. However, these improvements are gained by assuming no cycles (other than selfloops) in the supervisor's structure. It is worth mentioning that our proposed distributed framework can handle every arbitrary state structure, while addressing issues such as minimality of communication.

1.3.1 The issue of time in communication

There have been some efforts to consider the communication in a timed framework. This is mainly motivated by the fact that in real-life systems, communication might be unreliable or erroneous due to delays or losses in communication channels. Therefore, it is important to find out how late the information needed by controllers can be delivered to them. However, this issue has been investigated only in few works. Ricker and Van Schuppen proposes a model based on an *event-recording automaton* to investigate the communication delays in decentralized control [34]. They use this model to show that to diagnose failures using decentralized observers and to find out the temporal order of events, local clock values should be communicated, too. Another treatment of communication delay appears in Tripakis' work [29] in which he assumes as the unit of time, the execution of an event, and measures the delay using this unit. He also assumes a lossless network and takes the very simple communication policy of "send everything you observe." These assumptions lead

to a characterization of communication networks into an infinite hierarchy of monotonically decreasing class of problems based on the communication delay, by which the class of solvable problems enlarges as the communication delay reduces. The issue of communication in the presence of network delays also has been considered in [35], [36], and [37] for the case of centralized and decentralized supervisors. These consider the delay which exists in receiving sensor observations or applying actuators' commands, and come up with the notion of "delay observability." Since the issue of time is not explicitly studied in this work, it is not discussed here anymore. However, it is worth mentioning that the logical notion of time, i.e. the time interval between the execution of two events, which is implicitly used in this work, can address issues related to the order of communication and logical delays which affect that.

1.4 Inferencing supervisors

As quoted from [26], "knowledge is a terrible thing to waste." Accordingly, an important question is how much a supervisor can use the information gathered from its direct observation or the part received from other supervisors to make a control decision. The answer is determined by the limit of the supervisor's *inference* which can be made on the grounds of the available information. Inferencing uses some information from other supervisors in a distributed system, hence entailing communication among supervisors. It may also affect the amount and the content of the communication because making an inference-based decision might eliminate the naive exchange of information among the controllers. These observations have motivated some researchers to employ inference in decision making. The first work in this area perhaps goes back to [38] in which a supervisor makes its decision based on other supervisors'

previous decisions.

The idea of using more sophisticated and inference-oriented control fusion rules was proposed in [39]. Yoo and Lafortune then suggest a more general architecture for decision making for control of DES in [15]. Following this architecture the authors consider a *conditional architecture* which allows "disable," "enable," and "conditional" control actions [40]. Their conditional decision making consists of actions of the form of "enable (globally) if nobody disables (locally)" or "disable (globally) if nobody enables (locally)." This architecture partitions the controllable event set into two "disable by default" and "enable by default" event sets *a priori*, where the "default" control action is applied when no decision is made by a controller for an event. The authors present a necessary and sufficient condition for the existence of decentralized supervisors when the specification is controllable and *conditionally coobservable*. The latter condition requires that the specification be conditionally $C\&P$-coobservable with respect to a subset of controllable events and conditionally $D\&A$-coobservable with respect to the other disjoint subset of controllable events so that these subsets form a partition of the controllable event set. As an example, conditional $C\&P$-coobservability implies that if a controllable event is an illegal continuation, there exists a local controller that can infer that the event can be disabled conditionally when it cannot be disabled unconditionally. The authors show that the class of problems solvable in the conditional structure is wider than those solvable using the conventional "unconditional" coobservability. They also present a method for the realization of the conditional decision making based on constructing deterministic observers which track the violations of coobservability [41]. The authors then employ the suggested architecture for fault diagnosis of DES [42]. Implied by this conditional structure is the need for a *coordinator*, who is a global entity

making the global decision about the fusion rule. This makes this architecture more centralized than decentralized.

Ricker and Rudie base their work in "inferencing supervisors" on the knowledge framework of [23]. They define *distributed observability* in [25], which necessitates that a group of controllers, including the ones having control over a particular event, after combining all the knowledge of the group, i.e. intersecting all possible worlds, give a clear idea of disabling or enabling an event to the controller of that event. They show that if a specification is observable but not know-observable, it has necessarily distributed observability [43]. Along the same line in [26] and [44], they investigate the inferencing power of supervisors. Motivated by the conditional structure of [40], the authors reformulate the same results in the knowledge framework. They introduce the notion of *inference observability*, which serves as a necessary and sufficient condition for the existence of a *joint knowledge-based protocol*[4]. This is a tuple of knowledge-based protocols, each mapping a supervisor's knowledge of an event, at its local state, to a "disable" or "enable" control decision. In simple words a specification is *inference observable* if at its every state and for every controllable event either the event cannot be generated by plant, or it is legal otherwise, or there exists a supervisor which can exercise control over that event which knows that the event is illegal or if it observes its happening it is certain that there is another such supervisor who is aware of the legality of that event[5]. One advantage of the knowledge framework is that its syntax can easily represent the complicated cases of information sharing. Thus it is relatively simple to express different levels of the availability of information to the controllers. However, as commented by the authors of the paper, employing

[4]This notion of protocol is not necessarily equal to a communication protocol.

[5]An advantage of the knowledge framework is its simple syntax for stating such long English statements in an efficient and clear manner. Unfortunately, such a syntax cannot be used in this introductory chapter.

the conditional architecture invokes the use of a coordinator, which makes the architecture more centralized, and an *a priori* partitioning of the controllable events, which is based on the decision making applied to them by default.

Qiu *et al* [45] use Prioritized Synchronous Composition (PSC) and Prioritized Composition with exclusion (PCX) to extend the conditional architecture of [40] by partitioning the controllable event set into 4 (respectively 8) sets which account for the different priorities and exclusivities of the control and the default actions. They show that the classes of achievable languages under PSC and PCX architectures are the same as $C\&P\vee D\&A$. However, in [46] Kumar and Takai argue that all such conditional architectures and their inspired inferencing schemes need an *a priori* partitioning of the controllable events into the so called "permissive" and "anti-permissive" sets. This observation, and the fact that such schemes may not easily yield a classification of supervisors' ambiguities, lead the authors to suggest a framework for decentralized decision making which does not require an *a priori* partitioning of the controllable events, while supporting the inferencing at different levels of ambiguity [46]. A control decision with "level-zero ambiguity" is taken for an event when a supervisor, based on its own observation alone, clearly knows that the one-step continuation of all the observationally equivalent strings by that event, which are feasible in plant's behavior, are exclusively either legal or illegal. However, if this local supervisor thinks that some of the strings might lead to illegal behavior and therefore tends to disable the event, yet it knows that there is another local supervisor which can issue an "enable" decision with level-zero ambiguity, the former supervisor issues a "disable" decision with "level-one ambiguity." This process may be applied iteratively to any level of ambiguity and the "winning" global decision would be the one with the minimum level of ambiguity, thereby it eliminates the need to compute the conjunction or disjunction of the

local decisions at global level. The authors claim that this makes the approach more decentralized comparing to those in [40] and [26]. Correspondingly they call a specification language *N-inference-observable* if the ambiguity level of a "winning" control decision is at most N and show that this condition together with controllability and $L_m(\mathbf{G})$-closure are necessary and sufficient for the existence of a set of non-blocking, so called, N-inferring decentralized supervisor. They further show that 0-inference-observability and 1-inference-observability are equivalent to $C\&P\vee D\&A$-coobservability and conditional $C\&P\vee D\&A$-coobservability of [40], respectively.

1.5 The perspective of this research on SCDS

As stated in [27], the main difficulty of the SCDS problem stems from the dependency of control, observation, and communication on each other. On top of this point, the fact that there does not exist necessarily a (unique) optimal solution and the associated complexity of finding any maximal solution,in the sense of the largest behavior, are good motivations for employing more specific methods and formalisms which take full advantage of the structure of every problem.

The language-based formalism, and its dual entity, automata-theory, study problems at an abstract level. These approaches are very useful at formalizing the general properties common to a large class of problems and their solutions. This has been the mainstream in research so far and has led to interesting results. However, when it comes to more specific properties of a system, which is captured in its dynamic structure, one needs to choose a more structure-preserving and structure-reflecting formalism.

In the author's viewpoint, studying the communication, as a part of the

communicating decentralized supervisor design, would be more fruitful if the algebraic structure of the supervisors are explored and utilized in more detail. To this end, each state of the corresponding centralized supervisor is encoded in a distributed way and, in a "divide and conquer" approach, the supervisor's observation and control tasks are represented as dynamic and algebraic equations. Thereby, the dynamics-related information of the transition graph of the centralized supervisor would be amenable to characterizations based on concrete algebraic structures, such as groups and fields. This makes the study of complex systems, and specifically design and computation of the communication, systematic and simplified. Furthermore, this approach leads to a finer partitioning of the communication and establishes insightful correspondences to the centralized supervisor's behavioral properties, as will be explained in later chapters. Moreover, since in practice a supervisor is implemented through encoding its system-related information, employing a (distributed) encoding scheme as an integral part of the (decentralized) supervisor design, can address implementation issues, too. As an example, it is important to reduce the communication content since it might be exchanged millions of times a day as part of a communication protocol (see Chapter 6). Since in practice the content is measured in bits rather than "events" or "state estimates," this issue may be addressed properly in a framework in which design is performed with an insight into quantitative measures for implementation.

Inspired by such considerations, the author first proposed the distributed Extended Finite-State Machine (EFSM) framework to study the communication among supervisors [47]. An EFSM models a closed-loop system by employing a state labeling map and computing guards and actions, defined on Boolean variables, as means to observe and control the plant's behavior,

respectively [48], [49]. In [50] Agent-wise Labeling Maps (ALMs) were introduced as efficiently-computable replacements for natural projections, based on which the control-related information of a centralized supervisor is represented by a network of *distributed* EFSMs. It was then observed that introducing Boolean variables at an early stage hinders further development of a theory for communicating supervisors by making the notations and computations unnecessarily cumbersome. Furthermore, working with an integer variable, as a meaningful representation of a state, inherits both the qualitative and the quantitative viewpoints to analysis and synthesis. This led the authors to propose a flexibly abstract mathematical framework for decentralized control synthesis, which, while taking an abstract qualitative-like vantage point to system representation, can readily lend itself to the concrete implementation of decentralized supervisors by (Boolean) variables at a later stage. This research presents the above two approaches in a chronological order.

The second approach assumes that a centralized supervisor is already designed using SCT and introduces an ALM to label its states with disjoint sets of integer vectors whose component i encode the ith decentralized supervisor's observation of the states. For each event, updating functions are defined to specify how vectors should be updated by its occurrence, and guard functions are defined to identify the vectors at which it should be enabled. A DES, which is thus equipped with guard and updating maps, is referred to as a Supervised DES (SDES). An SDES inherits the centralized supervisor's properties such as optimality and nonblocking. The SDES framework, which encompasses EFSM framework as its special case, serves to provide a *polynomial dynamical system* (PDS) representation of the centralized supervisor (and closed-loop system), upon formulating its guard and updating functions as polynomial equations

over finite fields. Therefore, the information structure of the centralized supervisor, which consists of the observation- and control-related information associated with its state structure, is captured in a compact and standard manner, and this in turn paves the way for further analysis and synthesis purposes. The resulting state space formulation is very much similar to what control theorists have at hand for studying physical systems and complements the behavioral viewpoint and analysis tools for DESs.

For a distributed SDES (DSDES), communication naturally appears to inform a supervisor of the labels assigned by other supervisors, which it uses to reevaluate its guard and updating functions. Within DSDES framework, SCDS is reduced to the synthesis of communication among supervisors, which is realized by designing "communication events" to help the decentralized supervisors, through sharing (part of) their private information, enforce the given specification. Once a DSDES is represented as a PDS, communication is characterized and solved based on the interconnections between polynomial equations defined over algebraic structures and the properties of such structures, leading to a general, systematic, and computationally efficient synthesis procedure for communicating decentralized supervisors. The proposed approach can accommodate the formalization of the communication of states and events, while providing an implementation-oriented encoding for the final field applications.

In the following, EFSM and SDES frameworks are explained in more detail.

1.6 EFSM and SDES frameworks

In the literature, the term "extended state machine" usually refers to a state machine whose state structure is augmented with variables and functions of states and variables. Different types of extended machines have been proposed in areas such as verification, Application Specific Integrated Circuit (ASIC) design and real-time implementation of DES [51], [52], [53]. In [54] the extended state machines use "data" and "activity" variables to model control and numerical information, respectively, and with their enhanced expressive power, they can address problems such as state explosion, fairness, and timing of the events in real-time DES. In the area of supervisory control, by equipping finite-state machines (FSMs) with resource and time constraints and employing inequalities and integer variables as "guard formulas," Chen and Lin increase the expressive power of DES models and efficiently represent systems that cannot normally be represented by regular FSMs without arbitrarily large state spaces [55].

The formalism of Extended Finite State Machines (EFSMs), whose distributed version is used in this work, was originally developed in [49] for implementing the solution of SCT to centralized control problems. In this formalism, controllers' information is encoded as (Binary) variables and therefore bits of control information necessary in the process of decision-making, rather than events or state estimates, are communicated from one supervisor to another. While it uses supervisory control theory in [6] to design a supervisor, it extends a plant's automaton with Boolean variables, guard formulas, and updating functions to implement the centralized supervisor. The resulting EFSM, which has the same expressive power a regular FSM, implements supervisory control by employing Boolean variables to encode the supervisors states, a set

of Boolean functions to observe events, and Boolean formulas to control transitions [49], [48]. As it was originally motivated, the resulting EFSM offers an economical representation of the closed-loop system, thus bridging the gap between supervisory control theory and the traditional designs of DESs, in which the closed-loop system is designed in an ad hoc manner.

Although the EFSM formalism proposed in [49] has an equal expressive power to an FSM, this approach makes clearer the relevance of supervisory control theory to many computer-control problems and specifically problems in software system synthesis. Moreover, it is more suitable to represent input-output behavior of the DES thanks to encoding a controller "command" by a guard formula and a plant "response" by a set of updating functions. Thus, for example, EFSM models can be readily translated to PLC implementations where Boolean variables are used to encode supervisor states and events. It may also be regarded as a *symbolic* representation of the set of supervisor states in which an event is enabled (see the next section).

At an initial step of my research, I extended EFSM framework to the distributed case by equipping the original framework with the notion of Agent-wise Labeling Maps (ALMs) (see Chapter 3). An ALM essentially provides a distributed representation, i.e. a representation from the viewpoint of more than one agent of the state structure of a centralized supervisor, as imposed by the observational windows of the associated component plants. This representation is obtained without going through computationally expensive and structurally destructive natural projections, and models the component plants' observational windows, in an efficient way such that the state structure of the centralized supervisor is respected. It was shown that for every centralized

supervisor an ALM exists. Next, using the ALM-assigned labels to the supervisor's states, a set of private Boolean variables is considered for each "component EFSM," associated with each component plant, to represent the private information at local sites. The decision as to whether to enable or disable a local event may in general depends on the values of supervisor's own private variables, and the local copies of variables owned by other supervisors. These copies are updated by communication among local supervisors. The distributed EFSMs thus developed enabled the author to model and synthesize ABP, a prototype protocol, in a completely automatic way for the first time and gain insight into partitioning of the class of solutions to the communication problem (see Chapter 3).

The EFSM formalism, on top of the language and automata theories, based on which supervisory control theory is formalized, may be used to model distributed DESs and their real-life applications and has the following advantages among others.

1. It can explore the structure of distributed dynamic systems using its symbolic and compact representation. This structure reveals the interdependencies of the components and reveals the information available to each controller, while reflecting the state structure of the centralized supervisor.

2. The resulting form of the dynamics of the centralized controller would be more similar to the current state-space representation in systems and control theory. This facilitates the establishment of familiar notions and tools of control theory in this context.

3. As it is usual in formal verification of hardware and software systems (see for instance [4]), symbolic representation is a tool to tackle the

computational complexity and state-explosion.

4. The bit-wise approach together with the functional syntax used by an EFSM may find more satisfying answers to some key questions in decentralized control of DES such as the *minimality* of communication.

Comparing the EFSM-based approach to the *estimator structure* of [1] and *possible worlds* of [24], which are two pioneering works on SCDS, the following observations can be made.

- While an ALM can be found for any deterministic automaton of a centralized supervisor, the other two approaches have been applied to reachability trees only, and their applicability to general automata containing loops is not claimed nor does it seem obvious.

- An ALM adopts an agent-wise viewpoint in labeling the states of a centralized supervisor, while the other two approaches rely on a global labeling for the states and then gathering the lookalike state labels for each agent as a set of state estimates [1] or possible worlds [24]. Since in decentralized control, decentralized supervisors view the plant's behavior subject to their partial observations, the ALM labeling provides a natural formulation for the distribution of information within the network. Moreover, the ALM approach views the labels as an integral part of the implementation of the centralized supervisor's commands, while in the other two approaches labeling is an auxiliary tool and the viewpoint is quite abstract.

- The final rules for communication in the other two approaches are always translated in terms of communicating the state estimates or possible worlds, while in EFSM framework, which is equipped with ALMs, everything is expressed with respect to bits of information used by each local

supervisor to encode the states of a centralized supervisor. Observe that the latter serves to define a practical measure, especially when issues such as minimality of communication are studied.

- Another advantage of the EFSM formalism is its compact representation of the supervisors' commands and observations using Boolean formulas and functions, while the other two approaches make use of the supervisors' automata.

- The works in [1] and [24] adopt "the latest safe point" and "as early as possible" communication policies, respectively, to deal with the issue of "when" to communicate. Although this issue is not explicitly addressed in our work, where the focus is on the logical aspects of communication design, it is implicit that communication takes place whenever necessary, in other words, within a time interval when guards or updating functions need to be reevaluated.

Despite the above merits, introduction of Boolean variables at an early stage of modeling hinders further development of the theory and makes the proofs unnecessarily cumbersome. Moreover, a Boolean variable seldom bears a (state-related) meaning singly but an integer variable can represent a state. Furthermore, analysis and synthesis of a symbolically represented system is in general more straightforward in an integer space thanks to the easier representation. Such considerations led the author to employ integer variables for extending the automaton of a centralized supervisor and introduce SDES framework. In this framework, guard and updating functions, defined on integers, are the counterparts of guards and actions of EFSM framework. These functions can next be put into polynomial forms over a finite field, leading to a number of algebraic and dynamic equations, totally called a polynomial

dynamical system. Being defined on Boolean field, EFSM framework may be regarded as a special case of SDES framework (see Chapter 4). As a result, the advantages listed above for EFSM framework over the frameworks based on "possible worlds" and "state estimates" are shared by the DSDES framework, too. Therefore, a blend of DSDES and EFSM frameworks enjoys the complementary merits of both.

1.7 Symbolic systems and PDSs

As mentioned in the previous two sections, SDES and EFSM frameworks provide symbolic representations of a centralized supervisor in the form of a polynomial dynamical system (PDS). For a distributed SDES or EFSM, this PDS provides a distributed symbolic form of the information structure of the centralized supervisor.

Symbolic systems and PDSs have been of interest since the early days of SCT. In [56] PDSs are introduced as formulations for control problems of DESs whose solutions can be computed using tools in computational algebra. In [57] control theories for a large class of systems are formulated based on PDSs. The author of [58] develops a modeling framework to transform a DES's dynamical equations into relations over finite domains and represent them using binary decision diagrams (BDDs), integer decision diagrams (IDDs), and polynomials. BDDs were also proposed in [59] to reduce the complexity of the SCT designs and are employed by a recently developed software tool for handling industrial-size problems in [60]. Zhang and Wonham introduce a software tool which gains its efficiency through embodiment of the modular composition of the plant and the specification in Integer Decision Diagrams (IDDs) [61]. By employing state-tree structures to reduce complexity, a symbolic algorithm

has been proposed in [62] to synthesize nonblocking supervisors for large plants yielding physically meaningful representations of a controller.

In all of the aforementioned works, symbolic representations and PDSs have been used to model, verify, and design *centralized* controllers or to reduce the computational complexity of such designs. In this research, however, PDSs are used to model, synthesize, and analyze communicating *decentralized* supervisors and specifically to design the communication. It is assumed that a centralized supervisor is already designed for a (distributed) DES and its global specification, i.e. no effort is put to design nor to simplify this supervisor nor to compute it more efficiently through employing symbolic representations. In other words, the proposed synthesis procedure of this thesis starts right after such a centralized supervisor is provided by SCT. More specifically, PDSs are derived using the DSDES model of a centralized supervisor, which inherits its SCT-based properties such as optimality and nonblockingness. The PDS represents the observation- and control-related information structure of the centralized supervisor in a distributed, i.e. agent-wise, manner. These two kinds of information are formulated, respectively, as polynomial dynamical and algebraic equations over integer variables whose values are taken from a finite field. The properties of these polynomials and the underlying filed is then used in a systematic and computationally efficient way to study the synthesis problem and formulate and address relevant issues such as content and minimality of the communication.

In summary, the PDS representation is used in this research as a compact and structured way of representing a centralized supervisor. This representation is amenable to characterizations based on more elaborate mathematical structures such as groups and fields, compared to set theory which is the prevailing tools of behavioral study of DESs. Notice that while improving the

computational complexity of the synthesis, through using structured and well-developed tools of computational algebra, could be one of the merits of the proposed approach of this research, it is not its main objective and therefore, details of solution techniques and optimization of the solutions are not within the scope of this research. It is worth emphasizing that whereas this thesis does not employ PDS representations to compute the centralized supervisor, it seems more promising if both symbolic tools, one for the centralized design and the other used here for the decentralized design can be integrated into a complete design process to improve the complexity and optimize the whole modeling and synthesis processes in this way. Such a procedure deserves a separate work and, in the author's belief, is one of the natural continuations of this research.

1.8 Other formalisms to study decentralized control

The previously-mentioned works, including SDES and EFSM frameworks, all are originally based on automata theory and languages. However, there have been extensive research on distributed systems and their verification in areas such as computer science, software and hardware verification, and economy since long time ago. Some of these works have adopted other formalisms to study their problems of interest.

Game Theory is among the most related formalisms. In [63] Overkamp and Van Schuppen characterize the *maximal* solutions[6] of decentralized supervisory control as a variation of Nash equilibria of strategic games. The authors also suggest an algorithm which can find some of these solutions[7].

[6]Maximal solutions are solutions which yield the maximal closed-loop language.

[7]The author has extended this work in [64] to the case of "disjunctive" fusion rules.

Apart from the insight gained from this characterization, the complexity of computing such solutions can be measured by employing the available results in *algorithmic game theory.*

Another interesting work is [65] in which the minimal amount of information needed to be communicated by the agents in a computer network is called "communication complexity." The authors link communication complexity to Shannon's *information theory* and employ different models of communication such as "Yao's two-party model," "variable partition models," and "communication complexity of relations" to study the problem. As mentioned by the authors, while the subject of "information theory" is a certain communication which needs to take place, the subject of "communication complexity" is a problem to be solved, say addition of two numbers by a computer [65]. The ideas formalized in this work might be employed in the RW framework and specifically in SDES and EFSM formalisms in a later work.

Another viewpoint is to look at the network topologies which form the interconnection of different components. A main objective in this area is to have inexpensive network structures which are yet efficient. These requirements might be formalized in *graph theory* in terms of "minimum time broadcast graphs" (see for example [66]). The ideas here might be insightful specifically in investigating communication over unreliable channels.

1.9 Problem statement and basic assumptions

Given a (distributed) DES and its specification(s), this thesis studies the problem of "synthesis of communicating decentralized supervisors." This problem,

However, this work is not directly related to the mainstream of this thesis and is not included in it.

through introduction and development of distributed SDES and EFSM frameworks, is reduced to the synthesis of (rules of) communication among supervisors. The two distributed frameworks are both built upon a central tool, called an agent-wise labeling map (ALM), which reveals the information structure of an already-designed centralized supervisor. This has been synthesized using SCT for the (distributed) DES and its specification(s), in a distributed manner, i.e. as reflected by the observable subalphabet associated with each component DES. Finally, synthesis of (the rules of) communication is in turn formulated and solved using polynomial system representations (PDSs) of the centralized supervisor. Each such PDS consists of a number of dynamical equations, associated with the centralized supervisor's observation, and a number of algebraic equations, associated with the centralized supervisor's control map. The associated polynomials are defined on (integer or Boolean) variables whose values are taken from an underlying finite field. These variables reflect the local information owned by each component DES.

As stated above, a basic assumption in the proposed synthesis procedure is the availability of a centralized supervisor, designed using SCT for the (distributed) DES in question and its specification(s). In the first place, this assumption requires that the DES and its specification(s) should be expressible using regular languages and finite automata. On the other hand, it is assumed that such a supervisor can be computed using SCT, which in turn implies that it should be within the computational power of SCT associated software tools. More specifically, if such a supervisor should be designed for a distributed plant, the complexity of synthesis increases. However, this issue, though being an important feasibility issue, is inevitable when dealing with complex multi-component systems in the sense that every procedure for synthesis of decentralized supervisors will face it at some level of design.

An implication of using the centralized supervisor as the starting point of SCDS, is that the properties of the supervisor, including its state structure, optimality, and nonblockingness are all inherited by its SDES model and PDS representation. On the positive side, this means that the given state structure is respected by the synthesis procedure, a fact which is neglected by all procedures which employ natural projections[8]. However, one should be warned that the size of the state structure and some of its selfloop transitions[9] may affect the size of the ALM and, in turn, the complexity of the final PDS representation. Again, this "burden of complexity," though imposing computational limitations, is due to the very nature of complex distributed plants and cannot be fully remedied in other synthesis procedures, either.

The main subject of this research is the synthesis of "communicating" decentralized supervisors. Since communication is not required in the presence of coobservable specifications, it is assumed that the given specifications are not coobservable in any sense, i.e. they are neither $C\&P$, nor $D\&A$, nor any other types. This assumption holds for many practical problems, as coobservability is a strong property for many networks and cannot be fulfilled.

In practical communication channels, data is received with delay or lost (i.e. received with infinite delay). In fact, one of the main goals of communication protocols is to guarantee the correct exchange of information within a network with such channels. A first assumption in this regard is that the communication channels for control signals and those for data are not assumed the same. This assumption, which could be justified in practice due to the importance of control signals, lets one limit the model of the communication

[8]A state, when designated for any reason, bears its own significance and should not be merged into other states. Natural projection merges some states which are related by unobservable transitions.

[9]Some selfloop transitions should be unfolded in the proposed procedure (see Remark 3.1).

channels to the part which is related to SCDS problem. A second assumption is that, whereas the issue of "delay" is an important and interesting subject, it is not studied in this thesis, except in Chapter 4, where a sufficient condition is provided. Accordingly, it is assumed in general that channels are delay-free. This assumption could be fulfilled in the networks where communication delay is negligible, though it is quite uncommon. However, the main reason for such an assumption is that, as shown in Chapter 5, the rules of communication can in general be divided into two classes, leading to two levels of communication design. The first set consists of the rules which "logically" prescribe the exchange of information to help the network satisfy its specifications (even in the presence of delay-free channels). In other words, only informational dependencies are important and physical (environmental) constraints are neglected. Such dependencies hold regardless of channels' characteristics. The second group of the rules are those which make the implementation of the first group feasible in the presence of delays or in the absence of some communication channels. Being formalized in Chapter 5, this classification gives rise to the notion of "information policy" and "routing policy," respectively, with the second one left for future work.

The above were the most fundamental assumptions of this work. In the following chapters, wherever necessary, the above assumptions are stated explicitly.

1.10 Research achievements

The advantages of using SDES and EFSM frameworks over some previously developed frameworks for SCDS were listed in Section 1.6. The major theoretical achievements of this research are as follows.

1. *Introduction of a general framework for modeling, analysis, and synthesis of (communicating) decentralized supervisors for DESs*: SDES framework, built on ALMs, provide a compact, symbolic, structured, and computationally efficient way for modeling a centralized supervisor in a distributed way. This framework is general in the sense that it can be applied to all state representations of an already-designed centralized supervisor, thanks to the fact that an ALM can always be found for a given supervisor in an efficient way.

2. *Introduction and development of a general systematic procedure for solving SCDS*: The procedure starts by labeling the states of a centralized supervisor, already designed using TCT, in a distributed way using an ALM. This is followed by computing updating and guard functions, which capture the observation- and control-related information of the supervisor, respectively. Next, these functions are represented by polynomial functions, defined over a finite field. The (rules of) communication among supervisors is then synthesized using the properties of the polynomials and the underlying field.

3. *Formal modeling and synthesis of a number of information policies*: A communication policy, which is formally defined in Chapter 5, can be simply defined as the rules for exchanging decentralized supervisors' private information among each other. In this work, a communication policy is divided into an information policy and a routing policy. As solutions to SCDS, a number of information policies are derived and formally proved to insure the satisfaction of the given specification(s), under the assumption of delay-free communication channels. Some of these policies prescribe the exchange of supervisors' information on the "observed states" and some prescribe the exchange of information on the "observed

events." In other word, both types of communication contents, i.e. states and events, can be accommodated in the proposed synthesis procedure.

4. *Providing an implementation-oriented solution to SCDS*: As will be formally shown in Chapter 5, the two types of communication contents, i.e. states and events, which are prescribed by the synthesis procedure within SDES framework, are finally encoded using Boolean variables, with the help of EFSM framework and event-encoding schemes, respectively. Such a bit-wise implementation of the communication rules not only suits minimality analysis of the communication but also is ready to use by field engineers, who encode every pieces of information using Booelan variables to run on digital processors or store in digital memories. It is an advantage of having an encoding scheme as an integral part of the proposed modeling frameworks to envisage implementation issues in theoretical studies, compared to non-optimally adding such a scheme at the last stages of implementation.

5. *Finding a finer partitioning of the class of solutions to SCDS problem*: Within SDES framework the structural information of a centralized supervisor is divided into observation- and control-related pieces, as captured by updating and guard functions, respectively. This division is reflected by the PDS representation of the supervisor, where polynomials may depend or not on other supervisor's private variables. Dependence on others' variables, as shown in Chapter 5, may call for a communication from the owner of the variable to the supervisor in question. This fact, on top of the properties of the polynomials and their underlying field, helps one distinguish different classes of solutions of SCDS depending on the variable dependency and the graph of the function. Such a classification, which has an impact on the recovery of the closed-loop system

from communication faults, has not been reported in the literature.

The next section discusses the application-oriented contribution of the thesis.

1.11 Synthesis of communication protocols

On the application side, this research was motivated by "automatic synthesis of communication protocols," examples of which are Alternating Bit Protocol (ABP) and Transmission Control Protocol-Internet Protocol (TCP-IP). In a communication network, which is a common example of a distributed DES, processes exchange among themselves data messages under an ordering specified by a set of rules, known as a communication protocol [67]. Although extensive research has been done in this area from the viewpoints of communication, software engineering, and computer science, there is no rigorous and general control theory behind the synthesis. In fact the current trend is to first formalize the specification, then "guess" a good solution by experts and finally verify the closed-loop behavior of the system against the given specifications. Needless to say, automation results in more reliability and safety, as well as less time and expenses of maintenance and design. Moreover, dependence on experts leads to non-optimal solutions and limits the flexibility and easy extensions needed for large-scale future problems which are not easy to handle manually anymore.

While the use of formal methods has significantly contributed to specifying the desired behavior of systems and validation techniques for communication protocols [68], the control community, in turn, has considered the automation of the synthesis problem such that given the model of the network and the desired behaviors, the protocol is generated mechanically (i.e.

automatically and without human interference). In [13] the example of ABP was first introduced as a solution to a decentralized supervisory control problem. However, since both plant components (i.e. sender and receiver) and the specification models "spelled out" the solution [69], the synthesis was ad hoc. In a later formulation in [69], the specification does not contain the solution and requires only a linear ordering among some events. It is shown that inclusion of ABP in the sender model makes the specification coobservable with respect to the plant and thus there exists a set of non-communicating decentralized supervisors which yield ABP. Conversely, in the absence of this inclusion coobservability does not hold and such a set does not exist.

When coobservability fails it may still be possible to design decentralized supervisors by allowing communication among them. In fact, the findings of this research support the idea that such problems yield protocols which require communication of some control- or observation-related information among the supervisors. In the proposed formulation of the problem in Chapter 3, assuming ideal communication channels, the protocol design for a special class of non-coobservable specifications, including ABP, is reduced to SCDS.

The proposed synthesis approach of this thesis requires neither the plant components nor the specification to "spell out" the protocol (i.e. part or all of the protocol need not be designed and included in the transition structures of plant components or specification beforehand). For ABP, which is a practical benchmark and an illustrative example, it is shown that the protocol arises naturally as a solution to the corresponding control problem with no *a priori* inclusion of the solution in the plant model or the specification language.

1.11.1 The application-oriented contribution of the thesis

In the following, first I briefly comment on two existing approaches to protocol synthesis by presenting their big pictures. Next, the proposed methodology of this thesis and its features will be explained. For the purpose of a simple illustration of the ideas, we limit ourselves to a network of 3 supervisors. We assume a "state representation of the specification, **E**," and a "state model of the communication system, **G**," are the two available pieces of information that all three approaches start from. Here, by *state representation* we mean a representation using a finite automaton. Two points are in order. First, modeling capability is limited to the expressive power of finite automata, though the limitation of using "state representation" is intrinsic to the proposed approach and its possible extensions to more expressive computational models. Second, it is possible that either plant or specification or both consist of more than one automaton. In that case, if a modular design is possible the synthesis can be broken into separate simpler designs, one for each plant and its own specification. However, if this leads to "blocking," synchronous product of plants with each other, and specifications with each other, would reduce them to a monolithic plant and specification.

Traditional Approach: Figure 1.1 illustrates the traditional approach. The synthesis mainly relies on a good "guess," say using human expertise, of the communication rules in the form of events and transitions considered in the system model. Such rules are then embedded into the system model, which is then verified for correctness. Passing this stage, the rules are output as the desired "protocol." Despite the efforts that have been made to formalize the whole process, the design is neither automated, nor systematic, nor correct by construction, because it lacks a "control" perspective. Moreover, changes

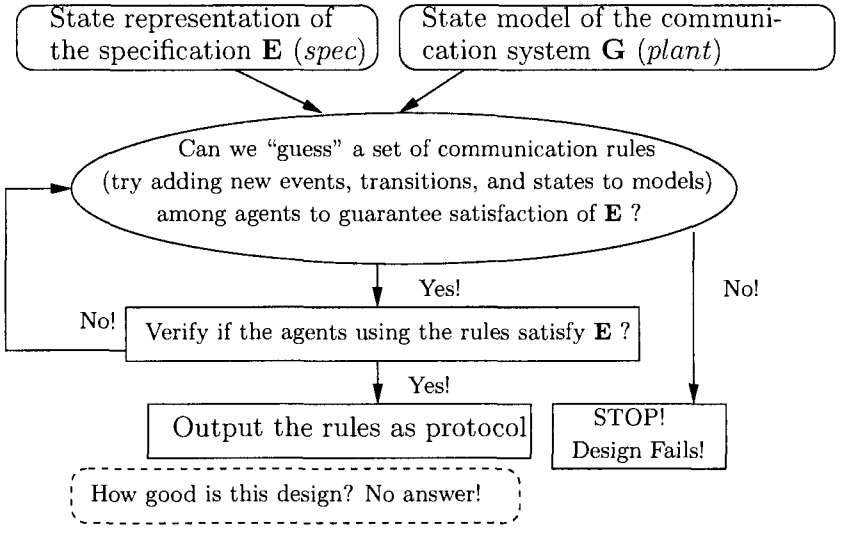


Figure 1.1: Traditional approach to protocol synthesis.

to the existing protocol cannot easily be made in general, and there is no guarantee on how this design is optimal in the sense that it minimally restricts the original behavior of the communication system. Needless to say, such a design is largely prone to human errors.

Coobservability-based Approach: Figure 1.2 depicts the second approach, which was suggested in [69]. Here the design is based on supervisory control theory, which makes it systematic and automated (up to the given plant and specification), and optimal in the sense that it restricts the plant's behavior minimally to the desired specification. However, if the specification fails to be controllable, a polynomially verifiable property, only its "supremally controllable" part, denoted by $SupC(spec)$ may replace **E**. The next important property is "observability" which can be verified efficiently, too. Unfortunately, in the absence of this property what can be guaranteed to achieve is the "infimally observable" part of the specification, which may not be satisfactory anymore (see [12]), in which case the model of the problem should be modified or there would be no solution, as the infimally observable specification does not

meet any desired objectives. It is so far clear that SCT can help the designer detect the nonexistence of a protocol (and provide solvability suggestions) at an early stage in case controllability and/or observability fail, an advantage of using a systematic design approach.

The specification being controllable and observable, the main part of the algorithm starts by performing an efficient verification for coobservability. This property ensures the existence of decentralized supervisors, one for each agent, such that under the supervision of all, the specification is satisfied. More precisely, decentralized supervisors can satisfy the specification with no communication amongst them, hence no protocol is required. For this reason, this approach suits the verification purposes. The synthesis, however, starts from the point that coobservability fails because of some ambiguity in distinguishing legal-illegal strings. In that case, some "creative work" is used to remodel the system by adding new events and transitions to resolve the problem. Once coobservability is achieved, the added parts are formulated as communication events, which are governed by some rules that are called protocol.

In short, this approach, though truly employing SCT as its synthesis tool, is based on human remodeling of the plant and specification to arrive at coobservability. This results in an inefficient and nonautomatic synthesis which may become very complicated in practice. However, it would certainly serve to verify the correctness of a designed protocol, through not to study its optimality.

Proposed Approach: In the author's viewpoint, implementing the observation and control functions of a centralized supervisor in a decentralized manner is a more natural approach to "decentralized controller synthesis" than verifying coobservability, which implies decentralization with no communication at all, and holds only for a small group of specifications. Towards this end, similar

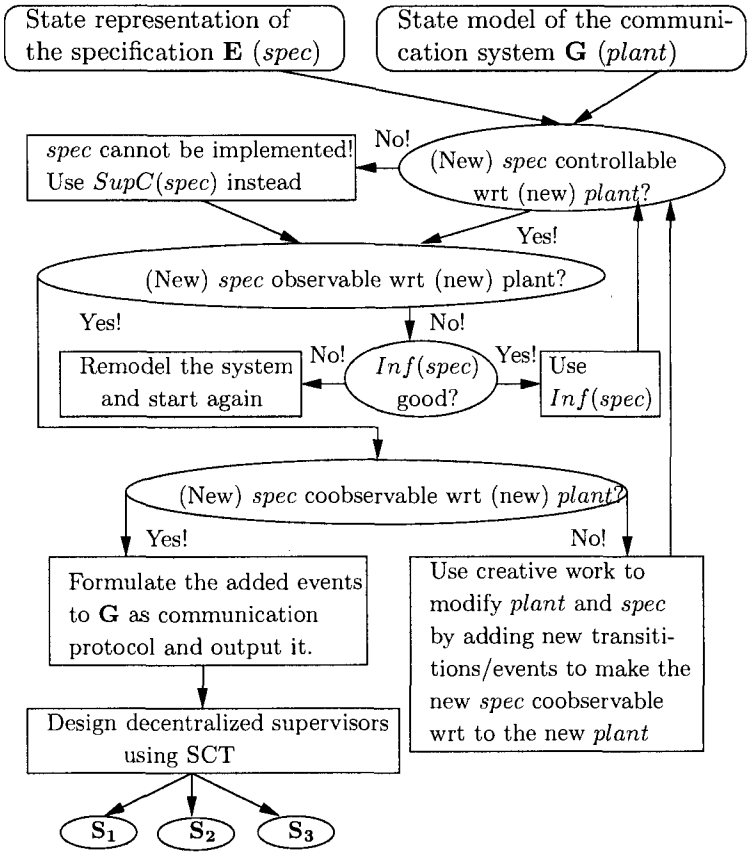


Figure 1.2: A first attempt at protocol synthesis using supervisory control theory based on coobservability.

to the previous case, the proposed approach verifies the two basic properties of controllability and observability to make sure that an (optimal) centralized supervisor exists, and if so, computes it. So far the procedure is the same as that of the previous case. Next, this supervisor is implemented in a decentralized manner in DSDES framework through which updating and guard functions are computed, recast as polynomial equations, analyzed to derive a communication policy, and then implemented using binary variables in EFSM framework, to communicate observed states, or using event-encoding Boolean variables, to communicate observed events. The last step is to derive the rules of Boolean variable exchange and output them as protocol.

The proposed approach is automatic and there is no "creative work" involved in the design process, except for simplifying the PDS representation (see Chapter 5). Computationally, checking controllability and observability can be done efficiently. Centralized supervisor can be synthesized in polynomial time in the number of states of plant and specification. An ALM, the DSDES model of the supervisor and PDS representations are all computed in polynomial time.

1.12 Outline of the thesis

The thesis is organized as follows. Chapter 2 provides the necessary mathematical background for this thesis including elements of decentralized supervisory control theory and EFSM formalism. Chapter 3 presents the author's first attempt in solving SCDS by extending EFSM framework to the distributed case and providing the solution to ABP. This chapter has been published as [50] and is repeated here as it is. As a result some of its notations are a bit different from the notations used in other chapters. Chapter 4 introduces the SDES

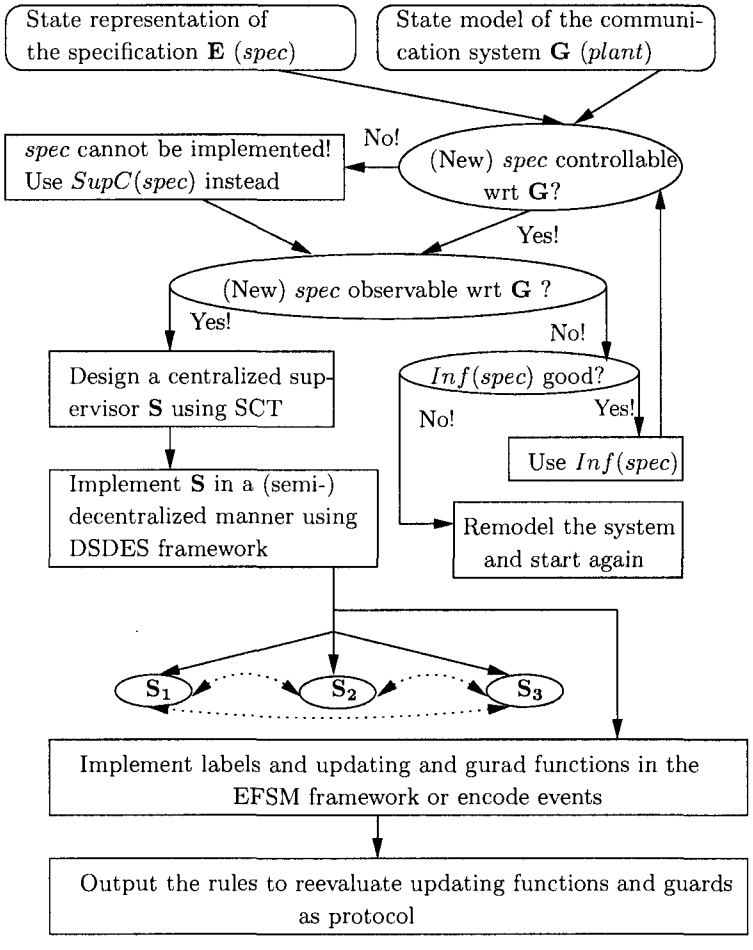


Figure 1.3: The proposed approach based on supervisory control theory and DSDES framework. Dotted lines indicate a possibly required communication between two supervisors.

framework, its distributed version, DSDES framework, and PDS representations of a (distributed) SDES. Chapter 5 employs the PDS representation of a DSDES to formally model the synthesis of communication among supervisors and derives solutions to it in the form of information policies. A partitioning of the class of solutions to SCDS is also introduced in this chapter with some preliminary results on communication-based state representations of a centralized supervisor. In Chapter 6, the theoretical results of Chapter 5 is applied to synthesize two information policies for a tree-like network used for asymmetric multicast data transport. Conclusions are drawn and suggestions for future are made in Chapter 7. Proofs of the claims in Chapter 4 and in Chapter 5, which are missing from the text, can be found in Appendices A and B, respectively. Also, details of computations for Chapter 6 are brought in Appendix C.

1.12.1 Some conventions used in mathematical proofs

The following points about proofs are in order. When the proof of a claim is missing, it can be found in Appendix A or Appendix B. In the mathematical proofs, whenever necessary, the justification of an implication is brought in "[]" right after the implication. When the steps of a proof follow continuously, each following step is shown after a symbol " $\Longrightarrow$," i.e. each inference starts at a new line. When a step of a proof is referred to as the justification of another step, the former is assigned a symbol, which is put in "()" in the same way as an equation number. Some steps of a proof consist of multiple conjuncts, in which case the justification for each conjunct is shown in front of it. To improve the readability of the mathematics, the following signs are used as the end of the mathematical environments.

- The end of a definition, assumption, remark, notation, or problem definition is denoted by "□",
- The end of a theorem, proposition, corollary, or lemma is denoted by "■",
- The end of an example is denoted by "◊".

Chapter 2

Background

2.1 Introduction

This chapter provides the basic materials of Ramadge-Wonham supervisory control theory of DESs, as reported in [3], and its decentralized extension in [12]. It also reviews the framework of extended finite-state machines, as appeared in [70], [47], and [48]. The common notation, used in all chapters, is also introduced and fixed here. This notation remains valid for the following chapters, unless otherwise superseded later.

2.2 Model of a discrete-event system

Consider[1] a finite alphabet of events Σ and define $\Sigma^* := \{\epsilon\} \cup \Sigma^+$, where ϵ denotes the zero-length string and Σ^+ is the set of all finite symbol sequences s over Σ. The power set of Σ is denoted by $Pwr(\Sigma)$ A language K is defined as a subset of Σ^* and its *prefix closure* is represented by $\overline{K}$. The *synchronous product* of two languages K_1 and K_2 is shown by $K_1||K_2$. The *catenation* of a

[1]We follow the notations, definitions, and results in chapters 2, 3, and 6 in [3] which originated in [6] and [71].

language K by the events in a set $\Sigma_a \subseteq \Sigma$ is denoted as $K\Sigma_a$ and is a language whose strings are all possible one-step continuations of the strings in K by all the events in Σ_a. A language may be represented[2] as a generator[3] and the operations such as synchronous product on generators are defined in the usual way [72].

A discrete-event system is modeled as a generator $\mathbf{G} = (Q, \Sigma, \xi, q_0, Q_m)$ where Q is the finite set of states, Σ is the finite set of events, ξ is the transition (partial) function, q_0 is the initial state, and Q_m is the finite set of marker states. Whenever $\mathbf{G}$ is represented as $\mathbf{G} = (Q, \Sigma, \xi, q_0)$, it is assumed that all states are marker states, i.e. $Q_m = Q$. The event set is assumed to be the union of the two disjoint controllable and uncontrollable event sets Σ_c and Σ_{uc}, respectively, such that $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Consider another partitioning of the event set into two disjoint sets, observable subalphabet Σ_o, and unobservable subalphabet, Σ_{uo}, such that $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Associated with this second partitioning, define a natural projection $P : \Sigma^* \rightarrow \Sigma_o^*$. This mapping simply erases from a string in Σ^* the events which belong to Σ_{uo}, thereby modeling the window through which the plant's behavior can be observed (by a controller).

We assume that $\mathbf{G}$ is trim, i.e. it is reachable and coreachable[4], and denote its marked and closed languages by $L_m(\mathbf{G})$ (or simply G) and $L(\mathbf{G})$ (or simply $\overline{G}$), respectively. $\mathbf{G}$ is called *nonblocking* if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$. A *safety specification* is specified as a language E whose corresponding generator[5] is $\mathbf{E}$, i.e. $L_m(\mathbf{E}) = E$ and $L(\mathbf{E}) = \overline{E}$.

[2]As a convention, languages and scalers are denoted by plain fonts and generator and vectors are bolded.

[3]A *generator* is the same as a finite-state automaton [72] except that its transition function can be partial. Since it can be a *recognizer* for a language, these two expressions are used interchangeably.

[4]In simple words, $\mathbf{G}$ is reachable if its closed language allows for reaching each of its states from the initial state. It is coreachable if the closed language allows for going from every state to at least one marker state.

[5]As a convention the representation of a language is denoted by the same name, but bolded.

Definition 2.1 ([3], Section 3.4) A specification language $E \subseteq L_m(\mathbf{G})$ is called *controllable* with respect to $L(\mathbf{G})$ if and only if $\overline{E}\Sigma_{uc} \cap L(\mathbf{G}) \subseteq \overline{E}$. □

Definition 2.2 ([3], Section 3.4) Let $E \subseteq L \subseteq \Sigma^*$. E is called L-closed if $E = \overline{E} \cap L$. □

By a *supervisor* we mean another generator $\mathbf{S}$, designed using supervisory control theory [3], whose synchronous product with $\mathbf{G}$, called the closed-loop system, yields a language (or behavior) inside $L(\mathbf{E})$.

Definition 2.3 ([3], Section 3.4) Let $\Gamma = \{\gamma \in Pwr(\Sigma) | \ \gamma \supseteq \Sigma_{uc}\}$. A *supervisory control* for $\mathbf{G}$ is any map $W : L(\mathbf{G}) \to \Gamma$. The closed behavior of $W/\mathbf{G}$ is a language[6] $L(W/\mathbf{G}) \subseteq L(\mathbf{G})$ such that $\epsilon \in L(W/\mathbf{G})$ and

$$s \in L(W/\mathbf{G}) \wedge \sigma \in W(s) \wedge s\sigma \in L(\mathbf{G}) \Leftrightarrow s\sigma \in L(W/\mathbf{G}).$$

We say that W is *nonblocking* for $\mathbf{G}$ if $\overline{L_m(W/\mathbf{G})} = L(W/\mathbf{G})$. □

Theorem 2.1 ([3], Theorem 3.4.1) Let $E \subseteq L_m(\mathbf{G})$ and $E \neq \emptyset$. There exists a nonblocking supervisory control W for $\mathbf{G}$ such that $L_m(W/\mathbf{G}) = E$ (and $L(W/\mathbf{G}) = \overline{E}$) if and only if E is controllable with respect to $\mathbf{G}$ and $L_m(\mathbf{G})$-closed. ■

Proposition 2.2 ([3], Section 3.6) Let $\mathbf{S}$ be any nonblocking DES over Σ such that

(a) $L_m(\mathbf{S})$ is controllable with respect to $L(\mathbf{G})$,

(b) $L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \neq \emptyset$,

(c) $\overline{L_m(\mathbf{S}) \cap L_m(\mathbf{G})} = \overline{L_m(\mathbf{S})} \cap L(\mathbf{G})$.

Let $\emptyset \neq E := L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and let W be a nonblocking supervisory control such that $L_m(W/\mathbf{G}) = E$. Then $\mathbf{S}$ implements W and in particular $L_m(W/\mathbf{G}) = L_m(\mathbf{G}) \cap L_m(\mathbf{S})$ and $L(W/\mathbf{G}) = L(\mathbf{G}) \cap L(\mathbf{S})$. ■

[6] $W/\mathbf{G}$ reads "$\mathbf{G}$ under the supervision of W".

Definition 2.4 ([3], Section 3.6) $\mathbf{S}$ is called a proper supervisor for $\mathbf{G}$ if

(a) $\mathbf{S}$ is trim,

(b) $\mathbf{S}$ is controllable with respect to $\mathbf{G}$, and

(c) $\overline{L_m(\mathbf{S}) \cap L_m(\mathbf{G})} = L(\mathbf{S}) \cap L(\mathbf{G})$.

With an abuse of notation, we denote the close-loop behavior enforced by any such supervisor by $L(\mathbf{S}/\mathbf{G})$.

Definition 2.5 ([3], Section 6.2) A specification $E \subseteq L_m(\mathbf{G})$ is observable with respect to $(\mathbf{G}, P)$ if:

a) $\forall s, s' \in \overline{E}, \forall \sigma \in \Sigma_c.\ s\sigma \in \overline{E} \wedge s'\sigma \in L(\mathbf{G}) \wedge P(s) = P(s') \Longrightarrow s'\sigma \in L(\mathbf{G})$, and

b) $\forall s, s' \in L_m(\mathbf{G}).\ s \in E\ \wedge\ P(s) = P(s') \implies s' \in E.$ □

Definition 2.6 ([3], Section 6.3) A feasible supervisory control for $\mathbf{G}$ is any map $W : L(\mathbf{G}) \rightarrow \Gamma$ such that $\ker(P|L(\mathbf{G})) \leq \ker(W)$[7], where $P|L(\mathbf{G})$ denotes the restriction of P to $L(\mathbf{G})$. □

Theorem 2.3 ([3], Theorem 6.3.1) Let $\emptyset \neq E \subseteq L_m(\mathbf{G})$. There exists a nonblocking feasible supervisory control W for $\mathbf{G}$ such that $L_m(W/\mathbf{G}) = E$ if and only if E is (i) controllable with respect to $\mathbf{G}$, (ii) observable with respect to $(\mathbf{G}, P)$, and (iii) $L_m(\mathbf{G})$-closed. ■

We assume that all specifications are subsets of $L_m(\mathbf{G})$, controllable, observable, and $L_m(\mathbf{G})$-closed. Following the above theorem and proposition, these assumptions guarantee the existence of a proper supervisor $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$[8] for $\mathbf{G}$, such that when running in parallel with the plant, the language of the composition is equal to the specification language E, i.e. $L_m(\mathbf{G}) \cap L_m(\mathbf{S}) = E$.

[7] ker(.) denotes the equivalence kernel of its argument. For two equivalence kernels, $\leq$ denotes that the left-side kernel is finer than the right-side kernel (see Chapter 1 in [3]).

[8] The description of 5-tuple is the same as what explained for $\mathbf{G}$.

2.3 Extended finite-state machines

2.3.1 Preliminaries

In an EFSM[9] a transition is equipped with a guard formula (or simply guard), and when it is taken, it triggers a number of actions. A set X of Boolean variables is introduced. A transition in the EFSM is enabled if and only if its *guard formula*, which is a predicate defined as a Boolean formula over X, evaluates to *true* (1). When a transition is taken, $|X|$ actions may follow. An action is a Boolean function that assigns a new value to a variable based on the old values of all variables. In the following definition, let $k = |X|$, $\mathcal{G}$ denote the set of all Boolean formulas over X, and $\mathcal{A}$ denote the set of all Boolean functions $b : \mathbb{B}^k \rightarrow \mathbb{B}$.

Definition 2.7 [49] An EFSM $\mathbf{L_x}$ is defined as an 8-tuple $\mathbf{L_x} = (Q, \Sigma, \xi, q_0, Q_m, X, g, a)$, where Q is a finite, nonempty, set of states; Σ is a finite alphabet; $\xi : Q \times \Sigma \rightarrow Q$ is a partial transition function; q_0 is the initial state; Q_m is the set of marker states; X is a finite set of Boolean variables; $g : \Sigma \rightarrow \mathcal{G}$ assigns to each event a *guard formula*; $a : X \times \Sigma \rightarrow \mathcal{A}$ assigns to each pair of event and variable an *action*. □

Assume that all variables are initialized to *false* (0). We extend ξ to $Q \times \Sigma^*$ in the usual way. For $\alpha \in \Sigma$, the guard formula $g(\alpha)$ is a Boolean formula with which all transitions labeled with α are guarded. For $\alpha \in \Sigma$ and $x \in X$, the action $a(x, \alpha) : \mathbb{B}^k \rightarrow \mathbb{B}$ is a Boolean function. When α is taken, it results in the assignment $x := a(\alpha, x)(\mathbf{v})$, where the vector $\mathbf{v}$ is the current values of variables in X. Let $V : \Sigma^* \rightarrow \mathbb{B}^k$ be a map that assigns to a string $s \in \Sigma^*$ a tuple of Boolean values obtained from recursively applying

[9]The basics of EFSMs are explained below. The reader is referred to [48] and [47] for more details.

the actions of events in s to $\mathbf{0}$[10], that is:

$$V(s) = \big(v(s,x)\big)_{x\in X} \tag{2.1}$$

where for $s \in \Sigma^*$, $\sigma \in \Sigma$ and $x \in X$, the function $v : \Sigma^* \times X \to \mathbb{B}$ is recursively defined as follows:

$$v(\epsilon, x) := 0 \ \wedge \ v(s\sigma, x) := a^x_\sigma(V(s)). \tag{2.2}$$

The closed language of $\mathbf{L_x}$, denoted by L_x, contains a string generated by $\mathbf{L_x}$ if guard formulas are respected at all its prefixes, i.e. $\epsilon \in L_x$ and

$$s \in L_x \wedge \xi(q_0, s\sigma)! \wedge g_\sigma(V(s)) = 1 \Leftrightarrow s\sigma \in L_x. \tag{2.3}$$

By virtue of having a control mechanism embedded in their structure, EFSMs can be used to model closed-loop systems. The synchronous product of EFSMs exists if they are *consistent*, that is, the actions of a common variable triggered by a common event are syntactically equal [49].

2.3.2 Implementation of supervisory control map by an EFSM

Given the automaton of a proper supervisor $\mathbf{S} = (Y, \Sigma, \xi, y_0, Y_m)$ over a plant $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, we implement the supervisory control map by extending $\mathbf{G}$ to an EFSM $\mathbf{G_x} = (Q, \Sigma, \delta, q_0, Q_m, X, g, A)$. EFSM $\mathbf{G_x}$ can be regarded as the closed-loop system satisfying the control objectives.

Let $\mathcal{M}_k$ be the set of all minterms[11] over k Boolean variables. For

[10] $\mathbf{0}$ denotes the zero vector whose dimension is clear from the context.

[11] Over a set of Boolean variables, a minterm is the product in which every Boolean variable appears in normal or complemented form (see [73]).

convenience we define two injective maps: *state-label* map ℓ and *label-minterm* map m. A state-label map $\ell : Y \rightarrow \mathbb{B}^k$ assigns unique labels to states in Y. A label-minterm map $m : \mathbb{B}^k \rightarrow \mathcal{M}_k$ associates each label with the unique minterm that is *true* at that label. Note that for the state-label map to be injective it is necessary that $|Y| \leq 2^k$.

The first five components of $\mathbf{G_x}$ are identical to those of $\mathbf{G}$, while X, g and A are derived from $\mathbf{S}$ as follows:

- $X = \{x_1, x_2, \ldots, x_k\}$, where $k = \lceil \log_2 |Y| \rceil$.

- For $\sigma \in \Sigma$,

$$g_\sigma = \begin{cases} 1 & , \sigma \in \Sigma_{uc} \\ \sum_{l \in L_{g(\sigma)}} m(l) & , \sigma \in \Sigma_c \end{cases} \tag{2.4}$$

  where $L_{g(\sigma)} = \{\ell(y) \mid y \in Y \wedge \xi(y, \sigma)!\}$. The set $L_{g(\sigma)}$ consists of the labels of all supervisor states where σ is enabled.

- For $\sigma \in \Sigma$, $A_\sigma = \left(a_\sigma^x\right)_{x \in X}$ where for $x \in X$,

$$a_\sigma^x := \sum_{l \in L_{a(\sigma,x)}} m(l) \tag{2.5}$$

  and $L_{a(\sigma,x)} = \{\ell(y) \mid y \in Y \wedge [\!| x |\!]_{\xi(y,\sigma)} = 1\}$. For $y \in Y$, the expression $[\!| x |\!]_y$ denotes the value of Boolean variable x in state y. The set $L_{a(\sigma,x)}$ consists of the labels of all supervisor states from which x becomes 1 *after* the occurrence of σ. □

The supervisory control map enforced by $\mathbf{S}$ is thus *encoded* in EFSM $\mathbf{G_x}$.

Theorem 2.4 [49] For EFSM $\mathbf{G_x}$ designed as above we have $L(\mathbf{G_x}) = L(\mathbf{S/G})$. In addition, $L_m(\mathbf{G_x}) = L_m(\mathbf{S/G})$ if $L_m(\mathbf{S})$ is $L_m(\mathbf{G})$-closed. ■

The above formalism may be also used to implement the modular supervisory control in which, for a distributed plant, control *decisions* are made *globally* (as reflected by the fact that the variables in X are global) [49]. The rest of this chapter focuses on the *decentralized control*, where for a distributed plant, control *decisions* are made *locally*.

2.4 Decentralized embedded supervisory control

In distributed systems where plant components are geographically widely separated, a centralized supervisor satisfying the global control objectives cannot be implemented; rather we need a *decentralized* solution, i.e., a set of local, embedded supervisors, each designed to monitor and control a plant component. A supervisor acting on all controllable events in the entire event set is called a *global* supervisor; in contrast, a supervisor that can only monitor and control subsets of events pertaining to a component is said to be *local*. A decentralized solution prescribes the control action that each local supervisor must take.

2.4.1 Plant model and problem definition

Let[12] $\mathbf{G_i}$ denote a local plant component over $\Sigma_i \subseteq \Sigma$, where $i \in I = \{1, \ldots, n\}$, and $\mathbf{G}$ denote the overall plant, which is the synchronous product of local components and defined over $\Sigma = \bigcup_{i \in I} \Sigma_i$. Let $\Sigma_{c,i} \subseteq \Sigma_i$ and $\Sigma_{o,i} \subseteq \Sigma_i$ denote the local controllable and observable event sets for supervisor $\mathbf{S_i}$, respectively and define $\Sigma_c = \bigcup_{i \in I} \Sigma_{c,i}$ and $\Sigma_o = \bigcup_{i \in I} \Sigma_{o,i}$. We selfloop all events in $\Sigma \setminus \Sigma_i$ at every state of $\mathbf{G_i}$ to obtain $\tilde{\mathbf{G}}_\mathbf{i}$. Consider the natural

[12]For the decentralized control, we follow the notation in [12].

projection $P_{o,i} : \Sigma^* \rightarrow \Sigma_{o,i}^*$. Define the local supervisor $\mathbf{S_i}$ by its control map $W_i : P_{o,i}[L(\mathbf{G})] \rightarrow 2^{\Sigma_i}$ such that $W_i[P_{o,i}(s)] \supseteq \Sigma_i \setminus \Sigma_{c,i}$ for all $s \in L(\mathbf{G})$. Let $\tilde{\mathbf{S}}_\mathbf{i}$ be the *global copy* of $\mathbf{S_i}$, i.e. it inherits from $\mathbf{S_i}$ the control decisions and transitions on events in $\Sigma_{c,i}$ and in $\Sigma_{o,i}$, respectively, and on top of this, it enables all events in $\Sigma \backslash \Sigma_{c,i}$ and does not change its states upon the occurrence of the events in $\Sigma \backslash \Sigma_{o,i}$, hence it is computed by adding $\Sigma \backslash \Sigma_i$ selfloops to every state of $\mathbf{S_i}$. Defined over Σ, $\mathbf{G_{ix}}$ implements the closed loop system $\tilde{\mathbf{S}}_\mathbf{i}/\tilde{\mathbf{G}}_\mathbf{i}$. From now on, we assume that the global specification E, $\emptyset \neq E \subseteq L_m(\mathbf{G})$ is controllable with respect to $\mathbf{G}$.

The main decentralized control problem we study is called Global Problem with Zero Tolerance (*GPZT*) [12]. This problem was defined originally using the *conjunctive fusion rule*, under which an event is disabled globally if at least one supervisor, who observes it and can exercise control over it, disables it. As a result the behavior of the system would be limited by all supervisors, i.e. [3]

$$L(\tilde{\mathbf{S}}_\mathbf{1} \wedge \ldots \wedge \tilde{\mathbf{S}}_\mathbf{n}/\mathbf{G}) \;=\; L(\tilde{\mathbf{S}}_\mathbf{1}/\mathbf{G}) \cap \ldots \cap L(\tilde{\mathbf{S}}_\mathbf{n}/\mathbf{G}). \tag{2.6}$$

Problem 2.1 [12] (GPZT) Given a plant $\mathbf{G}$ over Σ, and $\Sigma_{c,1}$, $\Sigma_{c,2}$, $\Sigma_{o,1}$, $\Sigma_{o,2} \subseteq \Sigma$, is there any n-tuple of local supervisors $(\mathbf{S_1}, \ldots, \mathbf{S_n})$ such that $L_m(\tilde{\mathbf{S}}_\mathbf{1} \wedge \ldots \wedge \tilde{\mathbf{S}}_\mathbf{n}/\mathbf{G}) = E$? □

Variants of this problem might be easily defined under other fusion rules [16]. Motivated by the work in [12], the Embedded GPZT (*EGPZT*) may be defined as follows:

Problem 2.2 (EGPZT) Given component plants $\mathbf{G_i}$ over Σ_i for $i \in I$, a language $E \subseteq L_m(\mathbf{G_1} \parallel \ldots \parallel \mathbf{G_n})$, and sets $\Sigma_{c,i}$ and $\Sigma_{o,i}$, is there any n-tuple of *extended* systems (EFSMs) $(\mathbf{G_{1x}}, \ldots, \mathbf{G_{nx}})$ such that $L_m(\mathbf{G_{1x}} \parallel \ldots \parallel \mathbf{G_{nx}}) = E$? □

Notice that while the two problems are fundamentally equivalent, the definition of Problem 2.2 in terms of a set of component plants emphasizes the distributed nature of the system.

2.4.2 Solution: coobservable specifications

We consider the conditions for the existence of solutions to the above-mentioned problems. The solvability of GPZT is related to *coobservability* of E [12][13]. For simplicity we choose $n = 2$, although the results can be generalized to any finite number of supervisors.

Proposition 2.5 A specification $E \subseteq L_m(\mathbf{G})$ is coobservable with respect to $(\mathbf{G}, P_{o,1}, P_{o,2})$ if and only if the following holds[14].

a) [1] $\forall s \in \overline{E}, \sigma \in \Sigma_c$.

$$s\sigma \notin \overline{E} \wedge s\sigma \in L(\mathbf{G}) \Longrightarrow [\exists i \in I.\ (P_{o,i}^{-1} P_{o,i}(s))\sigma \cap \overline{E} = \emptyset \ \wedge \ \sigma \in \Sigma_{c,i}],$$

b) [12] For i in part a,

$$\forall s, s' \in L(\mathbf{G}).\ P_{o,i}(s) = P_{o,i}(s') \Longrightarrow [s' \in E \wedge s \in \overline{E} \cap L_m(\mathbf{G}) \Rightarrow s\sigma \in E].\blacksquare$$

In plain words, a language E is coobservable with respect to $(\mathbf{G}, P_{o,1}, P_{o,2})$ if (a) *at least* one supervisor can detect and prevent violations of $\overline{E}$ and (b) the decision as to whether or not to mark a string can be made by at least one of the supervisors.

Theorem 2.6 [12] There exist supervisors $(\mathbf{S_1}, \mathbf{S_2})$ that solve GPZT if and only if E is controllable with respect to $\mathbf{G}$ and *coobservable* with respect to $(\mathbf{G}, P_{o,1}, P_{o,2})$. $\blacksquare$

Once GPZT is solved, the decentralized control might be embedded as a set of EFSMs, too. In [12] $\mathbf{S_i}$ is constructed as an automaton with a feedback

[13]This property is called C&P coobservability in [15].

[14]This definition of coobservability is a blend of its original definition in [12] and a more compact equivalent form of it in [1].

map provided that E is coobservable. Below we employ that approach to embed the control of $\tilde{\mathbf{S}}_\mathbf{i}$.

Construction of $\tilde{\mathbf{S}}_\mathbf{i}$: Let $\mathbf{E} = (Y, \Sigma, \xi, y_0, Y_m)$ be a recognizer for a controllable and coobservable specification language E. For $i = 1, 2$, let Y_i be the set of nonempty subsets of Y. Since Y is finite, Y_i is guaranteed to be finite. The supervisor $\tilde{\mathbf{S}}_\mathbf{i} = (Y_i, \Sigma, \xi_i, y_{0i}, Y_{mi})$ is given by:

$\forall \sigma \in \Sigma_{o,i}, y_i \in Y_i$:

$$\xi_i(y_i, \sigma) = \begin{cases} \{\xi(y, s) \mid s \in \Sigma^*, y \in y_i, P_{o,i}(s) = \sigma\} & \\ & \text{if this is nonempty} \\ undefined & \text{otherwise} \end{cases}$$

$\forall \sigma \in (\Sigma_i \setminus \Sigma_{o,i}), y_i \in Y_i$:

$$\xi_i(y_i, \sigma) = \begin{cases} y_i & \text{if } \exists y \in y_i, \xi(y, \sigma)! \\ undefined & \text{otherwise} \end{cases}$$

$\forall \sigma \in (\Sigma \setminus \Sigma_i), y_i \in Y_i : \xi_i(y_i, \sigma) = y_i,$

$$y_{0i} := \{\xi(y_0, s) \mid s \in \Sigma^*, P_{o,i}(s) = \epsilon\}$$
$$Y_{mi} := \{y_i \in Y_i \mid (\exists y \in y_i) y \in Y_m\}$$

Next, we extend $\tilde{\mathbf{G}}_\mathbf{i}$ to $\mathbf{G_{ix}}$ to enforce the supervision of $\tilde{\mathbf{S}}_\mathbf{i}/\tilde{\mathbf{G}}_\mathbf{i}$. Since $\tilde{\mathbf{S}}_\mathbf{i}$ is constructed over Σ, $\mathbf{G_{ix}}$ is defined over Σ. The following result states that $\mathbf{G_{1x}}||\mathbf{G_{2x}}$ implements E by enforcing the supervision of $\tilde{\mathbf{S}}_\mathbf{1}$ and $\tilde{\mathbf{S}}_\mathbf{2}$[15].

Lemma 2.1 $L_m(\mathbf{G_{1x}} \parallel \mathbf{G_{2x}}) = L_m(\tilde{\mathbf{S}}_\mathbf{1} \wedge \tilde{\mathbf{S}}_\mathbf{2}/\mathbf{G})$. ■

Theorem 2.7 The marked language of the overall controlled system is equal to E, i.e., $L_m(\mathbf{G_{1x}} \parallel \mathbf{G_{2x}}) = E$. ■

[15]Note that an event unobservable to $\mathbf{G_i}$ does not update any of its Boolean variables as it can only appear as a selfloop in some states of $\tilde{\mathbf{G}}_\mathbf{i}$.

We conclude this subsection by a simple example to show the above scheme works when a controllable specification is coobservable, and how it fails when it is not.

Example 2.1 Local plants $\mathbf{G_1}$ and $\mathbf{G_2}$ defined respectively over $\Sigma_1 = \{\alpha, \gamma\}$ and $\Sigma_2 = \{\beta, \gamma\}$, and the specification $\mathbf{E}$ are shown in Fig. 2.1(a), where $\Sigma_{o,1} = \{\alpha\}$, $\Sigma_{o,2} = \{\beta\}$ and $\Sigma_{c,1} = \Sigma_{c,2} = \{\gamma\}$. The overall plant $\mathbf{G}$ is the synchronous product of $\mathbf{G_1}$ and $\mathbf{G_2}$, and is shown in Fig. 2.1(b) together with $\tilde{\mathbf{G}}_1$ and $\tilde{\mathbf{G}}_2$. It can be verified that E is coobservable with respect to $(\mathbf{G}, P_{o,1}, P_{o,2})$. Therefore we can design $\tilde{\mathbf{S}}_1$ and $\tilde{\mathbf{S}}_2$ supervising the local plant components to ensure that the language E is marked by the closed-loop system.

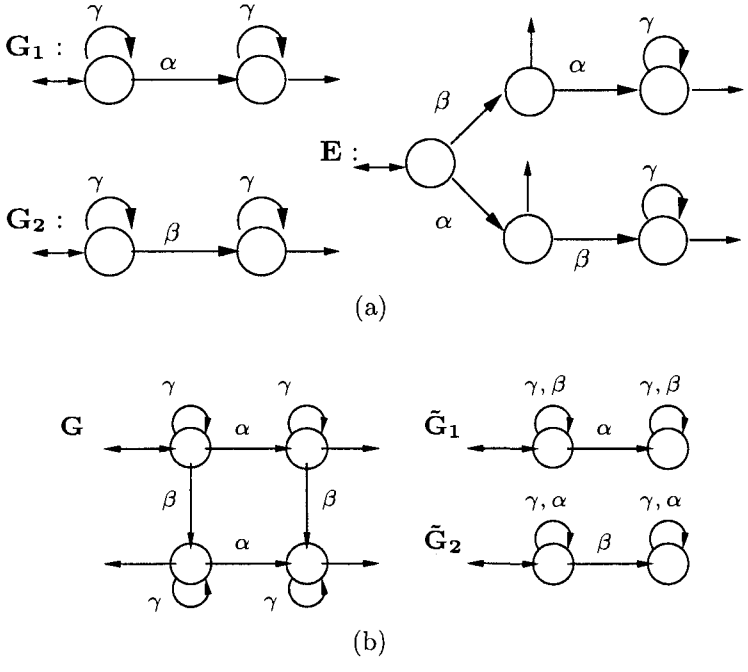


Figure 2.1: (a) Decentralized embedded supervisory control with a coobservable specification: component plants and specification. (b) Plants $\mathbf{G}$, $\tilde{\mathbf{G}}_1$ and $\tilde{\mathbf{G}}_2$.

Using the procedure of this section a suitable pair of supervisors $\tilde{\mathbf{S}}_1$ and

$\tilde{\mathbf{S}}_2$ are designed as shown in Fig. 2.2(a). Next the supervision of $\tilde{\mathbf{S}}_\mathbf{i}$ on $\tilde{\mathbf{G}}_\mathbf{i}$ is embedded in $\mathbf{G_{ix}}$, shown also in Fig. 2.2(a). The overall controlled system is shown in Fig. 2.2(b), which satisfies $L_m(\mathbf{G_{1x}} \parallel \mathbf{G_{2x}}) = E$.

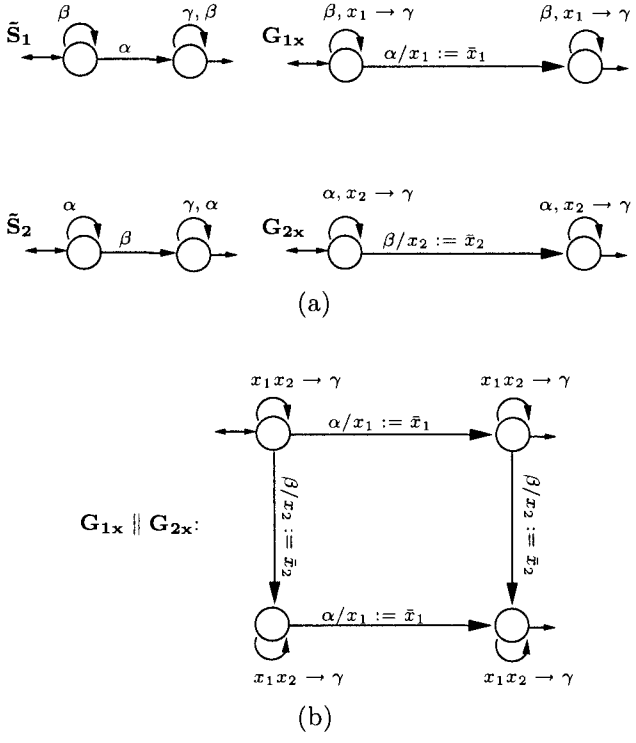


Figure 2.2: (a) Supervisors $\tilde{\mathbf{S}}_1$ and $\tilde{\mathbf{S}}_2$, and embedded controllers $\tilde{\mathbf{G}}_{\mathbf{1x}}$ and $\tilde{\mathbf{G}}_{\mathbf{2x}}$. (b) Overall controlled system $\mathbf{G_{1x}} \parallel \mathbf{G_{2x}}$.

A decentralized solution does not exist when the legal language[16] is not coobservable. For instance, for the language of $\mathbf{E}'$ in Fig. 2.1 neither of the local supervisors can distinguish between $s = \alpha\beta$ and $s' = \beta\alpha$, which is crucial in deciding whether to disable γ. Thus E' cannot be implemented unless local supervisors "talk" to each other. $\Diamond$

[16]In this thesis, "legal" language is meant "specification."

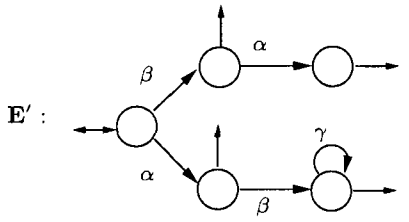


Figure 2.3: A specification that is not coobservable.

2.5 Conclusion

This chapter introduces the main notation used throughout the thesis, EFSM framework as developed in [48] and [49], and its extension to the decentralized case as done in [47]. As can be seen, the original idea behind EFSM framework is to "embed" the centralized or decentralized supervisory control of "already designed" supervisors in guard formulas and actions, which are defined over Boolean variables. Whereas this was to "bridge the gap" [49] between traditional ad hoc designs of DESs and those based on supervisory control theory, the concept of capturing a supervisor's observation and control tasks in functions is used for the synthesis of decentralized (communicating) supervisors in later chapters.

Chapter 3

Decentralized Supervisory Control of Discrete-Event Systems over Communication Networks

3.1 Introduction

A common example of a distributed DES is a communication network in which processes exchange among themselves data messages under an ordering specified by a set of rules, known as a communication protocol [67]. While the use of formal methods has significantly contributed to specifying the desired behavior of systems and validation techniques for communication protocols [68], the control, in turn, community has considered the automation of the synthesis problem. In [13] the example of Alternating Bit Protocol (ABP) was first introduced as a solution to a decentralized supervisory control problem. However, since both plant components (i.e. sender and receiver) and the specification

models "spelled out" the solution, the synthesis was ad hoc. In a later formulation in [69], the specification does not contain the solution and requires only a linear ordering among some events. It is shown that inclusion of ABP in the sender model makes the specification coobservable with respect to the plant and thus there exists a set of non-communicating decentralized supervisors which yield ABP. Conversely, in the absence of this inclusion coobservability does not hold and such a set does not exist.

When coobservability fails it may still be possible to design decentralized supervisors by allowing communication among them. In fact, the findings of this research support the idea that such problems yield protocols which require communication of some control- or observation-related information among the supervisors. In the proposed formulation of the problem in this chapter, assuming ideal communication channels, the protocol design for a special class of non-coobservable specifications, including ABP, is reduced to the Synthesis of Communicating Decentralized Supervisors (SCDS).

The initial perspective of this research to study SCDS relied on the formalism of Extended Finite State Machines (EFSMs) in which bits of control information necessary in the process of decision-making, rather than events or state estimates, are communicated from one supervisor to another. An EFSM implements supervisory control [6] by employing Boolean variables to encode the supervisor's states, a set of Boolean functions to observe events, and Boolean formulas to control transitions [49]. This formalism was extended in [47] to the decentralized case by assigning a set of private variables to each component EFSM to make decision-making possible at local sites. The decision as to whether enable or disable a local event may in general depend on the values of supervisor's own private variables, and the local copies of

variables owned by other supervisors. These copies are updated by communication among local supervisors. It is shown that the dependence of actions on copy variables is related to a modified version of "joint observability" [31]. Solutions are developed for a special class of problems where the sole purpose of communication is control [47].

This chapter[1] introduces a distributed extension of EFSM framework and applies it to protocol design for non-coobservable specifications. This approach requires neither the plant components nor the specification to "spell out" the protocol (i.e. part or all of the protocol need not be designed and included in the transition structures of plant components or specification beforehand). In the first part of this chapter, we define Discrete-Event Control over Communication Networks (DECCN) problem and present its partial solution under the assumption of ideal channels. Thereby it completes the previous works on ABP, a practical benchmark and an illustrative example, by showing that the protocol arises naturally as a solution to the corresponding control problem with no *a priori* inclusion of the solution in the plant model or the specification language. This result is then extended to other special classes of protocol design problems for ideal channels. Moreover, the difficulties of tackling unreliable channels is discussed briefly and some positive results are presented.

The rest of this chapter is organized as follows: After a brief review of EFSM formalism in Section 3.2, Section 3.3 states the general problem. Section 3.4 then formulates the ABP problem in EFSM framework and synthesizes ABP as a solution to this problem under the assumption of ideal channels. Section 3.5 discusses ways in which the problem can be generalized for ideal channels and the case of unreliable communication channels are

[1]This chapter has been published as [50].

discussed in Section 3.6.

3.2 Extended finite-state machines

Notation: In this chapter we assume that all state machines are deterministic. We denote a state machine and its generated (closed) language by bold and regular capital letters, respectively.

In an EFSM[2] a transition is equipped with a guard formula, and when it is taken it triggers a number of actions. A set X of Boolean variables is introduced. A transition in the EFSM is enabled if and only if its *guard formula*, which is a predicate defined as a Boolean formula over X, evaluates to *true* (1). When a transition is taken, $|X|$ actions may follow. An action is a Boolean function that assigns a new value to a variable based on the old values of all variables. Given the set X, in the following definition let $k = |X|$, $\mathcal{G}$ denote the set of all Boolean formulas over X, and $\mathcal{A}$ denote the set of all Boolean functions $b : \mathbb{B}^k \to \mathbb{B}$.

Definition 3.1 [49] An EFSM $\mathbf{L_x}$ is defined as a 7-tuple $\mathbf{L_x} = (Q, \Sigma, \xi, q_0, X, g, a)$, where $\mathbf{L} = (Q, \Sigma, \xi, q_0)$[3] is an FSM in which Q is a finite set of states; Σ is a finite alphabet; $\xi : Q \times \Sigma \to Q$ is a partial transition function; $q_0 \in Q$ is an initial state; X is a finite set of Boolean variables; $g : \Sigma \to \mathcal{G}$ assigns to each event a *guard formula*; and $a : X \times \Sigma \to \mathcal{A}$ assigns to each pair of event and variable an *action*. When $\mathbf{L}$ is understood from the context, $\mathbf{L_x}$ is simply written as $\mathbf{L_x} = (-, X, g, a)$. □

Assume that all variables are initialized to *false* (0). We extend ξ to $Q \times \Sigma^*$ in the usual way. For $\alpha \in \Sigma$, the guard formula $g(\alpha)$ is a Boolean

[2]The main part of this section is copied from Subsection 2.3.1 to improve the readability of the text.

[3]Whenever $\mathbf{L}$ is represented as a quadruple $\mathbf{L} = (Q, \Sigma, \xi, q_0)$, it is assumed that all states are marker states, i.e. $Q_m = Q$.

formula with which all transitions labeled with α are guarded. For $\alpha \in \Sigma$ and $x \in X$, the action $a(x, \alpha) : \mathbb{B}^k \rightarrow \mathbb{B}$ is a Boolean function. When α is taken, it results in the assignment $x := a(x, \alpha)(\mathbf{v})$, where the vector $\mathbf{v}$ represents the current values of variables in X.

Let $V : \Sigma^* \rightarrow \mathbb{B}^k$ be a map that assigns to every string $s \in \Sigma^*$ a tuple of Boolean values obtained from recursively applying the actions of events in s to $\mathbf{0}$, that is:

$$V(s) = \big(v(s, x)\big)_{x \in X} \tag{3.1}$$

where for $s \in \Sigma^*$, $\sigma \in \Sigma$ and $x \in X$, the function $v : \Sigma^* \times X \rightarrow \mathbb{B}$ is recursively defined as $v(\epsilon, x) := 0$ and $v(s\sigma, x) := a(x, \sigma)(\big(v(s, x)\big)_{x \in X})$. Let the expression $\xi(q_0, s)!$ denote the fact that the string $s \in \Sigma^*$ belongs to the closed language of $\mathbf{L}$. The closed language of $\mathbf{L_x}$, denoted by L_x, contains a string generated by $\mathbf{L_x}$ if guard formulas are respected at all its prefixes, i.e.:

$$\epsilon \in L_x \text{ and, } s \in L_x \wedge \xi(q_0, s\sigma)! \wedge g(\sigma)(V(s)) = 1 \Leftrightarrow s\sigma \in L_x. \tag{3.2}$$

By virtue of having a control mechanism embedded in their structures, EFSMs can be used to model closed-loop systems. It is shown in [49] and [74] that when the control action of a centralized supervisor is encoded by plant components' EFSMs, the language of the synchronous product of the EFSMs is equal to the language of the system under supervision.

3.3 Problem statement

Fix an index set $I = \{1, \ldots, n\}$ and consider a system $\mathcal{N}$ consisting of n communicating parallel processes $\mathbf{P_{1x}}, \ldots, \mathbf{P_{nx}}$ which are connected through a strongly connected network of potentially unreliable channels in which data

may be lost or delayed. Accordingly, an ideal channel is defined to be one in which data is instantly transmitted without any losses. We refer to the set of rules governing the exchange of data among these processes as a *communication protocol* or in short *protocol* [67]. For brevity we write $\mathbf{P_{ix}} \rightarrow \mathbf{P_{jx}}$ when there is a potentially unreliable channel from $\mathbf{P_{ix}}$ to $\mathbf{P_{jx}}$. Fig. 3.1 shows the network topology for the case when $n = 4$. Each process $\mathbf{P_{ix}}$ is modeled by an EFSM, to which we assign sets $\Sigma_{o,i}$, $\Sigma_{uo,i}$, and $\Sigma_{c,i} \subseteq \Sigma_{o,i} \cup \Sigma_{uo,i}$ of respectively observable, unobservable, and controllable events by the process. Each β_{ij} label, $i, j \in I, i \neq j$, represents a *set* of communication-related events between two processes $\mathbf{P_{ix}}$ and $\mathbf{P_{jx}}$, each of which can be exclusively observed by the two processes (see the following subsection).

3.3.1 Processes

Each process $\mathbf{P_{ix}}$ is modeled by an EFSM $\mathbf{P_{ix}} = (Q_i, \Sigma_i, \xi_i, q_{0i}, X_i, g_i, a_i)$, where $(i \in I)$

- $\Sigma_i = \Sigma_{o,i} \dot{\cup} \Sigma_{uo,i} \dot{\cup} \{\beta^s_{ij} |\ \mathbf{P_{ix}} \rightarrow \mathbf{P_{jx}}\} \dot{\cup} \{\beta^e_{ji}, \beta^r_{ji} | \mathbf{P_{jx}} \rightarrow \mathbf{P_{ix}}\}$;

- $X_i = X_{ii} \dot{\cup} X_{ci}$ where X_{ii} is the set of private variables of process i whose k^{th}

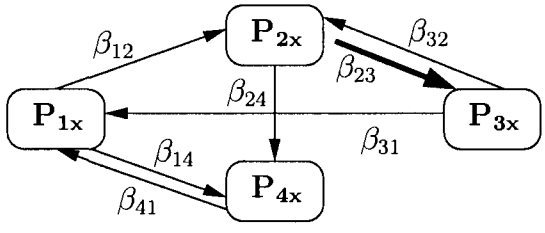


Figure 3.1: A network of n communicating parallel processes ($n = 4$). Each process $\mathbf{P_{ix}}$ has a number of observable, unobservable and communication-related events. A process may be connected to others through ideal (bold arrows) or potentially unreliable (regular arrows) channels.

($k \in \mathbb{N}$) element is denoted by x_{ii}^k (k is removed when X_{ii} is a singleton), and $X_{ci} = \bigcup_{j \in I, j \neq i} X_{ij}$, where X_{ij} stores copies of process $\mathbf{P_{jx}}$'s private variables, $j \in I$, $j \neq i$. A copy of the k^{th} private variable of process j, $j \neq i$, which is stored in X_{ij}, is denoted by x_{ij}^k. All sets are finite;

- Guards and actions are to be designed from the centralized supervisor, except for the following "updates" which are fixed *a priori:*

  - When $\mathbf{P_{jx}} \rightarrow \mathbf{P_{ix}}$, $a_i(X_{ik}, \beta_{ji}^r) = X_{jk}$, $k \in I$, $k \neq i$, which is an abbreviation for $\forall l \in \mathbb{N}.\ a_i(x_{ik}^l, \beta_{ji}^r) = x_{jk}^l$. Similarly, write $X_{ik} := X_{jk}$ when $\forall l \in \mathbb{N}.\ x_{ik}^l := x_{jk}^l$.

The alphabet of process $\mathbf{P_{ix}}$ includes its observable and unobservable events in $\Sigma_{o,i}$ and $\Sigma_{uo,i}$, communication events β_{ij}^s for each process $\mathbf{P_{jx}}$ to which process $\mathbf{P_{ix}}$ sends communication through a potentially unreliable channel, and two events β_{ji}^e and β_{ji}^r for each process $\mathbf{P_{jx}}$ from which process $\mathbf{P_{ix}}$ receives erroneous and error-free communication, respectively, through an unreliable channel. The set X_i consists of variables in the set X_{ii} which are *private* to $\mathbf{P_{ix}}$, and sets of variables X_{ij}, $j \neq i$, which are used to store copies of private variables of processes $\mathbf{P_{jx}}$ (i.e. X_{jj}). When a communication from $\mathbf{P_{jx}}$ is received error-free (event β_{ji}^r) all local copies in X_{ci} are updated with the values of the corresponding variables in $\mathbf{P_{jx}}$, that is, $\forall k \neq i.\ X_{ik} := X_{jk}$. This guarantees that process $\mathbf{P_{ix}}$ is updated with the values of the private variables of $\mathbf{P_{jx}}$, i.e. $\forall l.\ x_{ij}^l := x_{jj}^l$. Moreover, by updating other local copies in X_{ik}, $k \neq j$, with the corresponding values of the variables in $\mathbf{P_{jx}}$, one can insure that local copies are updated even when no *direct* connection between a pair of processes exists. For instance, if process $\mathbf{P_{3x}}$ communicates to process $\mathbf{P_{1x}}$ only through process $\mathbf{P_{2x}}$, then variables in X_{13} are updated with the values of variables in X_{33} after communication events β_{32}^r and β_{21}^r occur in

sequence: β^r_{32} results in the assignment $X_{23} := X_{33}$, and subsequently β^r_{21} updates $X_{13} := X_{23}(= X_{33})$.

We impose no restriction on the structure of $\mathbf{P_{ix}}$ except that when an event in $\Sigma_{c,i}$ is disabled by a protocol none of the communication events β^s_{ij}, β^e_{ji} or β^r_{ji} are affected. This restriction insures that the assumption that the network is strongly connected always remains valid.

Notation: In what follows we let $\Xi = \bigcup_{i\in I} \Sigma_i$, $\Sigma = \bigcup_{i\in I} \left(\Sigma_{o,i} \cup \Sigma_{uo,i}\right)$, $\Sigma_o = \bigcup_{i\in I} \Sigma_{o,i}$, $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, and $I_o(\sigma) = \{i \in I \mid \sigma \in \Sigma_{o,i}\}$. Define the natural projections $\pi : \Xi^* \longrightarrow \Sigma^*$ to erase the communication-related (β) events, $\pi_o : \Sigma^* \longrightarrow \Sigma_o^*$ to erase unobservable events, and $\pi_i : \Sigma^* \longrightarrow \Sigma_{o,i}^*$ to specify the observation window of process i. Also let L and E be respectively the plant and specification languages such that $L \subseteq \Xi^*$ and $E \subseteq \pi(L) \subseteq \Sigma^*$.

3.3.2 Channels

A process $\mathbf{P_{ix}}$ may communicate to process $\mathbf{P_{jx}}$ through a communication channel C_{ij} whenever it exists. Channels may be *ideal* or *unreliable*. In our diagrams, unreliable and ideal channels are denoted by regular and bold arrows, respectively.

Unreliable channels

In practice channels can be potentially unreliable: data could get lost or corrupted, and communication delay cannot be ignored. When $\mathbf{P_{ix}} \rightarrow \mathbf{P_{jx}}$, the unreliable channel C_{ij} is modeled as depicted in Fig. 3.2.

When event β^s_{ij} occurs in $\mathbf{P_{ix}}$ the values of all variables in X_i are transmitted to the channel C_{ij}. *Eventually* the message is delivered to $\mathbf{P_{jx}}$. We assume that each process has perfect error-detection facilities. If the message

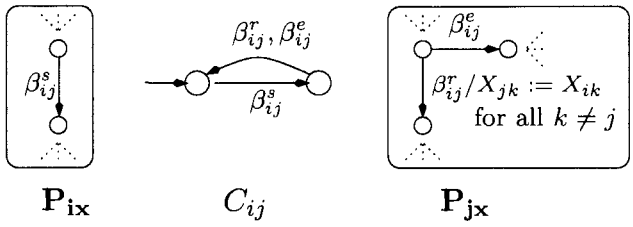


Figure 3.2: An unreliable channel.

delivered by the channel is erroneous (event β^e_{ij}), communication has failed and process $\mathbf{P_{jx}}$ might just change state. If the communication is successful (event β^r_{ij}), then $\mathbf{P_{jx}}$ updates all but its own private variables with the values of $\mathbf{P_{ix}}$'s variables received from the channel.

Ideal channels

A channel is ideal when it is free from any communication loss or delay. While the assumption of ideality lets one focus on the "logical" aspects of the control problem, it is also valid in communication networks where communication delay is negligible compared to the processing time at each site. In an ideal network, each process has instant access to all variables of all other processes which it needs for reevaluating its guard and actions. Focusing on ideal channels enables us to find out *what* needs to be communicated in order to achieve the control objective, without worrying about the logistics of such communication, which will be dealt with in Section 3.6.

3.3.3 Control problem

Assumption 3.1 We assume that the desired behavior of the network is specified by a prefix-closed language E which is controllable with respect to $\pi(L)$ and observable with respect to $\pi(L)$ and π_o. Therefore, there always exists a centralized supervisor, say $\mathbf{S} = (R, \Sigma, \eta, r_0)$, which enforces E, i.e.

$\pi(S||L) = E$ [7]. Note that events in Σ_{uo} may appear only as selfloops in $\mathbf{S}$ and are left out from our transition diagrams.

The control objective is then to design a controller for each process such that the natural projection of the language $P_{1x}||P_{2x}||\ldots||P_{nx}$ onto Σ equals E. Notice that since the control map is embedded in each process model, implementing the centralized control map reduces to finding suitable guard formulas and actions for each process.

Definition 3.2 Discrete-Event Control over Communication Networks (DECCN): Let $\mathcal{N}$ be a system consisting of n communicating parallel processes $\mathbf{P_{1x}}, \ldots, \mathbf{P_{nx}}$, each modeled by an EFSM as in Subsection 3.3.1, which are connected through a strongly connected network of potentially unreliable channels, and let E and S be respectively the languages of specification and its enforcing centralized supervisor as described in Assumption 3.1. Design guard formulas and actions for each process such that $P_{1x}||P_{2x}||\ldots||P_{nx} = S||L$. □

In the next two sections we focus on the logics of control implementation by assuming that channels are ideal, while in Section 3.6 we study the problem when channels are unreliable.

3.4 DECCN solution—special case

In this section we present a solution to a subclass of DECCN problems under the assumptions that a) communication channels are ideal, b) for each $i \in I$, $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$ are singletons and $\Sigma_{uo,i} = \emptyset$, where α_i is called the "significant" event of process i, and c) occurrence of each significant event is counted modulo 2.

Note that under assumption a) variables in X_{ci} are identical to the private variables of other processes, which justifies using x_{jj}^k instead of x_{ij}^k when

needed ($j \neq i$). The controllability and observability of significant events make E controllable and observable, and thus $\mathbf{E}$ may be used as a centralized supervisor in this section. Following the simplifying assumptions b) and c) we use Alternating Bit Protocol (ABP) as a running example, and partially design the protocol as the solution to the corresponding DECCN problem. This simplification leads us to a key observation of the solution approach for general DECCN problems in the next section. The protocol design will be complete in Section 3.6 after the assumption of ideal channels is lifted.

Since the significant event of process i, denoted by α_i, needs to be counted modulo 2, X_{ii} reduces to a singleton, whose only variable x_{ii} is toggled each time α_i occurs:

$$a_i(x_{ii}, \alpha_i) = \overline{x}_{ii} \tag{3.3}$$

With the actions fixed, a solution to DECCN consists of finding guard formulas $g_i(\alpha_i)$, for each $i \in I$.

3.4.1 ABP: Problem formulation in EFSM framework

Alternating Bit Protocol (ABP) [75], [76] is used for reliable transmission of files over half-duplex channels. As shown in Fig. 3.3, two processes $\mathbf{P_1}$ and $\mathbf{P_2}$ communicate over a channel **ch**. Process $\mathbf{P_1}$ fetches a message and sends it to the channel. Then process $\mathbf{P_2}$ receives the message from the channel, and accepts it if it is error-free. The control objective requires that every message fetched by $\mathbf{P_1}$ be accepted by $\mathbf{P_2}$ exactly once. When a transmission error occurs, $\mathbf{P_1}$ should resend its message until it is received error-free and is accepted by $\mathbf{P_2}$.

A schematic of the plant is shown in Fig. 3.4, where a transmission error is denoted by a broken arrow. The system events are defined in Table 3.1.

Table 3.1: System events.

Event	Description
α_1	data fetched by $\mathbf{P_1}$
β^s_{12}	data sent by $\mathbf{P_1}$
β^r_{12}	data received by $\mathbf{P_2}$
β^e_{12}	data received by $\mathbf{P_2}$ erroneous
α_2	data accepted by $\mathbf{P_2}$
β^s_{21}	acknowledgement sent by $\mathbf{P_2}$
β^r_{21}	acknowledgement received by $\mathbf{P_1}$
β^e_{21}	acknowledgement received by $\mathbf{P_1}$ erroneous

Figure 3.5 shows FSM models for sender $\mathbf{P_1}$, receiver $\mathbf{P_2}$ and channel **ch** as well as the specification E of the desired behavior defined as an ordering of events in $\{\alpha_1, \alpha_2\}$. The following is a short description of each FSM in Fig. 3.5.

1. **Sender $\mathbf{P_1}$.** At the initial state, sender $\mathbf{P_1}$ nondeterministically does one of the following:

   (a) It sends a data message to the channel (message could be empty if nothing is yet fetched).

   (b) It fetches a data message and sends it to the channel.

   After receiving acknowledgement from the channel (possibly erroneous),

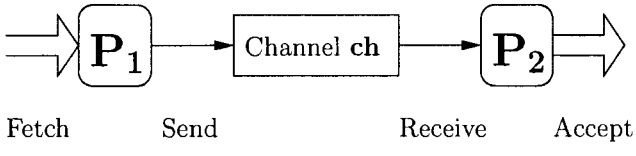


Figure 3.3: Two processes $\mathbf{P_1}$ and $\mathbf{P_2}$ communicating over a channel.

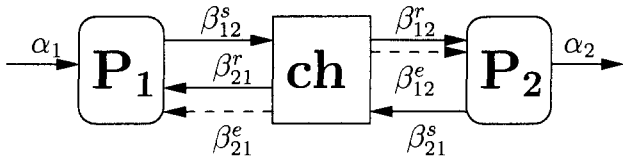


Figure 3.4: Schematic of the plant.

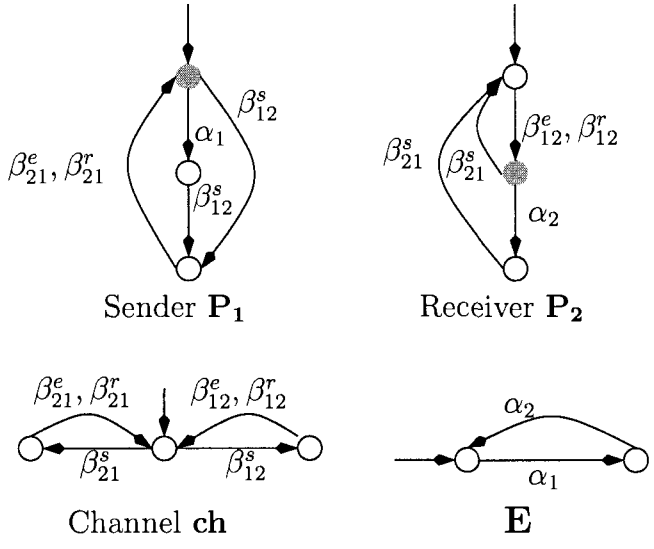


Figure 3.5: Plant FSMs and the requirement specification. $\Sigma_1 = \{\alpha_1, \beta^s_{12}, \beta^e_{21}, \beta^r_{21}\}$ and $\Sigma_2 = \{\alpha_2, \beta^s_{21}, \beta^e_{12}, \beta^r_{12}\}$.

sender $\mathbf{P_1}$ returns to its initial state.

2. **Channel ch**. Any type of message received by the channel from one party (data β^s_{12} or acknowledgement β^s_{21}) will be delivered to the other party (β^r_{12} or β^r_{21}, respectively), or it will get lost or corrupted (β^e_{12} or β^e_{21}, respectively). Note that **ch** is the composition of C_{12} and C_{21}, as defined in the previous section.

3. **Receiver $\mathbf{P_2}$**. After receiving a data message from the channel (possibly erroneous), receiver $\mathbf{P_2}$ nondeterministically does one of the following:

   (a) It sends an acknowledgement to the channel.

(b) It accepts the message, and sends an acknowledgement to the channel.

To make our models simpler we allow slightly more permissive behavior than that of an actual data transmission system. For example, we allow an empty message to be transmitted indefinitely.

It turns out that the plant in Fig. 3.5 violates the specification in two fundamental ways. The following two strings are accepted by the plant but not by the specification:

$$\begin{aligned} &\alpha_{\mathbf{1}}; \beta^s_{12}; \beta^r_{12}; \alpha_{\mathbf{2}}; \beta^s_{21}; \beta^e_{21}; \beta^s_{12}; \beta^r_{12}; \alpha_{\mathbf{2}} \quad \text{and} \\ &\alpha_{\mathbf{1}}; \beta^s_{12}; \beta^e_{12}; \beta^s_{21}; \beta^e_{21}; \alpha_{\mathbf{1}} \end{aligned} \tag{3.4}$$

The well-known ABP [75], [76] provides a standard solution to this control problem. To find a solution in our framework we extend the two processes to $\mathbf{P_{ix}} = (-, X_i, g_i, a_i)$, $i = 1, 2$, where $X_i = \{x_{ii}, x_{ij}\}$, $j = 1, 2$, $j \neq i$, and actions are identity except $a_i(x_{ii}, \alpha_i) = \overline{x_{ii}}$. Note that the assumption of ideal channels allows us to use x_{jj} instead of x_{ij}. The control problem is to find guard formulas g_1 and g_2 such that the projection of $P_{1x}||P_{2x}$ onto $\{\alpha_1, \alpha_2\}$ equals E.

3.4.2 Solution

In supervisory control theory of DES [6], if a given specification is controllable and observable with respect to the plant, there always exists a centralized supervisor which enforces the legal language. In case of distributed DES where each agent has partial observation of the plant behavior, such a controllable global specification is enforceable if and only if it is coobservable with respect to the plant and agents' corresponding observational natural projections [12].

In simple words, coobservability requires that for every two observationally equivalent plant strings, and every event which extends one to a legal string while the other to an illegal string, there exists at least one agent which can disambiguate the strings and inhibit the illegal behavior. The set of decentralized supervisors synthesized in this case need not communicate amongst themselves.

Therefore, if the controllable global specification were coobservable, the solution to DECCN would simply be obtained by separately implementing the supervisory control maps [49] of the computed decentralized supervisors using only their private variables [47]. This case has been discussed in [69] using the FSMs of the plant components for the ABP example where the authors have shown that if the sender model is enriched by incorporating two events associated with the 0/1 status of the ABP's attached bit, the specification will become coobservable with respect to the plant. The same can be said about other defined notions of coobservability with other fusion rules [16]. Thus, in this case no control information need to be communicated over the network to implement the rules of data exchange (i.e. the protocol).

Unfortunately, the specification E in DECCN is *not* in general coobservable. For example, in ABP, $E = (\alpha_1\alpha_2)^*(\epsilon + \alpha_1)$ is not coobservable. To see why, let $s = \alpha_1\beta_{12}^s\beta_{12}^r$ and $s' = \beta_{12}^s\beta_{12}^r$. Note that α_2 is eligible to occur at both s and s', $s\alpha_2$ is legal while $s'\alpha_2$ is illegal, and finally $\pi_2(\pi(s)) = \pi_2(\pi(s'))$. Since process 2 is the only process that can disable α_2, E is not coobservable. In the rest of this chapter we will show how a controllable and observable but non-coobservable specification may be satisfied by communicating information among local processes.

To begin with, we note that under the assumption of ideal channels, one can work with the variable set $X = \{x_{11}, x_{22}, \ldots, x_{nn}\}$. The function

$V : \Sigma^* \rightarrow \mathbb{B}^n$ of Section 3.2 is $V(s) = \big(v(s,x)\big)_{x \in X}$, where for all $i \in I$ we have:

$$v(\epsilon, x_{ii}) = 0 \quad \text{and} \quad v(s\alpha_i, x_{ii}) = \begin{cases} 1 & ; \quad v(s, x_{ii}) = 0 \\ 0 & ; \quad v(s, x_{ii}) = 1 \end{cases} .$$

It turns out that a solution can be found only for a restricted class of problems. To this end, let $\mathbf{E} = (R, \Sigma, \eta, r_0)$ be the *centralized supervisor*'s FSM, and $\mathcal{L}$ denote the set of all labeling maps $l : R \rightarrow pwr(\mathbb{B}^n)$. For $1 \leq i \leq n$ we write a member of $\mathbb{B}^n$ as $\mathbf{v} = (v_i, v_{-i})$, where v_i is the i^{th} element of the n-tuple $\mathbf{v}$, and v_{-i} denotes the $(n-1)$-tuple formed by the remaining elements of $\mathbf{v}$. Define a partial ordering $\preceq$ on $\mathcal{L}$ as follows:

$$\forall l_1, l_2 \in \mathcal{L}. \quad l_1 \preceq l_2 \iff \forall r \in R.\ l_1(r) \subseteq l_2(r) \tag{3.5}$$

It can be verified that $(\mathcal{L}, \preceq)$ is a complete lattice. Let ℓ be the smallest labeling map satisfying the following properties:

1) $\mathbf{0} \in \ell(r_0)$,

2) $\forall r, r' \in R,\ \alpha_i \in \Sigma,\ \mathbf{v} \in \mathbb{B}^n$.

$$\mathbf{v} \in \ell(r) \wedge r' = \eta(r, \alpha_i) \implies (\overline{v_i}, v_{-i}) \in \ell(r'). \tag{3.6}$$

The labeling map ℓ is chosen so that a transition labeled with α_i toggles the i^{th} element of each vector in the state's label. We show by induction that the label of a state reached by s includes the vector of values $V(s)$.

Lemma 3.1 We have $\forall s \in E, r \in R.\ r = \eta(r_0, s) \implies V(s) \in \ell(r)$.

Proof: We prove this lemma by induction on the length of s.

- Base: Let $s = \epsilon$. Then $r_0 = \eta(r_0, s)$, and by definition $V(s) = \mathbf{0} \in \ell(r_0)$.

- Inductive step: For $s \in \Sigma^*$ and $\alpha_i \in \Sigma$ let $s\alpha_i \in E$. Denote $r := \eta(r_0, s)$ and $r' := \eta(r_0, s\alpha_i)$. It follows from the induction assumption that $V(s) \in \ell(r)$. Let $V(s) := (v_i, v_{-i})$. We have:

  $V(s\alpha_i) = (\overline{v_i}, v_{-i}) \in \ell(r')$ (by definition of ℓ) ■

Under certain conditions the labeling map ℓ can in effect encode the states of $\mathbf{E}$: knowing the current value $\mathbf{v} \in \mathbb{B}^n$ of Boolean variables, it is possible to know which state r the centralized supervisor is in by checking whether $\mathbf{v} \in \ell(r)$, as long as $\mathbf{v}$ does not appear in the label of any other state. This idea is formalized in the following definition.

Definition 3.3 Let $\mathbf{E} = (R, \Sigma, \eta, r_0)$ be a centralized supervisor and $\ell : R \rightarrow pwr(\mathbb{B}^n)$ be as defined above. Then $\mathbf{E}$ is said to be *state-independent* with respect to ℓ if

$$\forall r, r' \in R.\ r \neq r' \implies \ell(r) \cap \ell(r') = \emptyset. \qquad \square$$

In other words, in a state-independent centralized supervisor the labels of a pair of distinct states are disjoint. When a centralized supervisor is state-independent, it is possible to uniquely determine its state by knowing the values assumed by the Boolean variables after a legal string; in other words, the inverse of the implication in Lemma 3.1 is true as well.

Lemma 3.2 When $\mathbf{E}$ is state-independent with respect to ℓ we have:

$$\forall s \in E, r \in R.\ r = \eta(r_0, s) \iff V(s) \in \ell(r).$$

Proof ($\Leftarrow$): By contradiction assume for $s \in \Sigma^*$ and $r \in R$ that $V(s) \in \ell(r)$ but $\eta(r_0, s) = r'$ for some $r' \neq r$ in R. It follows from Lemma 3.1 that $V(s) \in \ell(r')$, contradicting the fact that $\mathbf{E}$ is state-independent. ■

The following result states that a solution to the control problem exists when the centralized supervisor is state-independent.

Theorem 3.1 Under the assumption that channels are ideal, DECCN has a solution if $\mathbf{E}$ is state-independent with respect to ℓ.

Proof: Let $\mathcal{L}_i = \bigcup_{r \in R \wedge \eta(r,\alpha_i)!} \ell(r)$ and $g_i(\alpha_i)$ be a Boolean formula that is true for $\mathbf{v} \in \mathbb{B}^n$ if and only if $\mathbf{v} \in \mathcal{L}_i$. By induction we show that for all $s \in \Sigma^*$ we have $s \in \pi(P_{1x}||P_{2x}|| \cdots ||P_{nx})$ if and only if $s \in E$.

Base is trivial since $\mathbf{E}$ and all $\mathbf{P_{ix}}$ are nonempty. For the inductive step let $s\alpha_i \in \pi(P_{1x}||P_{2x}|| \cdots ||P_{nx})$. Since all languages are prefix-closed it follows that $s \in \pi(P_{1x}||P_{2x}|| \cdots ||P_{nx})$ and hence by the induction assumption $s \in E$. Let $r := \eta(r_0, s)$. We have:

$$s\alpha_i \in \pi(P_{1x}||P_{2x}|| \cdots ||P_{nx}) \iff g_i(\alpha_i)(V(s)) = 1$$
$$\iff V(s) \in \mathcal{L}_i \iff \eta(r, \alpha_i)! \qquad \text{(Lem. 3.2)}$$

i.e. $s\alpha_i \in E$. ■

The next 2 examples illustrate the idea.

Example 3.1 Shown in Fig. 3.6 are two centralized supervisors $\mathbf{E_1}$ and $\mathbf{E_2}$ where $n = 3$ and $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $i = 1, 2, 3$. A state r is labeled with all values in the set $\ell(r)$. For example, in $\mathbf{E_1}$, we have $\ell(r_1) = \{(1, 0, 0), (0, 1, 1)\}$ (for brevity a triple (i, j, k) is written as ijk).

The centralized supervisor $\mathbf{E_1}$ is state-independent as for any pair of distinct states (r, r') we have $\ell(r) \cap \ell(r') = \emptyset$. On the other hand, $\mathbf{E_2}$ is clearly not state-independent: we have $\ell(r_1) \cap \ell(r_2) = \ell(r_1) = \ell(r_2)$. ◊

Example 3.2 As shown in Fig. 3.7, the specification (centralized supervisor) $\mathbf{E}$ of our running ABP example is state-independent. We have: $\mathcal{L}_1 = \{00, 11\}$ and $\mathcal{L}_2 = \{01, 10\}$. Thus

$$g_1(\alpha_1) = \overline{x_{11} \oplus x_{22}}, \qquad g_2(\alpha_2) = x_{11} \oplus x_{22}.$$ ◊

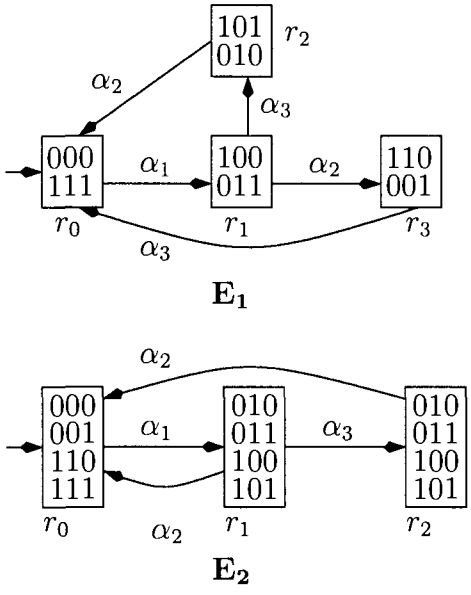


Figure 3.6: The centralized supervisor $\mathbf{E_1}$ is state-independent while $\mathbf{E_2}$ is not.

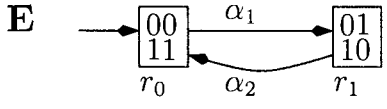


Figure 3.7: The centralized supervisor $\mathbf{E}$ of the ABP is state-independent.

Note that if channels were unreliable then, say, the private variable x_{22} in the guard formula $g_1(\alpha_1)$ must be replaced with its local copy x_{12}. The mechanism by which x_{12} is updated with x_{22} is discussed in Section 3.6.

3.5 Towards the general problem in the presence of ideal channels

In Section 3.4 we used tuples of Booleans to label the states of a centralized supervisor $\mathbf{S} = (R, \Sigma, \eta, r_0)$, and used a fixed updating mechanism in which the occurrence of a significant event α_i toggles the value of the variable x_{ii}, $1 \leq i \leq n$. In general, the class of state-independent centralized supervisors, which can be implemented by communicating decentralized supervisors when channels are ideal, will be widened if one dedicates more bits to count the significant events of processes. The next example illustrates the point.

Example 3.3 As shown in Fig. 3.8-a the centralized supervisor $\mathbf{S}$ is not state-independent with respect to ℓ when events are counted modulo 2 as $\ell(r_0) \cap \ell(r_2) \neq \emptyset$. Now, let us use two binary variables x_{11}^1 and x_{11}^2 to count α_1 as in part (b). The first two occurrences of α_1 increment $x_{11}^1 x_{11}^2$ by one, while its next two occurrences decrement $x_{11}^1 x_{11}^2$ by one back to 00, i.e. the actions count α_1 modulo 3 (as opposed to modulo 2 counting of the previous section). With the new labeling map $\ell' : R \rightarrow pwr(\{0,1,2\} \times \mathbb{B})$, we have $\forall r, r' \in R.\ r \neq r' \Rightarrow \ell'(r) \cap \ell'(r') = \emptyset$, i.e. the centralized supervisor is state-independent with respect to ℓ'. $\Diamond$

Thus, in general, more elegant coding schemes are required to insure that labels are unique, and that each event changes only the value(s) of the process's own private variable(s). With such coding schemes, which may use more than one private variable, there is no reason to limit to one "significant" event per process, and this assumption can be relaxed, too. The following definition characterizes the labeling maps that have the above desired properties.

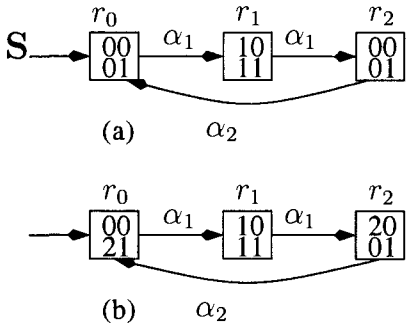


Figure 3.8: The centralized supervisor $\mathbf{S}$, with $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $\Sigma_{uo,i} = \emptyset$, $i = 1, 2$, is not state-independent when one Boolean variable is used to count α_1, while it becomes state-independent when two Boolean variables are used to count α_1.

Remark 3.1 Such coding schemes rely on the observation and encoding of *state changes* in an automaton $\mathbf{S} = (R, \Sigma, \eta, r_0)$ (of the centralized supervisor). Since no state change is observed for events which participate solely in self-loops, i.e. events in $\Sigma_{loop} = \Sigma_{uo} \cup \{\sigma \in \Sigma_o \mid \forall r, r' \in R.\ r' = \eta(r, \sigma) \Rightarrow r = r'\}$, these events might be safely ignored as long as such coding schemes are concerned. However, if an event, say α_i, which is selflooped in one state, say r_1, causes a state change in another state, say r_2, then some provisions should be made to help the coding scheme *observe* all α_i-labeled transitions, including the selfloops. As a remedy, in this case a state $\hat{r}_1$ is added to $\mathbf{S}$ which inherits all the outgoing non-selfloop transitions of r_1, while all selfloop transitions in r_1, which are not labeled by events in Σ_{loop}, are replaced with transitions with the same labels from r_1 to $\hat{r}_1$ and vice versa. By following this procedure, all selfloops in a state that cause state changes in other states are made *observable* to the coding scheme. Note that in the worst case, the state size of the new automaton (which is still deterministic) would be twice as large as the original. In what follows, the coding schemes are always assumed to be applied to automata with possible selfloops labeled by events in Σ_{loop} only. Moreover, we

assume, without loss of generality, that in the next examples $\Sigma_{uo} = \emptyset$. □

Definition 3.4 Let $\mathbf{S} = (R, \Sigma, \eta, r_0)$ be a centralized supervisor modified if necessary as in Remark 3.1. An Agent-wise Labeling Map (ALM) is a map $\ell : R \to pwr(\mathbb{N}^n)$ with the following properties:

1. $\mathbf{0} \in \ell(r_0)$;

2. $\forall r, r' \in R.\ r \neq r' \Rightarrow \ell(r) \cap \ell(r') = \emptyset$ (labels are pairwise disjoint);

3. $\forall r, r' \in R, r \neq r',\ \forall \sigma \in \Sigma_o,\ \forall \mathbf{v} \in \mathbb{N}^n.$
   $v \in \ell(r) \wedge r' = \eta(r, \sigma) \Longrightarrow \exists\ \mathbf{v}' \in \mathbb{N}^n.\ \mathbf{v}' \in \ell(r')$
   $\wedge\ [\forall i \in I_o(\sigma).\ v_i \neq v'_i]\ \wedge\ [\forall j \in I \setminus I_o(\sigma).\ v_j = v'_j].$

We call an ALM *finite* if its image is a finite set. □

Remark 3.2 By the second property $\mathbf{S}$ is state-independent with respect to an ALM. □

Remark 3.3 The last property implies that an ALM neither limits the number of events participating from each process in $\mathbf{S}$, nor makes any distinction among them. □

To show the existence of a finite ALM, we need the following definitions.

Definition 3.5 Consider a centralized supervisor $\mathbf{S} = (R, \Sigma, \eta, r_0)$ and an index set I. Two distinct states $r, r' \in R$ are called *I-connected* if for all $i \in I$ there exists a $\sigma \in \Sigma_{o,i}$ such that $r' = \eta(r, \sigma)$. The automaton $\mathbf{S}$ is I-connected if every pair of distinct states in $\mathbf{S}$ are I-connected. □

Figure 3.9 illustrates an example of an I-connected automaton $\mathbf{S}$.

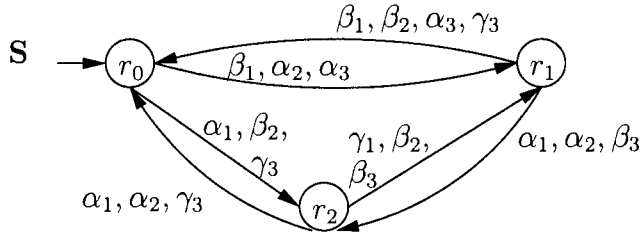
Definition 3.6 Let $\mathbf{v}, \mathbf{v}' \in \mathbb{N}^n$ be labels and $i \in I$. We say $\mathbf{v}$ is an *i-sibling* of $\mathbf{v}'$ if $v_i \neq v'_i$ and $v_{-i} = v'_{-i}$. □

Theorem 3.2 There exists an efficiently computable[4] finite ALM for every centralized supervisor $\mathbf{S} = (R, \Sigma, \eta, r_0)$, where $\mathbf{S}$ is modified if necessary as in Remark 3.1.

Proof: The proof is done by establishing a bijection between constructing an ALM for $\mathbf{S}$ and another problem described below. Assume that $R = \{r_0, r_1, \ldots, r_{m-1}\}$ and define $J = \{0, \ldots, m-1\}$. Notice that since all events in Σ_o are observable, each transition's event in $\mathbf{S}$ belongs to at least one $\Sigma_{o,i}$, $i \in I$.

We make two assumptions which are relaxed later in the proof: (i) that $\Sigma_{o,i}$'s are mutually disjoint and (ii) that $\mathbf{S}$ is I-connected. By Definition 3.4, constructing an ALM for $\mathbf{S}$ is equivalent to finding m *mutually disjoint* sets $L_j = \ell(r_j)$, $j \in J$, each consisting of labels $\mathbf{v} \in \mathbb{N}^n$ satisfying Items 1 and 3. Item 1 implies that $\mathbf{0} \in L_0$. Under assumptions (i) and (ii) mentioned above, since for each i there is a transition from every state r_k to every other state, Item 3 of this definition requires that each tuple $\mathbf{v} \in L_k$ have an i-sibling in every other state, for a total of $m-1$ distinct i-siblings (since label sets must

[4]The construction can be done in polynomial time, as shown in the proof.



$\Sigma_{o,1} = \{\alpha_1, \beta_1, \gamma_1\}$, $\Sigma_{o,2} = \{\alpha_2, \beta_2\}$, $\Sigma_{o,3} = \{\alpha_3, \beta_3, \gamma_3\}$

Figure 3.9: An example of an I-connected automaton.

be disjoint by Item 2 of Definition 3.4).

Graphically, each n-tuple label $\mathbf{v}$ may be identified with a point in $\mathbb{N}^n$. For the ease of representation, a point $\mathbf{v} \in \mathbb{N}^n$ is marked by one of m distinct *objects*, each corresponding to a state of $\mathbf{S}$, to indicate the membership of $\mathbf{v}$ to the label of a state. For instance, in Fig. 3.10, the label $(0, 2, 0)$ is marked by a square, indicating its membership to the label set of state r_2. Note that for any $\mathbf{v} \in \mathbb{N}^n$, all i-siblings of $\mathbf{v}$ must be located on a straight line parallel to $i \in I$ axis. As argued before, to have i-siblings of $\mathbf{v}$ in all other states, along every dimension $i \in I$ there must exist exactly one copy of each object, for a total of m distinct objects. Accordingly, one arrangement would be to construct an m by m hypercube in $\mathbb{N}^n$, one corner of which is located at the origin, and in its every dimension $i \in I$ there exists exactly one copy of each of the m distinct objects, i.e. m i-sibling labels, each belonging to one L_j, $j \in J$. Such an arrangement is called a *Latin hypercube of side* m, and can be efficiently computed [77, 78]; a simple example is shown in Fig. 3.10.

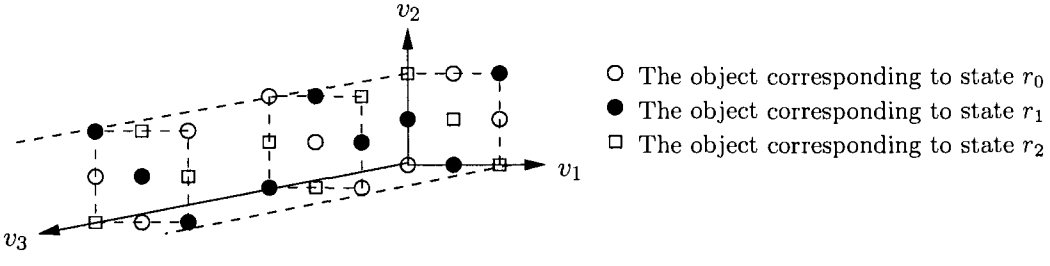


Figure 3.10: A Latin cube with $n = 3$ (the number of axes) and $m = 3$ (the number of objects): There exists exactly one copy of each object in every direction.

The above argument reveals that there exists a finite ALM for a given $\mathbf{S}$ under the assumptions (i) and (ii). Assumption (ii) creates a worst-case scenario; an ALM for $\mathbf{S}$ is also an ALM for $\mathbf{S}'$ which is identical to $\mathbf{S}$ with

some transitions removed (thus (ii) may no longer hold).

Let us now assume that assumption (i) is relaxed, i.e. there is an event σ for which $|I_o(\sigma)| > 1$. Item 3 of Definition 3.4 thus requires that the occurrence of σ move the current point in the Latin hypercube to a point whose every coordinates in $I_o(\sigma)$ changes, while others in $I \setminus I_o(\sigma)$ remain unchanged. Such a point always exists since there is exactly one copy of each of the m distinct objects in each dimension of the Latin hypercube, and therefore, there always exists a path which starts from the current point, each time moves along one of the dimensions specified by $I_o(\sigma)$ in some specific order, and ends up in the required point in the hypercube. Hence the proof remains valid if all the assumptions are lifted. ■

Remark 3.4 It is interesting to note that, in general, the hypercube of m^n labels, with exactly m copies of each object along each dimension, provides an upper bound for the number of labels required by an ALM, in the sense that it is possible to find an ALM with a smaller image size if assumption (ii) is relaxed. On the other hand, it provides the *minimum* number of the required labels in the worst-case scenario where for every pair of automaton's states and for each i, some events in $\Sigma_{o,i}$ trigger a move from one state of the pair to the other. □

The next example illustrates the procedure mentioned in the above proof and Remark 3.1.

Example 3.4 Consider the centralized supervisor **S** in Fig. 3.4-a and the sub-alphabets $\Sigma_{c,1} = \Sigma_{o,1} = \{\alpha, \alpha_1, \beta_1\}$ and $\Sigma_{c,2} = \Sigma_{o,2} = \{\alpha, \alpha_2, \beta_2\}$. Following Remark 3.1, we examine selfloop transitions in **S** and notice that β_1 causes no state change and can thus be safely ignored. On the other hand, α_1 and α_2 cause state change from r_0 to r_1, and therefore they are replaced by transitions

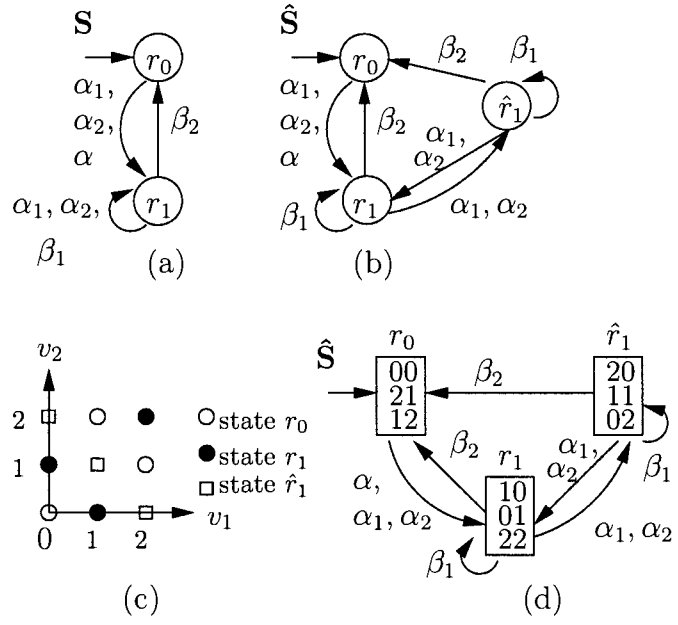


Figure 3.11: (a) A centralized supervisor and (b) its unfolded version. (c) Graphical representation of a finite ALM. (d) The encoded supervisor.

between r_1 and the new state $\hat{r}_1$, which inherits its outgoing transitions from r_1. For the new automaton $\hat{\mathbf{S}}$ in part (b), which has 3 states r_0, r_1, and $\hat{r}_1$, by the proof of Theorem 3.2 a finite ALM may be found using a Latin square of side 3. Such an arrangement is shown in Fig. 3.4-c simply by associating the horizontal and vertical axes with agents 1 and 2, respectively, and placing three objects, each representative of one state, in the first row, and shifting this row one unit to the left each time to create the other rows. By Item 1 of Definition 3.4, point $(0,0)$ is assigned to r_0. We notice that there are transitions from the state r_0 to r_1 which are labeled with events $\alpha_1 \in \Sigma_{o,1}$, $\alpha_2 \in \Sigma_{o,2}$, and the common event α. Thus, corresponding to each vector of values in $\ell(r_0)$ (e.g. $(0,0)$), there are a 1-sibling (e.g. $(1,0)$), a 2-sibling (e.g. $(0,1)$), and a vector differing in *both* coordinates (e.g. $(2,2)$) in $\ell(r_1)$. Similar observations can be made for the other states and their labels. $\Diamond$

Definition 3.7 Let an ALM be employed for labeling the states of a centralized supervisor $\mathbf{S} = (R, \Sigma, \eta, r_0)$ and denote by V_i the set of numbers used by each agent for labeling; that is,

$$\forall i \in I. \quad V_i := \{v_i \in \mathbb{N} | \ \exists r \in R, v_{-i} \in \mathbb{N}^{n-1}. \ (v_i, v_{-i}) \in \ell(r)\} \tag{3.7}$$

The set of private Boolean variables with which each agent needs to implement its labels is denoted by $X_{ii} = \{x_{ii}^k | k \in \{1, \ldots, \lceil log_2 |V_i| \rceil\}\}$. □

In general the guard formula of an event α_i is a function of all of agent i's variables, i.e. $g_i(\alpha_i) = h_i(X_{ii}, X_{ci})$. Also, the action associated with the private variable x_{ii}^k of process i and an arbitrary event of the process, say α_i, is not *a priori* fixed and is a function of all private and copy variables of process i, i.e. $a_i(x_{ii}^k, \alpha_i) = f_{i,k}(X_{ii}, X_{ci})$. The function $f_{i,k}$ must be designed to implement the desired labeling map as part of the solution to the decentralized control implementation problem. The next example illustrates this point.

Example 3.5 For the centralized supervisor in Fig. 3.8-b assume that all channels are ideal. Then using two (one) private variables for process 1 (2) to encode the states as $(x_{11}^1 x_{11}^2, x_{22}^1)$, the non-identity actions can be calculated as: $a_1(x_{11}^1, \alpha_1) = x_{11}^2 \overline{x_{22}^1}$, $a_1(x_{11}^2, \alpha_1) = \overline{x_{11}^2}$ and $a_2(x_{22}^1, \alpha_2) = \overline{x_{22}^1}$. The guard formulas $g_1(\alpha_1) = \overline{x_{11}^1 \oplus x_{22}^1} + x_{11}^2$ and $g_2(\alpha_2) = x_{11}^1 \overline{x_{22}^1} + \overline{x_{11}^1} x_{22}^1 \overline{x_{11}^2}$ insure that α_1 is enabled only in r_0 and r_1, while α_2 is enabled only in r_2. (Calculation of guards and actions are detailed in [49].) ◊

As is evident from the above example, in general, both guards and actions depend on the values of (copies of) private variables of other processes. When $g_i(\alpha_i) = h_i(X_{ii}, X_{ci})$, communication is needed to update the copies in X_{ci} to insure that the right control decision is made ("communication for control"). When $a_i(x_{ii}^k, \alpha_i) = f_{i,k}(X_{ii}, X_{ci})$, communication is needed to update

the copies in X_{ci} to insure that the variables in X_{ii} are properly updated; in other words, to update an agent's estimate of the centralized supervisor's state ("communication for observation"). Thus, given a controllable, observable, but non-coobservable specification and its enforcing centralized supervisor, in a network with ideal channels where local copies of agents' private variables can be updated instantaneously, the communication protocol is specified by the following entities; The *control decision* of each agent, i.e. guards, and the communications for control and/or observation amongst agents. In this sense, the protocol design is equivalent to SCDS where each decentralized supervisor makes control decisions based on its own observation of the plant behavior and the received communications from other supervisors. Note that in the EFSM formalism supervisors do not exist as separate entities; they are implemented by guards and actions of the processes' EFSMs. As such, communication takes place between the processes themselves.

While in general finding answers to questions regarding ordering and minimality of communication might be a difficult challenge, in what follows we restrict EFSM models so that they do not need "communication for observation," and identify a class of centralized supervisors that can be implemented by such EFSMs.

Definition 3.8 [47] We say we have *independent actions* when

$$\forall i \in I,\ \forall k \in \mathbb{N},\ \forall x_{ii}^k \in X_{ii},\ \forall \alpha_i \in \Sigma_{o,i}.$$
$$a_i(x_{ii}^k, \alpha_i) = f_{i,k}(X_{ii}). \qquad \square$$

The following Lemma identifies ALMs that yield independent actions.

Lemma 3.3 The actions associated with an ALM for the centralized supervisor **S** are independent if and only if the ALM assigns the same component labels to the states of **S** which are reached by strings that are observationally equivalent for that component.

Proof: Please refer to [79]. ■

It turns out that EFSMs with independent actions can implement a centralized supervisor *only if* its language satisfies a weak version of "joint observability" property [31]. We show this point next.

Definition 3.9 [31] S is *jointly observable* with respect to $\pi(L)$ and $(\Sigma_{o,1}, \ldots, \Sigma_{o,n})$ if and only if

$$\forall \rho \in S,\ \forall \rho' \in \pi(L) \setminus S,\ \exists i \in I.\ \ \pi_i(\rho) \neq \pi_i(\rho').$$ ■

In words, joint observability requires that for every two lookalike legal-illegal sequences in the plant's behavior, there exists at least one supervisor which can tell them apart. However, in control problems one always cares about the first instance at which the legal behavior is violated, and any subsequent evolution of illegal behavior is of no interest (as it is to be prevented by a controller). From this viewpoint joint observability is too strong a property for control applications, and therefore below we introduce a weaker notion which requires the existence of a supervisor which can distinguish two legal strings when an event extends one to a legal string while extends the other to an illegal string.

Definition 3.10 [47] S is *weakly jointly observable* with respect to $\pi(L)$ and $(\Sigma_{o,1}, \ldots, \Sigma_{o,n})$ if and only if

$$\forall s, s' \in S,\ \forall \sigma \in \Sigma.$$

$$s\sigma \in S \wedge s'\sigma \in \pi(L) \setminus S \ \Rightarrow\ \exists i \in I.\ \ \pi_i(s) \neq \pi_i(s').$$ □

Lemma 3.4 [47] Joint observability implies weak joint observability.

Proof: Choose any $s, s' \in E$, and $\sigma \in \Sigma$ such that $s\sigma \in E\ \wedge\ s'\sigma \in L \setminus E$. Take $\rho = s\sigma$ and $\rho' = s'\sigma$. By joint observability we know that there exists

$i \in \{1, \ldots, n\}$ such that

$$\begin{aligned} & \pi_i(\rho) \neq \pi_i(\rho') \\ & \Rightarrow \pi_i(s\sigma) \neq \pi_i(s'\sigma) \\ & \Rightarrow \pi_i(s)\pi_i(\sigma) \neq \pi_i(s')\pi_i(\sigma) \\ & \Rightarrow \pi_i(s) \neq \pi_i(s'). \end{aligned}$$

■

Lemma 3.5 [47] A language S is weakly jointly observable with respect to $\pi(L)$ and $(\Sigma_{o,1}, \ldots, \Sigma_{o,n})$ if there exists an ALM for $\mathbf{S}$ such that the associated actions are independent.

Proof: Assume that there exist independent actions and let $s, s' \in E$ and $\sigma \in \Sigma$ be such that $s\sigma \in E$ and $s'\sigma \in L \setminus E$. Write the states reached by s and s' as r and r', respectively, so that there exist $\mathbf{v}, \mathbf{v}' \in \mathbb{N}^n$ such that $\mathbf{v} = (v_i, v_{-i}) \in \ell(r)$ and $\mathbf{v}' = (v'_i, v'_{-i}) \in \ell(r')$ as in Definition 3.4. If E is not weakly joint observable, then:

$\forall i \in \{1, \ldots, n\}.\ \pi_i(s) = \pi_i(s')$

$\Rightarrow \forall i \in \{1, \ldots, n\}.\ v_i = v'_i$ (Item 3, Defn. 3.4)

$\Rightarrow \mathbf{v} = \mathbf{v}'$

$\Rightarrow r = r'$ (Only if part of Lemma 3.3)

which is a contradiction. ■

The above result states a structural property for the language of the centralized supervisor without which no independent actions may be derived regardless of the choice of ALM. However, for an action to be independent of other agents' variables, it is necessary that for each agent the component labels assigned by an ALM be such that any changes in their values depend solely on the current values of the component labels. In simple words, the

choice of the ALM should be such that updating the labels of every agent is a *function* of its own label values. The next example illustrates these points.

Example 3.6 It can be verified that S' in Fig. 3.12-a is not weakly jointly observable. As a counterexample, let $s = \alpha_1\alpha_2$, $s' = \alpha_2\alpha_1$ and the dashed arrow represent the plant's illegal move. Then while $s\alpha_1$ is legal and $s'\alpha_1$ is illegal, we have $\pi_i(s) = \pi_i(s') = \alpha_i$ for $i = 1, 2$. Therefore, by the previous lemma a set of independent actions cannot be found to implement $\mathbf{S}'$ regardless of the choice of ALM.

For the weakly jointly observable S in Fig. 3.8-b, the labeling map ℓ' used in Example 3.5 does not yield independent actions: for agent 1, the component label 1 in state r_1 is mapped sometimes to 2 and sometimes to 0, depending on the label assigned by agent 2, so that its action cannot be expressed as a function on its set of labels $\{0, 1, 2\}$, but as a function on the cartesian product of both agents' labels, i.e. $\{0, 1, 2\} \times \{0, 1\}$, which makes the actions dependent (recall the expression for $a_1(x_{11}^1, \alpha_1)$ in Example 3.5). Now, let us apply the ALM ℓ'' of Fig. 3.12-b to the same specification; note that the specification remains state-independent with respect to ℓ''. Observe that under the new labeling every component label in the set $\{0, 1, 2, 3\}$ for agent 1 is uniquely mapped to an element in the same set. In this case the set of Boolean variables and the last two actions remain as in Example 3.5, while the first action becomes $a_1(x_{11}^1, \alpha_1) = x_{11}^1 \oplus x_{11}^2$, hence independent actions are achieved. $\Diamond$

When actions are independent, as in the ABP example, the solution of SCDS enjoys the following property. We first need to define "minimality" of Boolean functions.

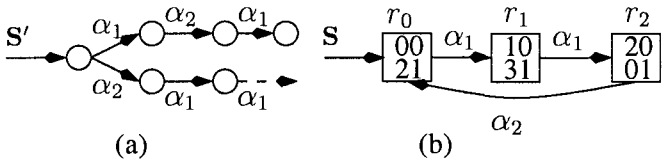


Figure 3.12: $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $i = 1, 2$. (a) A language that is not weakly jointly observable. (b) A state-independent centralized supervisor yielding independent actions.

Definition 3.11 We say a Boolean formula is in a *reduced form* if it contains a minimal number of Boolean variables after possibly utilizing *don't care* conditions [73] . □

Remark 3.5 Notice that when computing reduced forms for guards and actions, one should take into account the fact that in the end it is desired to have minimal exchange of information among the supervisors. As a result, whenever there is more than one reduced form for a Boolean formula or function, the one(s) which share more common variables with other formulas and functions are selected. This issue is outside the scope of the present work. □

Lemma 3.6 Let E be a global controllable, observable, but non-coobservable specification and **S** be the centralized supervisor enforcing E, whose associated actions are independent. Then E can be implemented over a network of ideal channels if a number of bits are communicated in order to reevaluate guards, while no communication is needed for reevaluating the actions. Moreover, this number may be chosen minimally, in the sense of Definition 3.11, up to the ALM used to label the states of **S**.

Proof: Similar to the proof of Theorem 3.1, by the state-independency of **S** with respect to the ALM (Item 3, Definition 3.4), the formulas representing the guards can be computed as functions of the private and copy variables,

i.e.:

$$\forall i \in I,\ \forall \alpha_i \in \Sigma_i.\ g_i(\alpha_i) = h_i(X_{ii}, X_{ci}). \tag{3.8}$$

Thus, to apply control over its corresponding event α_i, agent i needs to receive only the updated values of the copy variables in X_{ci} (i.e. communication for control). Following the fact that the image of the ALM is finite, only a *finite* number $|X_{ci}|$ of bits must be received (instantaneously, under the assumption of ideal channels) in order to make the right control decisions. On the other hand, the independency of actions implies that every such agent updates its private variables in X_{ii} based on its own observation of the plant behavior (Lemma 3.3), and therefore no communication for observation is required.

Upon computing one of the (possibly several) reduced forms of the guard formulas (see Definition 3.11), a minimal number of copy variables in X_{ci} are needed for communication. We notice that there might exist more than one ALM to label the states of $\mathbf{S}$, each using $|X_{ii}|$ private variables for agent i. As a result, the minimality is up to the ALM used in labeling the states of $\mathbf{S}$. ■

In conclusion, when channels are ideal and actions are independent, a protocol for a non-coobservable specification simply requires the communication of a (minimal) number of bits for agents' control purpose of reevaluating their guard formulas.

Example 3.7 For $\mathbf{S}$ in Fig. 3.12-b, we have $g_1(\alpha_1) = \overline{x^1_{11} \oplus x^1_{22}}$ and $g_2(\alpha_2) = x^1_{11} \oplus x^1_{22}$. Therefore, the *protocol* requires process 1 (2) to attach to each data message it sends the value of x^1_{11} (x^1_{22}, respectively). Notice that value of x^2_{11} need not be communicated. ◊

Remark 3.6 It is worth comparing our ALM-based approach to the *estimator structure* of [1] and *possible worlds* of [24]. The following observations can be made about our approach versus those of [1] and [24].

- While an ALM can be found for any deterministic automaton of a centralized supervisor (after a possible modification as explained in Remark 3.1), the other two approaches have been applied to reachability trees only, and their applicability to general automata containing loops is not claimed nor does it seem obvious.

- An ALM adopts an agent-wise viewpoint in labeling the states of a centralized supervisor, while the other two approaches rely on a global labeling for the states and then gathering the lookalike state labels for each agent as a set of state estimates [1] or possible worlds [24]. Since in decentralized control the supervisors view the plant's behavior subject to their partial observations, the ALM labeling provides a natural formulation for the distribution of information within the network. Moreover, the ALM approach views the labels as an integral part of the implementation of supervisor's commands, while in the other two approaches labeling is an auxiliary tool and the viewpoint is quite abstract.

- The final rules for communication in the other two approaches are always translated in terms of communicating the state estimates or possible worlds, while in the ALM approach (more specifically, in EFSM framework) everything is expressed with respect to bits of information[5] used by each local supervisor to encode the states of a global supervisor. Observe that the latter serves to define a practical measure, especially when issues such as minimality of communication are studied.

- Another advantage of the EFSM formalism is its compact representation of the supervisors' commands and observations using Boolean formulas

[5]When talking about "bits of information," we mean "binary digits," not bits in the sense of information theory.

and functions, while the other two approaches make use of the supervisors' automata.

- The works in [1] and [24] adopt "the latest safe point" and "as early as possible" communication policies, respectively, to deal with the issue of "when" to communicate. Although this issue is not explicitly addressed in our work and the focus is on the logical aspects of protocol design to reveal the informational dependencies among every two supervisors (of two distinct processes), it is implicit that communication takes place whenever necessary, in other words, when guards or actions need to be reevaluated.

- Moreover, the case of unreliable channels, which is the subject of the last section, is not studied in the aforementioned papers.

Noting the similarities between [1] and [24], where either state estimates or possible worlds are communicated, through the following example, taken from [1], we illustrate our formulation and solution and that of [1] for a simple problem. □

Example 3.8 Consider the centralized supervisor **S** in Fig. 3.13-a where $\Sigma_{o,1} = \{\alpha_1, \beta_1, \gamma_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$ and event γ_1 is controllable by the first supervisor. Part (b) shows the labels assigned to the states by an ALM. Representing the component labels $\{0,1,2,3\}$ and $\{0,1,2\}$ of, respectively, the first and the second supervisors, using binary variables $x^1_{11}x^2_{11}$ and $x^1_{22}x^2_{22}$, the guard associated with γ_1 would be $g_1(\gamma_1) = x^2_{22}$ and the actions may be computed as $a_1(x^1_{11}, \alpha_1) = 0$, $a_1(x^2_{11}, \alpha_1) = 1$, $a_1(x^1_{11}, \beta_1) = 1$, $a_1(x^2_{11}, \beta_1) = 0$, $a_1(x^1_{11}, \gamma_1) = 1$, $a_1(x^2_{11}, \gamma_1) = 1$, $a_2(x^1_{22}, \alpha_2) = x^2_{11}$, and $a_2(x^2_{22}, \alpha_2) = \overline{x^2_{11}}$. Therefore, by the time supervisor 1 wants to make its control decision for γ_1

at states 6 or 7, it should have received the updated value of x_{22}^2 from supervisor 2 (i.e. *communication for control*). However, the last time x_{22}^2 is updated is upon the occurrence of α_2, for which supervisor 2 needs to receive the most recent value of x_{11}^2 (i.e. *communication for observation*). This latter variable is updated solely based on the local observation of supervisor 1, so no more communication is required. As a result, our solution requires that a) whenever α_2 occurs, supervisor 2 receive 1 bit to reevaluate its action(s) and b) before making a decision on disabling γ_1, supervisor 1 receive 1 bit to reevaluate its guard.

The solution in [1] relies on first a global labeling of states of $\mathbf{S}$ as in part (a) of the figure, and second on the estimator structure in part (c). Every state of the estimator structure consists of a quadruple whose top and bottom elements correspond to the event occurred and the state reached in $\mathbf{S}$, respectively. The second and third elements are, respectively, the state estimates made by supervisors 1 and 2 after the occurrence of events. Computing the latest safe point as state 5, the authors in [1] come up with the communication policy which prescribes that supervisor 2 communicate its state estimate $\{2, 5\}$ at the latest safe point, and as supervisor 2 cannot tell apart state 5 from state 2, it does the same communication at state 2 as well.

Accordingly, the following observations can be made: a) The content of communication consists of 2 bits in our formulation and 2 states (or their labels) in the formulation in [1], which, in general, consists of more than two bits (especially since labels are global). b) Also, our formulation provides a more detailed treatment of the (qualitative) time of each communication. However, we would like to point out that this example is not an exhaustive comparison between the two methods.

Notice that while our approach is capable of handling any *arbitrary* finite

automaton **S** with equal ease, this simple example serves to illustrate how naturally the purpose of communication (observation v. control) manifests itself in the designed protocol. ◊

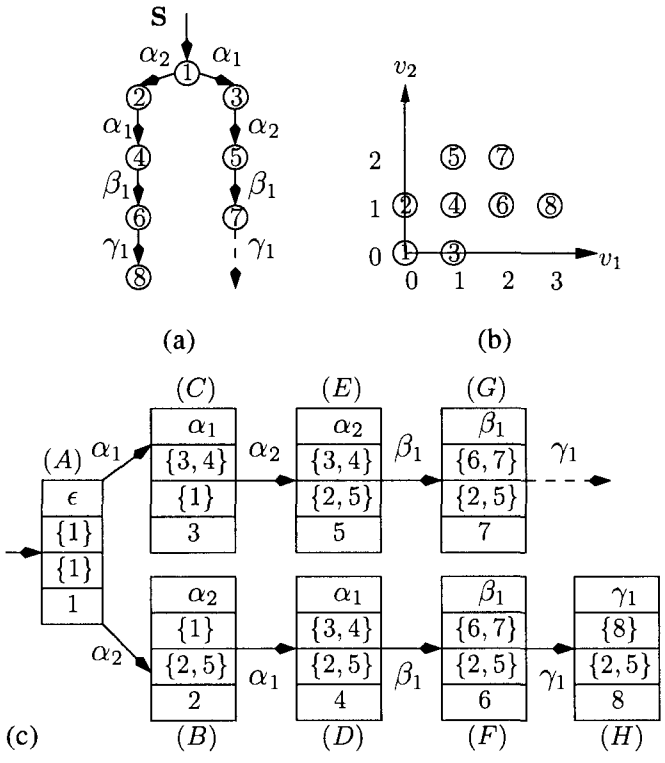


Figure 3.13: (a) A centralized supervisor and (b) its labels assigned by an ALM. (c) The estimator structure (without communication) for part (a) (reprinted from Fig. 3 in [1]).

3.6 DECCN solution—unreliable channels

This section studies the effects of unreliable channels on implementation of a centralized supervisor. To simplify the study of such effects, we keep assumptions b) and c) of Section 3.4. However, the results can be generalized to the case of Section 3.5 in an appropriate manner.

When process $\mathbf{P_{ix}}$ is connected to process $\mathbf{P_{jx}}$ through an unreliable channel, we assume that process $\mathbf{P_{ix}}$ sends the values of its variables to the channel infinitely often (event β_{ij}^s). Although the transmission could fail several times (event β_{ij}^e), we assume that the channel is *weakly fair*, in the sense that the control information is received error-free by process $\mathbf{P_{jx}}$ (event β_{ij}^r) infinitely often. Thus, the copies of variables in $\mathbf{P_{jx}}$ are updated with the corresponding values in $\mathbf{P_{ix}}$ infinitely often, but as a result of possible transmission errors the eventual update of copies in $\mathbf{P_{jx}}$ may experience unbounded delay. Unfortunately delay in a communication network makes it nearly impossible to locally implement any centralized supervisor which offers nondeterministic[6] choice among events of several processes. To see this, suppose at a state of a centralized supervisor both α_i and α_j are enabled, and the occurrence of one entails disabling the other. Then, say, if α_i occurs first, α_j remains enabled until process $\mathbf{P_{jx}}$ is informed that α_i has occurred in $\mathbf{P_{ix}}$ (in our proposed framework, this means that x_{ji} is updated with the value of x_{ii}). Until then, α_j may occur, contradicting the behavior of the centralized supervisor. The following example further illustrates the problem.

Example 3.9 Assume that we would like to implement the centralized supervisor shown in Figure 3.14, where $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, $i \in I = \{1, 2, 3\}$. When channels are ideal this could be achieved by introducing Boolean variables x_{ii}, $i \in I$, where x_{ii} is toggled upon the occurrence of α_i, i.e. $a_i(x_{ii}, \alpha_i) = \overline{x_{ii}}$, while guard formulas are found to be $g_1(\alpha_1) = \overline{x_{11} \oplus x_{22} \oplus x_{33}}$ and $g_2(\alpha_2) = g_3(\alpha_3) = x_{11} \oplus x_{22} \oplus x_{33}$.

In the presence of unreliable channels, process i keeps local copies of private variables of processes j and k, denoted respectively by x_{ij} and x_{ik},

[6]Here "nondeterminism" refers to the existence of two or more paths between two processes. This is clearly different from the notion of "nondeterminism" in automata theory.

which are updated with the values of private variables x_{jj} and x_{kk} whenever an error-free communication from the corresponding process is received ($i, j, k \in I,\ i \neq j,\ i \neq k,\ j \neq k$). Thus, $X_i = \{x_{ii}, x_{ij}, x_{ik}\}$. Accordingly, the guard formulas are evaluated "locally," i.e.: $g_1(\alpha_1) = \overline{x_{11} \oplus x_{12} \oplus x_{13}}$, $g_2(\alpha_2) = x_{21} \oplus x_{22} \oplus x_{23}$ and $g_3(\alpha_3) = x_{31} \oplus x_{32} \oplus x_{33}$.

Initially, all variables are zero; thus α_1 is enabled (since $g_1(\alpha_1) = 1$) while α_2 and α_3 are disabled (since $g_2(\alpha_2) = g_3(\alpha_3) = 0$), as required at the initial state of the centralized supervisor. Assume that α_1 is taken, and the values of x_{21} and x_{31} are updated with the new value of $x_{11}(= 1)$. At this point, $g_2(\alpha_2) = g_3(\alpha_3) = 1$ while $g_1(\alpha_1) = 0$, as required at state 'b' of the centralized supervisor. Next, assume that α_2 is taken and thus the value of x_{22} is toggled to 1. As a result, $g_2(\alpha_2) = 0$, as required at state 'a' of the centralized supervisor. However, α_3 remains enabled (i.e. $g_3(\alpha_3) = 1$) until the value of x_{32} is updated with the new value of x_{22} by a successful communication from process $\mathbf{P_{2x}}$ to process $\mathbf{P_{3x}}$. Until then, α_3 may be taken, and thus our attempt to implement the centralized supervisor fails. Intuitively, for decentralized supervisory control to work, processes $\mathbf{P_{2x}}$ and $\mathbf{P_{3x}}$ must be *immediately* notified of the occurrence of the other process's significant event.

$\Diamond$

The problem is further complicated when the network itself is *nondeterministic*, i.e. there are two or more paths from one process to another. Suppose, for instance, that the centralized supervisor requires α_j to happen

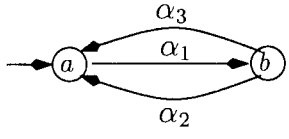


Figure 3.14: The centralized supervisor of Example 3.9.

after α_i, and that there are two paths $\wp_a$ and $\wp_b$ from $\mathbf{P_{ix}}$ to $\mathbf{P_{jx}}$. Assume that $\mathbf{P_{jx}}$ enables α_j after it is informed through $\wp_a$ that α_i has occurred. After α_j is taken it should be disabled by $\mathbf{P_{jx}}$ until the next time α_i occurs. Now assume that process $\mathbf{P_{jx}}$ is informed through $\wp_b$ that α_i occurred 0 times modulo 2 (note that counting is performed modulo $N = 2$; more elaborate examples can be devised for arbitrary finite N). Then $\mathbf{P_{jx}}$ does not know for certain what to make of the information just received: if α_i occurred 0 times, then the information is outdated (i.e. the communication was initiated by $\mathbf{P_{ix}}$ before α_i was taken) and must be ignored. In this case, α_j should remain disabled. On the other hand, process $\mathbf{P_{jx}}$ needs to re-enable α_j if it is informed that α_i has occurred for the second time (note that $2 \equiv 0 \mod 2$).

We conclude that the class of specifications satisfiable over unreliable communication channels is severely restricted. One can hope for a solution to DECCN when the network is deterministic in the sense defined above, and the centralized supervisor enables a *single* event in its every state. In particular, the following result offers a solution when the specification requires a linear ordering among significant events. First we define a deterministic network.

Definition 3.12 Let $\mathcal{N}$ be a system consisting of n communicating parallel processes which are connected through a strongly connected network of potentially unreliable channels. $\mathcal{N}$ is *deterministic* if for every i and j, $i \neq j$, there is a unique path from $\mathbf{P}_i$ to $\mathbf{P}_j$. □

Theorem 3.3 Let $(k_1, k_2, \ldots, k_n)$ be a permutation of $\{1, 2, \ldots, n\}$. If $\mathcal{N}$ is deterministic, the controllable specification $E = \overline{(\alpha_{k_1}\alpha_{k_2}\ldots\alpha_{k_n})^*}$, with $\Sigma_{o,i} = \Sigma_{c,i} = \{\alpha_i\}$, can be satisfied by guarding α_{k_i} with $g_{k_i}(\alpha_{k_i})$, where:

$$g_{k_i}(\alpha_{k_i}) = \begin{cases} \overline{x_{k_1k_1} \oplus x_{k_1k_n}} & ; \quad i = 1 \\ x_{k_ik_i} \oplus x_{k_ik_{i-1}} & ; \quad 2 \leq i \leq n \end{cases}$$

Proof. Since E is controllable and is defined over an observable alphabet, $\mathbf{E}$ can be used as a centralized supervisor enforcing E. Without loss of generality assume that $k_i = i$. We name the states of $\mathbf{E}$ from r_1 to r_n, so that α_i is enabled in state r_i. We show by an inductive argument that in state r_i of $\mathbf{E}$ we have $\forall j.\ g_j(\alpha_j) = 0$, until $g_i(\alpha_i) = 1$ and $\forall j \neq i,\ g_j(\alpha_j) = 0$, at which point α_i can be taken and thus E is satisfied.

- $i = 1$. Since all variables are initialized to 0 we have $g_1(\alpha_1) = 1$ and $\forall j \neq 1.\ g_j(\alpha_j) = 0$.

- $i = k,\ 1 \leq k < n$ (the argument for $i = n$ is similar). In state r_k of $\mathbf{E}$ let $g_k(\alpha_k) = 1 \wedge \forall j \neq k.\ g_j(\alpha_j) = 0$, i.e. α_k is the only event enabled in r_k. When α_k is taken, it sets $x_{kk} := \overline{x_{kk}}$ and moves $\mathbf{E}$ to state r_{k+1}. Since $g_k(\alpha_k) = x_{kk} \oplus x_{k,k-1}$ was previously 1, after the assignment $x_{kk} := \overline{x_{kk}}$ the guard formula $g_k(\alpha_k)$ evaluates to 0. Thus, temporarily we have $\forall j.\ g_j(\alpha_j) = 0$.

  Observe that when the value of the private variable of $\mathbf{P_k}$ is changed, communication eventually updates all copies $x_{jk},\ j \neq k$, with x_{kk}. Since $g_j(\alpha_j)$ is only a function of x_{jj} and $x_{j,j-1}$, the only guard formula that will be affected by such communications is $g_{k+1}(\alpha_{k+1}) = x_{k+1,k+1} \oplus x_{k+1,k}$, which evaluates to 1 after $x_{k+1,k}$ is updated with the value of x_{kk}. Thus, we have established that in state r_{k+1} eventually $g_{k+1}(\alpha_{k+1}) = 1$ and $\forall j \neq k+1.\ g_j(\alpha_j) = 0$. The proof is complete. ■

Remark 3.7 The restriction on the network can be relaxed if there is a dedicated communication channel between each pair of processes, that is, we have $\forall i, j.\ \mathbf{P_{ix}} \rightarrow \mathbf{P_{jx}}$. In this case, the copy of the private variable of $\mathbf{P_{ix}}$ in $\mathbf{P_{jx}}$ is updated only when a *direct* communication from $\mathbf{P_{ix}}$ to $\mathbf{P_{jx}}$ is received error-free: $a_j(x_{ji}, \beta_{ij}^r) = x_{ii}$, while for $k \notin \{i, j\}$ we have $a_j(x_{jk}, \beta_{ij}^r) = x_{jk}$. □

In the next examples Theorem 3.3 is used to design decentralized communicating supervisors.

Example 3.10 Consider a system consisting of 4 processes in Fig. 3.15. The dynamics of each process is unimportant and is thus abstracted by self-loops. Shown in the same figure is a centralized supervisor **S** enforcing an ordering between events, which we would like to implement by decentralized supervisors embedded in each process. Note that conditions of Theorem 3.3 are satisfied. The complete design is shown in Fig. 3.16. ◊

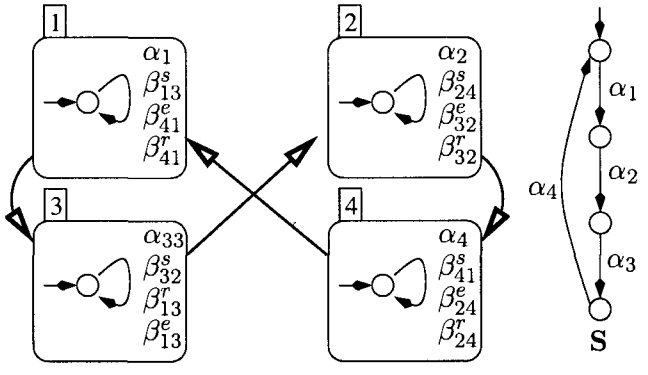


Figure 3.15: Four processes in a deterministic network and the centralized supervisor **S**.

Example 3.11 The complete model of ABP in EFSM framework is shown in Fig. 3.17. ◊

3.7 Conclusion

Our formulation of the class of protocol synthesis problems (including ABP) makes it plausible to think that over ideal channels the problem of "protocol

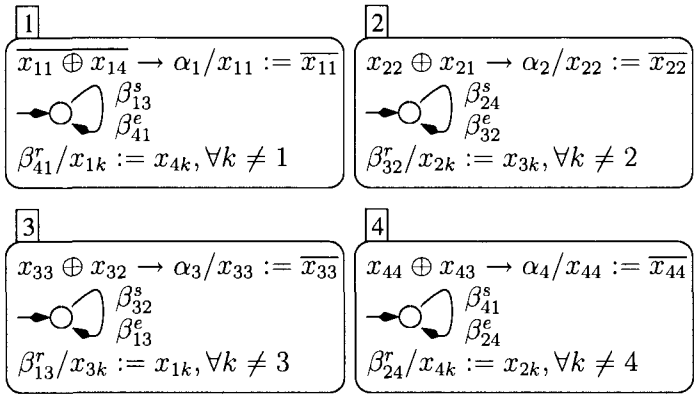


Figure 3.16: The complete design for the system of Fig. 3.15.

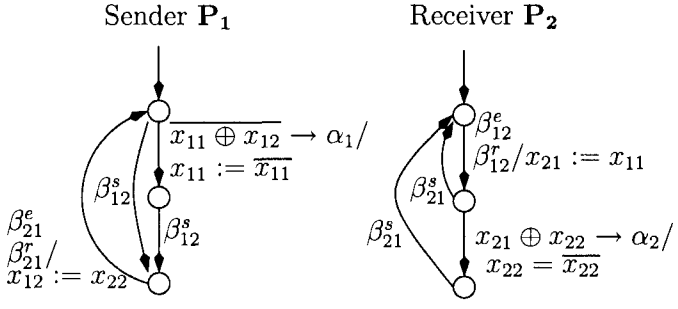


Figure 3.17: ABP design in EFSM framework.

design" for communication processes with non-coobservable specifications can be reduced to the synthesis of communicating decentralized supervisors. Solutions to a special class of problems are presented where the processes need to communicate amongst themselves only for control, and a positive result is stated when channels are unreliable.

One of the important contributions of this chapter is that the crucial role the communication network plays in solvability of the decentralized control problem is investigated. With the exception of [29], most works in this area leave one with the impression that generalization from the case where $n = 2$ to arbitrary $n > 2$ is straightforward. Interestingly enough, when $n = 2$ the network is always deterministic. As discussed in this chapter, for $n > 2$, one has to require that the network be deterministic, or that every process be

connected to every other process through dedicated channels.

Chapter 4

The Framework of Supervised Discrete-Event Systems

4.1 Introduction

In the authors' viewpoint, studying the communication, as a part of the decentralized supervisor design, would be more fruitful if the algebraic structure of the supervisors were explored and utilized in more detail. To this end, the states of the corresponding centralized supervisor are encoded in a distributed way and, in a "divide and conquer" approach, the supervisor's observation and control tasks are represented as dynamic and algebraic equations. Thereby, the dynamics-related information of the transition graph of the centralized supervisor would be amenable to characterizations based on concrete algebraic structures, such as groups and fields. This makes the study of complex systems, and specifically computation of the communication, systematic and simplified. Furthermore, this approach leads to a finer partitioning of the communication and establishes insightful correspondences to the supervisor's behavioral properties [50]. Moreover, since in practice a supervisor is implemented through

encoding its system-related information, employing a (distributed) encoding scheme as an integral part of the (decentralized) supervisor design can address implementation issues, too. As an example, it is important to reduce the communication content as it might be exchanged millions of times a day as part of a communication protocol. Since in practice the content is measured in bits[1] rather than "events" or "state estimates," this issue may be addressed properly in a framework in which design is performed with an insight into quantitative measures for implementation.

Inspired by such considerations, the author first proposed the distributed EFSM framework to study the communication among supervisors [47]. An EFSM models a closed-loop system by employing a state labeling map and computing guards and actions, defined on Boolean variables, as means to observe and control the plant's behavior, respectively [48], [49]. In [50] Agent-wise Labeing Maps (ALMs) were introduced as efficiently-computable replacements for natural projections, based on which the control-related information of a centralized supervisor is represented by a network of *distributed* EFSMs. It was then observed that introducing Boolean variables at an early stage hinders further development of a theory for communicating supervisors by making the notations and computations unnecessarily cumbersome. Furthermore, working with an integer variable, as a meaningful representation of a state, inherits both the qualitative and the quantitative[2] viewpoints to analysis and synthesis. This led the authors to propose a flexibly abstract mathematical framework for decentralized control synthesis, which, while taking an abstract qualitative-like vantage point to system representation, can

[1] When talking about "bits of information," we mean binary digits, not bits in the sense of information theory.

[2] "Qualitative" in the sense of representing a state and thus bearing an abstract meaning, and "quantitative" in the sense of being readily implementable using Boolean variables and serving to define quantitative measures such as those used for minimality analysis.

readily lend itself to the concrete implementation[3] of decentralized supervisors by (Boolean) variables at a later stage.

This chapter assumes that a centralized supervisor is already designed using SCT and introduces an ALM to label its states with disjoint sets of integer vectors whose component i encode the ith decentralized supervisor's observation of the states. For each event updating functions are defined to specify how vectors should be updated by its occurrence, and guard functions are defined to identify the vectors at which it should be enabled. We refer to a DES equipped with guard and updating maps as a Supervised DES (SDES). An SDES inherits the centralized supervisor's properties such as optimality and nonblockingness. SDES framework, which encompasses EFSM framework as its special case, is first developed for centralized supervisory control, and then is extended to the decentralized case. Guard and updating functions are then formulated as polynomial equations over finite fields, thereby representing an SDES as a *polynomial dynamical system* (PDS).

This chapter is organized as follows. Section 4.2 introduces SDES framework and employs it to implement centralized supervisors. Sections 4.3 extends the ideas developed in Section 4.2 to the decentralized case through the employment of ALMs. Section 4.4 introduces a polynomial dynamical representation for a DSDES. The chapter ends with a comment on computational complexity of computing a DSDES and its PDS representation in Section 4.5.

[3]Whereas the implementation of state machines is well understood, the advantage of the proposed approach is to envisage such implementation from the early stages of the theoretical study and integrate it in theoretical formulations. This is justified by the fact that it is more near-to-optimal to consider simultaneously both theoretical and implementation issues as much as possible to address their common aspects and avoid redundant modeling, analysis, and computations. Whereas this optimality is desirable, often it is computationally more difficult to do, leading researchers to divide the whole design process into theoretical and practical phases.

4.2 Supervised DESs

Notation 4.1 Let a given alphabet Σ be partitioned into controllable Σ_c and uncontrollable Σ_{uc} subalphabets, i.e. $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Also assume a second partitioning into observable Σ_o and unobservable Σ_{uo} subalphabets is given, i.e. $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Let $P : \Sigma^* \rightarrow \Sigma_o^*$ be a natural projection which simply erases from $s \in \Sigma^*$ all events in Σ_{uo}. Assume that a language $L \subseteq \Sigma^*$ is given and denote its prefix-closure by $\overline{L}$. Define $\mathbb{N} = \{0, 1, \cdots\}$. □

4.2.1 The formalism of supervised DESs

Definition 4.1 A Supervised DES (SDES) $\mathcal{D}$ is denoted by a quadruple $\mathcal{D} = (\Sigma, L, \mathcal{A}, \mathcal{G})$ where

- Σ is a finite set of events (alphabet);
- L is a (regular) language defined over Σ;
- $\mathcal{A} : \Sigma \times \mathbb{N} \rightarrow \mathbb{N}$ is called an *updating* function;
- $\mathcal{G} : \Sigma \rightarrow pwr(\mathbb{N})$ is a *guard* function. □

Map $\mathcal{A}$ is extended to $\mathcal{A} : \Sigma^* \times \mathbb{N} \rightarrow \mathbb{N}$ according to: for $v \in \mathbb{N}$, $\mathcal{A}(\epsilon, v) = v$, and for $s \in \Sigma^*$ and $\sigma \in \Sigma$, $\mathcal{A}(s\sigma, v) = \mathcal{A}(\sigma, \mathcal{A}(s, v))$. An SDES is thus a language equipped with two mappings.

Definition 4.2 The closed and marked languages of an SDES $\mathcal{D} = (\Sigma, L, \mathcal{A}, \mathcal{G})$ are denoted by $L(\mathcal{D})$ and $L_m(\mathcal{D})$, respectively, and are defined recursively as follows: $\epsilon \in L(\mathcal{D})$ and

$$\forall s \in \Sigma^*, \sigma \in \Sigma. \quad s\sigma \in L(\mathcal{D}) \iff [s \in L(\mathcal{D}) \wedge s\sigma \in \overline{L} \wedge \mathcal{A}(s, 0) \in \mathcal{G}(\sigma)],$$

and

$$L_m(\mathcal{D}) = L(\mathcal{D}) \cap L.$$ □

The semantics of $\mathcal{D}$ is as follows: to each string $s \in \Sigma^*$ a label $\mathcal{A}(s,0)$ is attached. Thus, starting recursively from ϵ, if s is in the behavior of $\mathcal{D}$ and $\sigma \in \Sigma$ is eligible in $\overline{L}$ after s (i.e. $s\sigma \in \overline{L}$), then σ is "enabled" if the label of s is in the image of σ under the guard function, i.e. $\mathcal{A}(s,0) \in \mathcal{G}(\sigma)$. When σ is taken, the label of $s\sigma$ is computed according to $\mathcal{A}(s\sigma,0) = \mathcal{A}(\sigma, \mathcal{A}(s,0))$. From Definition 4.2, the behavior of $\mathcal{D}$ is a subset of L. An SDES is obtained by *guarding* events, i.e. limiting their occurrence, based on the *observation* of event sequences of the guarded language. Thereby, an SDES is equipped with means to *control* and *observe* a given behavior L; in other words, an SDES may be used to *implement* the control decisions of an already designed supervisor for L, and correspondingly it is suitable to model a closed-loop DES. Example 4.1 of the next subsection shows an SDES.

4.2.2 SDESs and centralized supervision

Assume that a plant is modeled by an automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite set of states, q_0 is the initial state, Q_m is the set of marked states, and $\delta : Q \times \Sigma \to Q$ is the partial transition function. Let $E \subseteq L_m(\mathbf{G}) = L$ be a given specification language. Denote by $\Gamma = \{\gamma \in Pwr(\Sigma) \mid \gamma \supseteq \Sigma_{uc}\}$ the set of all control patterns. A map $W : L(G) \to \Gamma$ such that $ker(P \mid L(G)) \leq ker(W)$ is called a *feasible supervisory control* for $\mathbf{G}$.

Theorem 4.1 ([7], [3] Thm 6.3.1) Let $\emptyset \neq E \subseteq L_m(\mathbf{G})$. There exists a nonblocking feasible supervisory control W for $\mathbf{G}$ such that $L_m(W/\mathbf{G}) = E$ if and only if E is (i) controllable with respect to $\mathbf{G}$, (ii) observable with respect to $(\mathbf{G}, P)$, and (iii) $L_m(\mathbf{G})$-closed. ■

Therefore, if a specification E satisfies the three conditions in Theorem 4.1, there exists a proper supervisor ([3], Chapter 3) $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$

for $\mathbf{G}$ such that $L_m(\mathbf{G}) \cap L_m(\mathbf{S}) = E$. The interaction between $\mathbf{S}$ and $\mathbf{G}$ can be formulated using SDESs in the following way.

Problem 4.1 *Control Problem for SDESs:* Let $\mathbf{G}$ be a plant, E be a specification satisfying the conditions of Theorem 4.1, and $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ be an already designed admissible, feasible and proper supervisor for $\mathbf{G}$ such that $L_m(\mathbf{S}) \cap L_m(\mathbf{G}) = E$. Design updating and guard functions $\mathcal{A}$ and $\mathcal{G}$ for SDES $\mathcal{D} = (\Sigma, L_m(\mathbf{G}), \mathcal{A}, \mathcal{G})$ such that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. □

In other words, we seek to design mappings $\mathcal{A}$ and $\mathcal{G}$ which implement a given supervisor. Towards this end, first, a labeling map is introduced to encode the states of $\mathbf{S}$.

Definition 4.3 A map $\ell : R \to \mathbb{N}$ is a *Global Labeling Map* (GLM) for $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ if

1. $\ell(r_0) = 0$, and
2. $\forall r, r' \in R,\ \ell(r) = \ell(r') \implies r = r'$. □

Note that a GLM yields a finite image when applied to a finite automaton.

When a GLM is employed to encode the states of a finite deterministic automaton, its transition structure may be equivalently represented by updating functions, as suggested by Lemma 4.1.

Lemma 4.1 Let $\mathbf{S}$ be a centralized supervisor whose closed language is denoted by $L(\mathbf{S})$ and $\ell(.)$ be a GLM labeling the states of $\mathbf{S}$. Define $\mathcal{A}$ according to

$$\forall r, r' \in R, \forall \sigma \in \Sigma. \quad \xi(r, \sigma) = r' \implies \mathcal{A}(\sigma, \ell(r)) = \ell(r') \tag{4.1}$$

Then $\forall s \in L(\mathbf{S}), \forall r \in R. \quad r = \xi(r_0, s) \iff \ell(r) = \mathcal{A}(s, 0)$. [4] ■

[4] When not mentioned, proofs can be found in the appendix.

Proposition 4.2 *A solution to Problem 4.1:* Let $\mathbf{G}$ and $\mathbf{S}$ be as before, and ℓ be a GLM. Define $\mathcal{D} = (L_m(\mathbf{G}), \Sigma, \mathcal{A}, \mathcal{G})$, where $\mathcal{A}$ is defined as in Lemma 4.1 and $\mathcal{G}$ is defined as:

$$\forall \sigma \in \Sigma, \qquad \mathcal{G}(\sigma) = \{\ell(r) \mid r \in R \ \wedge \ \xi(r, \sigma)!\} \tag{4.2}$$

Then $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

By Proposition 4.2 the maximal permissiveness of the supervisor $\mathbf{S}$, guaranteed by SCT, is inherited by its implementing SDES, too. Since by SCT no uncontrollable event may be disabled by a supervisor, the image of such an event under guard function may always be taken to be equal to $\cup_{r \in R} \ell(r)$, i.e. it is always enabled. It is then the plant whose behavior may limit the occurrence of any such event. From now on we implicitly assume that $\mathcal{G}(\sigma) = \cup_{r \in R} \ell(r)$ for every uncontrollable event σ, and specify $\mathcal{G}(\sigma)$ for controllable events only. Similarly, a supervisor designed by SCT makes no state change upon the occurrence of an unobservable event. Hence we always assume that $\mathcal{A}(\sigma, \cdot) = id_{\mathbb{N}}$ for an unobservable event σ, where $id_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} : n \mapsto n$ denotes the identity function, and specify $\mathcal{A}(\sigma, \cdot)$ for observable events only.

Example 4.1 Figure 4.1-a shows plant $\mathbf{G}$ which is defined over $\Sigma = \{\alpha, \beta, \gamma\}$, where $\Sigma_o = \{\alpha\}$ and $\Sigma_c = \{\gamma\}$. It can be verified that the specification $\mathbf{E}$ in Part (b) is $L_m(\mathbf{G})$-closed, controllable and observable. Computed using SCT, $\mathbf{S}$ in Part (c) is a minimally restrictive supervisor which enforces $L_m(\mathbf{E})$. The states of $\mathbf{S}$ are encoded as 0 and 1, based on which the guard and updating functions (for controllable and observable events, respectively) are computed

as follows:

$$\mathcal{G}(\gamma) = \{1\}, \quad \mathcal{A}(\alpha, m) = \begin{cases} 1; & \text{if } m = 0, \\ \text{arbitrary}; & \text{if } m \neq 0. \end{cases}$$

Note that α cannot happen at $m = 1$ because it is ineligible in $\mathbf{G}$. Since values of $m > 1$ can never be reached, for $m \neq 0$, $\mathcal{A}(\alpha, m)$ can be defined arbitrarily, say $\mathcal{A}(\alpha, m) = 1$, in order to simplify its functional form (see Section 4.4). Thus we end up with an SDES $\mathcal{D} = (\Sigma, L, \mathcal{A}, \mathcal{G})$ where $L = L_m(\mathbf{G})$, and $\mathcal{A}$ (and $\mathcal{G}$) are defined for observable (and controllable) events as specified above and for the other events as explained in the paragraph preceding this example. The languages of $\mathcal{D}$ are $L_m(\mathcal{D}) = L_m(\mathbf{G}) \cap L_m(\mathbf{S})$ and $L(\mathcal{D}) = \overline{L_m(\mathcal{D})}$. $\Diamond$

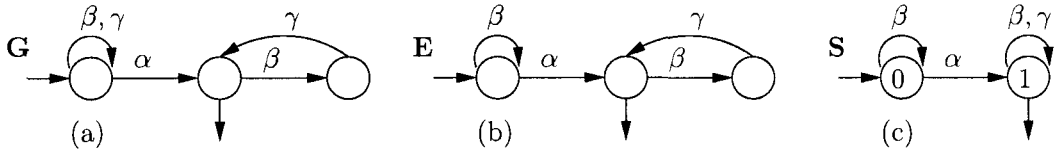


Figure 4.1: (a) Plant $\mathbf{G}$, (b) specification $\mathbf{E}$, and (c) supervisor $\mathbf{S}$ with labeled states.

4.3 Distributed SDESs and decentralized supervisors

This section extends SDES formalism to the case of decentralized supervisors for controlling a (distributed) plant. Consider a network consisting of distributed sensors and actuators as means to observe and control, respectively, the plant's behavior $L \subseteq \Sigma^*$ by n supervisors.

Notation 4.2 Fix $I = \{1, 2, \ldots, n\}$ and associated with the ith supervisor, $\mathbf{S}_i$, in the network ($i \in I$), define observable and controllable event subsets $\Sigma_{o,i}$ and $\Sigma_{c,i}$, respectively, where $\Sigma_{o,i}, \Sigma_{c,i} \subseteq \Sigma$, and let $\Sigma_i = \Sigma_{c,i} \cup \Sigma_{o,i}$. Thereby $\mathbf{S}_i$ observes the plant's behavior through its observational window, modeled by the natural projection $P_i : \Sigma^* \to {\Sigma_{o,i}}^*$, and exercises control on events in $\Sigma_{c,i}$. Thus, from $\mathbf{S}_i$'s viewpoint we have $\Sigma_{uo,i} = \Sigma \setminus \Sigma_{o,i}$ and $\Sigma_{uc,i} = \Sigma \setminus \Sigma_{c,i}$. Associated with each event $\sigma \in \Sigma$ denote by $I_o(\sigma)$ the set of all supervisors which can observe σ, i.e. $I_o(\sigma) = \{i \in I \mid \sigma \in \Sigma_{o,i}\}$. We define a centralized supervisor, denoted by $\mathbf{S}$, to be one which has access to all sensors' observations and can exercise control over all controllable events. For this supervisor we define $\Sigma_c = \bigcup_{i \in I} \Sigma_{c,i}$, $\Sigma_o = \bigcup_{i \in I} \Sigma_{o,i}$, $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, $\Sigma_{uc} = \Sigma \setminus \Sigma_c$, and $P : \Sigma^* \to {\Sigma_o}^*$. Denote by $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{N}^n$ a vector of n natural numbers and let $\mathbf{0}$ be a vector of n zeros. We sometimes write $\mathbf{v}$ as (v_i, v_{-i}) to emphasize on its ith component, v_i, where $v_{-i} \in \mathbb{N}^{n-1}$ is obtained by removing v_i from $\mathbf{v}$. Consider a map $\pi_i : \mathbb{N}^n \to \mathbb{N}$ such that $\pi_i(\mathbf{v}) = v_i$ which picks the ith component of v, and extend π_i to a map $pwr(\mathbb{N}^n) \to pwr(\mathbb{N})$. For $\mathbf{v}, \mathbf{v}' \in \mathbb{N}^n$, we say $\mathbf{v}$ is an *i-sibling* of $\mathbf{v}'$ if $v_i \neq v'_i$ and $v_{-i} = v'_{-i}$. □

4.3.1 Distributed SDESs and synthesis of decentralized supervisors

Definition 4.4 *Distributed SDES:* A Distributed SDES (DSDES) is denoted by $\mathcal{D} = \{\mathcal{D}_i\}_{i \in I}$, where each quadruple $\mathcal{D}_i = (\Sigma, L, \mathcal{A}_i, \mathcal{G}_i)$ is defined as follows.

- Σ is a finite set of events (alphabet);
- L is a (regular) language defined over Σ;
- $\mathcal{A}_i : \Sigma_i \times \mathbb{N}^n \to \mathbb{N}$ is an *updating* function;
- $\mathcal{G}_i : \Sigma_i \to pwr(\mathbb{N}^n)$ is a *guard* function. □

For convenience we extend the domain of $\mathcal{A}_i$ and $\mathcal{G}_i$ to the alphabet of all events. Define $\hat{\mathcal{A}}_i : \Sigma \times \mathbb{N}^n \to \mathbb{N}$ and $\hat{\mathcal{G}}_i : \Sigma \to pwr(\mathbb{N}^n)$ according to: for $\sigma \in \Sigma$ and $\mathbf{v} \in \mathbb{N}^n$,

$$\hat{\mathcal{A}}_i(\sigma, \mathbf{v}) = \begin{cases} \mathcal{A}_i(\sigma, \mathbf{v}) & ; \quad \sigma \in \Sigma_i \\ \pi_i(\mathbf{v}) & ; \quad \sigma \notin \Sigma_i \end{cases}, \qquad \hat{\mathcal{G}}_i(\sigma) = \begin{cases} \mathcal{G}_i(\sigma) & ; \quad \sigma \in \Sigma_i \\ \mathbb{N}^n & ; \quad \sigma \notin \Sigma_i \end{cases} \tag{4.3}$$

With a slight abuse of notation, we shall use $\mathcal{A}_i$ and $\mathcal{G}_i$ to denote $\hat{\mathcal{A}}_i$ and $\hat{\mathcal{G}}_i$, respectively. Define a map $\mathcal{A} : \Sigma^* \times \mathbb{N}^n \to \mathbb{N}^n$ recursively as

$$\forall \mathbf{v} \in \mathbb{N}^n, \forall s \in \Sigma^*, \forall \sigma \in \Sigma. \quad \mathcal{A}(\epsilon, \mathbf{v}) = \mathbf{v}; \ \mathcal{A}(s\sigma, \mathbf{v}) = \Big(\mathcal{A}_i(\sigma, \mathcal{A}(s, \mathbf{v}))\Big)_{i \in I}. \tag{4.4}$$

Definition 4.5 The closed and marked languages of $\mathcal{D}_i$ are denoted by $L(\mathcal{D}_i)$ and $L_m(\mathcal{D}_i)$, respectively, and are defined as follows: $\epsilon \in L(\mathcal{D}_i)$ and

$$\forall s \in \Sigma^*, \sigma \in \Sigma. \ s\sigma \in L(\mathcal{D}_i) \iff [s \in L(\mathcal{D}_i) \wedge s\sigma \in \overline{L} \wedge \mathcal{A}(s, \mathbf{0}) \in \mathcal{G}_i(\sigma)],$$

and

$$L_m(\mathcal{D}_i) = L(\mathcal{D}_i) \cap L.$$

The closed and marked languages of a DSDES $\mathcal{D} = \{\mathcal{D}_i\}_{i \in I}$ are denoted by $L(\mathcal{D})$ and $L_m(\mathcal{D})$, respectively, and are defined as $L(\mathcal{D}) = \cap_{i \in I} L(\mathcal{D}_i)$ and $L_m(\mathcal{D}) = \cap_{i \in I} L_m(\mathcal{D}_i)$. □

Associated with each index $i \in I$, a DSDES is equipped with guard and updating functions to capture control and observation, respectively. Control for each $\mathcal{D}_i$ is based upon n dimensional vectors of natural numbers; component i of a vector is updated with $\mathcal{A}_i$.

Problem 4.2 *Control problem for DSDESs:* Let the plant be modeled by an automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ and E be a specification satisfying the conditions of Theorem 4.1, enforced by a proper, feasible and admissible centralized

supervisor $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$. Design guard and updating functions for each $\mathcal{D}_i = (\Sigma, L_m(G), \mathcal{A}_i, \mathcal{G}_i)$ such that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. □

4.3.2 Agent-wise labeling maps and design of updating and guard functions

In what follows we investigate a solution to Problem 4.2. To begin with, note that if E is decomposable [12] into n component specifications, each defined over the subalphabet Σ_i, then the problem would be reduced to n independent monolithic designs, which can be done using the method of Subsection 4.2.2. Also, if the specification is coobservable, there exist n decentralized supervisors $\mathbf{S_i}$, $i \in I$, each partially observing the plant's behavior directly through P_i and exercising control over events in $\Sigma_{c,i}$, such that their concurrent operation enforces the specification. In this case, separate application of the method of Subsection 4.2.2 to each $\mathbf{S_i}$ would result in guard and updating functions which depend only on the labels associated with the states of $\mathbf{S_i}$. It can be shown that this set of independent guard and updating functions would solve the problem, too. Since the focus of the present work is to develop SDES framework to model communication among decentralized supervisors, we assume that specification E is neither decomposable nor coobservable with respect to $(\mathbf{G}, P_1, \cdots, P_n)$ [12].

We start by introducing a labeling map[5] which encodes the states of the centralized supervisor, $\mathbf{S}$. In accordance with the distributed nature of the system, such a labeling scheme should reflect the observations of the plant's behavior from the viewpoint of n "component supervisors." To define such

[5]The material on agent-wise labeling maps originates in Chapter 3 and is repeated briefly here for the sake of notational convenience and clarity.

labeling schemes, the selfloops of $\mathbf{S}$ should be modified such that if an (observable) event makes a state change at some state, it results in a state change at every state where it makes a transition (see Remark 3.1). The desired labeling maps are characterized below.

Definition 4.6 ([50], Defn. 4 and Chapter 3, Defn. 3.4) Let $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ be a centralized supervisor. An Agent-wise Labeling Map (ALM) for $\mathbf{S}$ is a map $\ell : R \to pwr(\mathbb{N}^n)$ with the following properties:

1. $\mathbf{0} \in \ell(r_0)$;

2. $\forall r, r' \in R.\ r \neq r' \Rightarrow \ell(r) \cap \ell(r') = \emptyset$ (labels are disjoint);

3. $\forall r, r' \in R, r \neq r',\ \forall \sigma \in \Sigma_o,\ \forall \mathbf{v} \in \mathbb{N}^n.\quad \mathbf{v} \in \ell(r) \wedge r' = \xi(r, \sigma)$
$\Longrightarrow\quad [\exists! \mathbf{v}' \in \mathbb{N}^n.\ \mathbf{v}' \in \ell(r')\ \wedge\ (\forall i \in I_o(\sigma).\ v_i \neq v'_i)\ \wedge\ (\forall j \in I \setminus I_o(\sigma).\ v_j = v'_j)]$. □

For every $\mathbf{S}$, Theorem 4 in [50] (also Chapter 3, Theorem 3.2) provides a constructive proof for the existence of an efficiently computable finite ALM, i.e. one with a finite image, based on the structure of a *Latin hypercube* [77] of dimension n and side of the size of R. Employing the structural information of $\mathbf{S}$, such as symmetries of states and transitions, might help reduce the side of the Latin hypercube. This issue, whose one instance is in Example 4.2, is currently under investigation.

The existence of a finite ALM paves the way for defining the updating functions associated with a DSDES $\{\mathcal{D}_i\}_{i \in I}$. The key point is the existence of a *unique* i-sibling in $\ell(r')$ of a $\mathbf{v} \in \ell(r)$ whenever there is a $\sigma \in \Sigma_{o,i}$ such that $r' = \xi(r, \sigma)$. This is formalized in the following lemma whose proof directly follows from Definition 4.6.

Lemma 4.2 Let $\ell : R \to pwr(\mathbb{N}^n)$ be a finite ALM for $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$. Then there exists a map $\mu : \Sigma \times \mathbb{N}^n \to \mathbb{N}^n$ that is consistent with the labeling of ℓ, i.e.

$$\forall r, r' \in R,\ \forall \sigma \in \Sigma,\ \forall \mathbf{v} \in \mathbb{N}^n.$$
$$\mathbf{v} \in \ell(r) \wedge r' = \xi(r, \sigma) \implies [\mu(\sigma, \mathbf{v}) \in \ell(r')$$
$$\wedge \big(\forall i \in I_o(\sigma).\ \pi_i(\mu(\sigma, \mathbf{v})) \neq \pi_i(\mathbf{v})\big) \wedge \big(\forall j \in I \backslash I_o(\sigma).\ \pi_j(\mu(\sigma, \mathbf{v})) = \pi_j(\mathbf{v})\big)].$$

■

Now, the updating functions can be defined using the map μ so that each supervisor only updates the label components it can observe:

$$\forall r, r' \in R, \forall \sigma \in \Sigma, \forall \mathbf{v} \in \mathbb{N}^n.\ r' = \xi(r, \sigma) \wedge \mathbf{v} \in \ell(r) \implies \mathcal{A}_i(\sigma, \mathbf{v}) = \pi_i(\mu(\sigma, \mathbf{v})). \tag{4.5}$$

This formula relates $\mathcal{A}$ in (4.4) to the transition structure of $\mathbf{S}$ as shown next.

Lemma 4.3 Let $\mathbf{S}$ and ℓ be as in Lemma 4.2. We have:

$$\forall s \in L(\mathbf{S}), \forall r \in R.\quad \mathcal{A}(s, \mathbf{0}) \in \ell(r) \iff r = \xi(r_0, s).$$ ■

Proposition 4.3 *A solution to Problem 4.2:* Let $\mathbf{G}$, E, $\mathbf{S}$, and $\mathcal{D}_i = (L_m(\mathbf{G}), \Sigma, \mathcal{G}_i, \mathcal{A}_i)$ be as in Problem 4.2 and ℓ and μ be as in Lemma 4.2. For all $i \in I$, let $\mathcal{A}_i$ be as in (4.5), and $\mathcal{G}_i$ be as follows.

$$\forall \sigma \in \Sigma. \qquad \mathcal{G}_i(\sigma) = \begin{cases} \{\ell(r) \mid r \in R \wedge \xi(r, \sigma)!\}; & \text{if } \sigma \in \Sigma_{c,i}, \\ \cup_{r \in R} \ell(r); & \text{if } \sigma \in \Sigma_{uc,i}. \end{cases} \tag{4.6}$$

Then $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

The following result can be easily proved using the definitions of μ and $\mathcal{A}_i$.

Corollary 4.1 Following (4.5), the updating function corresponding to every unobservable event is an identity function, i.e. we have

$$\forall i \in I, \forall \mathbf{v} \in \mathbb{N}^n, \forall \sigma \in \Sigma_{uo,i}.\ \mathcal{A}_i(\sigma, \mathbf{v}) = v_i.$$ ■

Example 4.2 (Construction of the DSDES for a network of 3 supervisors): Figure 4.2-a shows the model of a distributed network consisting of three plant components and four events $\alpha_1, \alpha_2, \alpha_3$ and β, where $\Sigma_{c,1} = \Sigma_{o,1} = \{\alpha_1, \beta\}$, $\Sigma_{c,2} = \Sigma_{o,2} = \{\alpha_2, \beta\}$, and $\Sigma_{c,3} = \Sigma_{o,3} = \{\alpha_3\}$. The specification $\mathbf{S}$ is shown in part (b) of the same figure and all its states are marked. Since $\Sigma = \Sigma_o = \Sigma_c$ and $\mathbf{S}$ is $L_m(\mathbf{G})$-closed, the specification meets the conditions of Theorem 4.1 and thus $\mathbf{S}$ is a proper centralized supervisor, too.

Following the explanations provided in Subsection 4.3.2, to define an ALM for $\mathbf{S}$ we unfold the selfloop at state r_2 since its event, α_2, causes a state change at r_1 and r_3. This results in the new finite deterministic automaton $\hat{\mathbf{S}}$ shown in part (c).

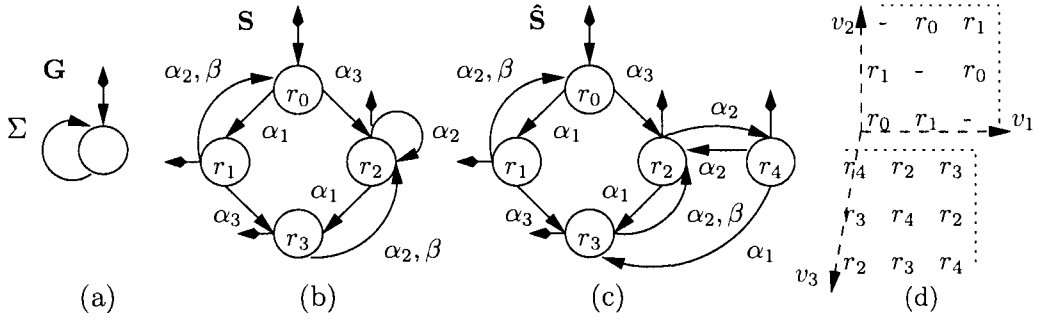


Figure 4.2: (a) The plant's model. (b) The specification. (c) The modified centralized supervisor. (d) The ALM-assigned labels; a point $\mathbf{v} \in \mathbb{N}^3$ is labeled with $r \in R$ if and only if $\mathbf{v} \in \ell(r)$; thus, for example, $\ell(r_4) = \{(201), (111), (021)\}$.

While the proof of Theorem 3.2 suggests to look up the labels within a Latin hypercube of side 5 and dimension 3, we notice that, in the transition structure of $\hat{\mathbf{S}}$, event α_3 merely takes the system from a state in the set $A = \{r_0, r_1\}$ to a state in the set $B = \{r_2, r_3, r_4\}$; this can be represented by a

function $f : A \to B$ where $f(r_0) = r_2$ and $f(r_1) = r_3$. Moreover, within each set the state transitions are labeled with events in $\Sigma_{o,1} \cup \Sigma_{o,2}$ such that for $r, r' \in A$, $\hat{r} \in B$ and $\sigma \in \Sigma_{o,1} \cup \Sigma_{o,2}$, if $\hat{r} = f(r)$, $r' = \xi(r, \sigma)$ and $\xi(\hat{r}, \sigma)!$, then $\xi(\hat{r}, \sigma) = f(r')$. This symmetry between the two sets of states allows us to choose the required labels from two Latin squares of side 3 whose elements are 3-siblings. This arrangement is shown in part (d), based on which the states' labels are as follows (A point $(a, b, c) \in \mathbb{N}^3$ is denoted by 'abc').

$$\ell(r_0) = \{000, 210, 120\}, \quad \ell(r_1) = \{100, 010, 220\}, \quad \ell(r_2) = \{001, 211, 121\},$$
$$\ell(r_3) = \{101, 011, 221\}, \quad \ell(r_4) = \{201, 021, 111\}.$$

Correspondingly, guard functions are computed as follows:

$$\mathcal{G}_1(\alpha_1) = \ell(r_0) \cup \ell(r_2) \cup \ell(r_4), \quad \mathcal{G}_2(\alpha_2) = \ell(r_1) \cup \ell(r_2) \cup \ell(r_3) \cup \ell(r_4),$$
$$\mathcal{G}_1(\beta) = \mathcal{G}_2(\beta) = \ell(r_1) \cup \ell(r_3), \quad \mathcal{G}_3(\alpha_3) = \ell(r_0) \cup \ell(r_1).$$

The updating functions are listed in Table 4.1. Arbitrary cases are denoted by "$-$" and are chosen later to simplify the guard and updating functions in the DSDES and EFSM frameworks. $\Diamond$

4.4 Guard and updating functions as polynomials over a finite field

Proposition 4.3 insures that the choice of updating and guard functions in (4.5) and (4.6) would indeed implement the centralized supervisory control in

Table 4.1: Updating functions ($a, b \in \{0,1,2\}$, $c \in \{0,1\}$) for the DSDES in Example 4.2

$\mathbf{v}$	$00c$	$21c$	$12c$	201	111	021	else			
$\mathcal{A}(\alpha_1, \mathbf{v})$	$10c$	$01c$	$22c$	101	011	221	–			
$\mathbf{v}$	$01c$	$10c$	$22c$	021	111	201	001	121	211	else
$\mathcal{A}(\alpha_2, \mathbf{v})$	$00c$	$12c$	$21c$	001	121	211	021	111	201	–
$\mathbf{v}$	$10c$	$01c$	$22c$	else						
$\mathcal{A}(\beta, \mathbf{v})$	$21c$	$12c$	$00c$	–						
$\mathbf{v}$	$ab0$	else								
$\mathcal{A}(\alpha_3, \mathbf{v})$	$ab1$	–								

a decentralized way. Although these two kinds of functions provide a compact way to present observation- and control-related information using a finite number of integer (vector) labels, it should be clear that the ALM does not rely on the absolute positions of these labels. In fact, it is the relative position of them which reflects the dynamic information structure, which, as in the case of (state-space) dynamical systems in control theory, could serve more control purposes if put in a *symbolic* form. To this end, once the labels of the states of $\mathbf{S}$ are represented by vectors of integer variables, the observation- and control-related information may be captured by (polynomial) functions over the underlying finite field from which the label values are selected. Accordingly, one may compute for the guard and updating functions associated with an event σ their characteristic equation and transition function, respectively. Thereby, system information can be accessed and studied in a unified manner using the algebraic structure of the finite fields and their well developed software tools.

4.4.1 Finite field representation of a DSDES

In the following we derive a polynomial representation of a DSDES over a finite field using a simple, yet general method called *interpolation polynomials in the Lagrange form* [80].

Notation 4.3 Let $p \geq 2$ be a natural number, $\mathbb{F}_p = \{0, 1, \cdots, p-1\}$ be a finite field of p integers with addition and multiplication defined modulo p, x_i $(i \in I)$ be a variable taking values from $\mathbb{F}_p$, $\mathbf{x} = (x_1, \cdots, x_n)$, and $\mathbb{F}_p[\mathbf{x}]$ be the ring of polynomials in the variables $x_1, \cdots, x_n$ and coefficients taken from $\mathbb{F}_p$ [81]. For a function or formula f, by writing $f(\mathbf{x})$ we mean that f can in general depend on some or all of the elements of $\mathbf{x}$. The set of variables on which f precisely depends is denoted by either $\arg(f)$ or explicit listing of such variables. Let x_i be $\mathbf{S}_i$'s private variable, with respect to which all x_j's, $j \in I \setminus \{i\}$, are referred to as *external* variables. For an event $\sigma \in \Sigma$, the polynomials corresponding to $\mathcal{A}_i(\sigma, .)$ and $\mathcal{G}_i(\sigma)$ are denoted by $\mathfrak{a}_i^\sigma(\mathbf{x})$ and $\mathfrak{g}_i^\sigma(\mathbf{x})$, respectively. □

As an example, let $\mathbb{F}_3 = \{0, 1, 2\}$, $n = 2$, and $\mathbf{x} = (x_1, x_2)$ and $\mathfrak{a}_i^\alpha(\mathbf{x}) = x_1^2 + 1$ replace $\mathcal{A}_2(\sigma, .)$ to update x_2 upon the observation of $\alpha \in \Sigma_{o,2}$. Here $\arg(\mathfrak{a}_i^\alpha) = \{x_1\}$.

Given the graph of a function $f : \mathbb{F}_p^n \to \mathbb{F}_p$ as a set $\mathcal{U} = \{(\mathbf{u}, y) \in \mathbb{F}_p^n \times \mathbb{F}_p \mid y = f(\mathbf{u})\}$, the method of "interpolation polynomials in the Lagrange form" computes a polynomial $q \in \mathbb{F}_p[\mathbf{x}]$ such that $q(\mathbf{u}) = f(\mathbf{u})$. The computation is based on the following fraction [82]

$$L_i(x) = \frac{x(x-1)\cdots(x-i+1)(x-i-1)\cdots(x-p+1)}{i(i-1)\cdots(i-(i-1))(i-(i+1))\cdots(i-p+1)}, \qquad (4.7)$$

which is 1 at $x = i$ and 0 otherwise. Let $\mathbf{u} \in \mathbb{F}_p^n$ and define

$$L_{\mathbf{u}}(\mathbf{x}) = L_{u_1 u_2 \cdots u_n}(\mathbf{x}) = L_{u_1}(x_1) \cdots L_{u_n}(x_n), \tag{4.8}$$

where we assume that the subscripts of L correspond to its arguments in the order they appear. Observe that $L_{\mathbf{u}}(\mathbf{x})$ is 1 at $\mathbf{x} = \mathbf{u}$ and 0 otherwise. The required polynomial would be as follows.

$$q(\mathbf{x}) = \sum_{\mathbf{u} \in \mathbb{F}_p^n} L_{\mathbf{u}}(\mathbf{x}) f(\mathbf{u}) \tag{4.9}$$

Algorithm 4.1 *Computation of polynomial functions associated with guard and updating functions:* Given a DSDES with $\mathbf{S}$ and ℓ as in Proposition 4.3 and updating and guard functions as in (4.5) and (4.6), and using the interpolation polynomials in the Lagrange form, do the following.

1. For each $i \in I$ compute $V_i = \pi_i(\bigcup_{r \in R} \ell(r))$ and $p_i = \max_{v \in V_i} v$. Determine $p = \inf\{p' \in \mathbb{N} \mid [\forall i \in I.\ p' \geq p_i] \ \wedge \ \mathbb{F}_{p'} \text{ is a field}\}$. Choose $\mathbb{F}_p^n$ as the smallest common underlying finite field which accommodates the whole range of the label values assigned by ℓ.

2. For each $\mathbf{S}_i$ and associated with each event σ, define $\mathfrak{g}_i^\sigma$ as the characteristic polynomial associated with $\mathcal{G}_i(\sigma)$ in (4.6), and assign the polynomial arbitrarily elsewhere, i.e.

$$\forall i \in I,\ \forall \sigma \in \Sigma. \begin{cases} \mathfrak{g}_i^\sigma(\mathbf{v}) = 1; & \forall \mathbf{v} \in \mathcal{G}_i(\sigma) \\ \mathfrak{g}_i^\sigma(\mathbf{v}) = 0; & \forall \mathbf{v} \in \cup_{r \in R} \ell(r) \setminus \mathcal{G}_i(\sigma) \\ \mathfrak{g}_i^\sigma(\mathbf{v}) = \text{arbitrary}; & \forall \mathbf{v} \in \mathbb{F}_p^n \setminus \cup_{r \in R} \ell(r) \end{cases} \tag{4.10}$$

3. For each $\mathbf{S}_i$ and associated with each event σ, define $\mathfrak{a}_i^\sigma$ as the restriction of the $\mathcal{A}_i(\sigma, \mathbf{v})$ in (4.5) to $\mathbb{N}^n \to \mathbb{N}$, where σ-labeled transitions is eligible

and arbitrary elsewhere, i.e.

$$\forall i \in I,\ \forall \sigma \in \Sigma. \begin{cases} \mathfrak{a}_i^\sigma(\mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{v}); & \forall \mathbf{v} \in \mathcal{G}_i(\sigma) \\ \mathfrak{a}_i^\sigma(\mathbf{v}) = \text{arbitrary}; & \forall \mathbf{v} \in \mathbb{F}_p^n \setminus \mathcal{G}_i(\sigma) \end{cases} \tag{4.11}$$

□

By the ordering of $\mathbb{N}$, p in Step 1 always exists. Notice that $V_i \subseteq \{0, 1, \cdots, p-1\}$ holds for every $i \in I$. The following definition specifies which polynomials serve to represent a DSDES.

Definition 4.7 Let $\mathbf{G}$, E, $\mathbf{S}$, ℓ, be as in Proposition 4.3 and $\mathbb{F}_p$ and $\mathbf{x}$ be as in Notation 4.3. For each $\mathbf{S}_i$ and each $\sigma \in \Sigma$ let polynomial equations $x_i := \mathfrak{a}_i^\sigma(\mathbf{x})$ and $\mathfrak{g}_i^\sigma(\mathbf{x}) = 1$ over $\mathbb{F}_p^n$ replace the updating function $\mathcal{A}_i(\sigma, .)$ in (4.5) and guard function $\mathcal{G}_i(\sigma)$ in (4.6), respectively. The polynomials are said to *represent* the DSDES if they result in $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. □

Algorithm 4.1 computes the required polynomials as shown below.

Proposition 4.4 Let $\mathbf{G}$, E, $\mathbf{S}$, ℓ, be as in Proposition 4.3. The polynomial equations which are obtained by computation of $\mathfrak{a}_i^\sigma$ and $\mathfrak{g}_i^\sigma$ using Algorithm 4.1 represent the DSDES. ■

Remark 4.1 Following Corollary 4.1, Algorithm 4.1 computes identity function for every updating function which is associated with a non-observable event. Similarly, based on (4.6), the algorithm computes unity function for every guard function associated with an uncontrollable event. Therefore these polynomials are not mentioned explicitly. □

Remark 4.2 (Computation of simplified polynomials): As we will see in Section 5.2, to reduce the communication among supervisors, it is often

required that updating or guard polynomials depend on fewer external variables. In DES models transition relations are usually partial functions and this results in the existence of a number of unassigned points in the domain of the updating functions, generally called *arbitrary* or *don't care* cases [73]. Polynomial expressions for updating functions can be simplified by assigning to arbitrary cases the points in the codomain which make the transition structure symmetric, thus eliminating the dependency on some (external) variables. To simplify, using a symmetric structure, the expression for the guard function associated with an event σ, one may assume that σ is enabled at states which are not reachable (i.e. there is no $\sigma \in \Sigma$, $s \in L(\mathbf{S})$, $i \in I$, such that $\mathcal{A}(\sigma, \mathcal{A}(s, \mathbf{0})) = \mathbf{v} \wedge \mathcal{A}(s, \mathbf{0}) \in \mathcal{G}_i(\sigma)$) or at states at which σ is disabled by the plant. When a symmetry is available, the identities

$$\sum_{i=0}^{p-1} L_i(x) = 1, \quad \sum_{j=0}^{p-1} L_k(x) L_j(x') = L_k(x), \quad (k \in \mathbb{F}_p) \tag{4.12}$$

which can be easily verifies, can be used to simplify polynomials. □

The next example illustrates the application of Algorithm 4.1 and the above simplifications.

Example 4.3 (Computation of updating and guard polynomials for Example 4.2): Table 4.2 shows the polynomials which represent the updating and guard functions in Example 4.2. Here $\mathbf{x} = [x_1, x_2, x_3]$, $V_1 = V_2 = \{1, 2, 3\}$, $V_3 = \{1, 2\}$, $p_1 = p_2 = 3$, and $p_3 = 2$. Step 1 of Algorithm 4.1 then determines $p = 3$, i.e. $\mathbb{F}_3$ would be the required underlying finite field. Examples of computations are as follows.

- The table of updating functions can be simplified by proper use of arbitrary cases. We show this for the case of $\mathfrak{a}_1^{\alpha_1}(\mathbf{x})$ here. Based on only the specified cases in Table 4.1 we have

$\mathfrak{a}_1^{\alpha_1}(\mathbf{x}) = 1\times[L_{000}(\mathbf{x}) + L_{001}(\mathbf{x})] + 2\times[L_{120}(\mathbf{x}) + L_{121}(\mathbf{x})] + 1\times[L_{201}(\mathbf{x})] + 2\times[L_{021}(\mathbf{x})]$.

Following (4.12), to eliminate x_3 in the final polynomial expression, it should hold that if $L_{mnt}(\mathbf{x})$ be in $\mathfrak{a}_1^{\alpha_1}(\mathbf{x})$, then for every $t' \in \mathbb{F}_3$, $L_{mnt'}(\mathbf{x})$ should appear in it, too. Observe in Fig. 4.2 that none of the points in plane $x_3 = 2$, nor the points (200), (110), and (020) are reachable from any other points within the two Latin squares. Therefore these points can be mapped arbitrarily to points in $\mathbb{F}_3^3$ whose values of x_1 meet the above requirement (2nd and 3rd elements of the images are still arbitrary). Accordingly we add L_{002}, $2\times L_{122}$, L_{200} and L_{202}, and $2\times[L_{020} + L_{022}]$ to the 4 brackets in $\mathfrak{a}_1^{\alpha_1}(\mathbf{x})$, respectively, i.e.

$$\begin{aligned}\mathfrak{a}_1^{\alpha_1}(\mathbf{x}) &= 1\times[L_{000}(\mathbf{x})+L_{001}(\mathbf{x})+L_{002}(\mathbf{x})]+2\times[L_{120}(\mathbf{x})+L_{121}(\mathbf{x})+L_{122}(\mathbf{x})] + \\ &\quad 1\times[L_{201}(\mathbf{x})+L_{200}(\mathbf{x})+L_{202}(\mathbf{x})]+2\times[L_{021}(\mathbf{x})+L_{020}(\mathbf{x})+L_{022}(\mathbf{x})]. \\ &= L_{00}(x_1,x_2) + 2L_{12}(x_1,x_2) + L_{20}(x_1,x_2) + 2L_{02}(x_1,x_2) \\ &= [L_{00}(x_1,x_2) + L_{20}(x_1,x_2)] + 2[L_{12}(x_1,x_2) + L_{02}(x_1,x_2)]\end{aligned}$$

Hence the above expression does not depend on x_3. The last grouping of the terms suggests that if $L_{10}(x_1, x_2)$ and $2L_{22}(x_1, x_2)$ could be added respectively to the two brackets, x_1 can be eliminated, too. Although x_1 is not an external variable, its elimination may reduce online computational time or save some memory space. To this end, notice that since the occurrence of α_1 is prohibited by $\mathcal{G}_1(\alpha_1)$ at states r_1 and r_3, the labels of these states can be arbitrarily mapped to points whose values of x_1, based on (4.12), allows for such elimination. To this end, we first add new terms to get $L_{10}(x_1, x_2)$ and $2L_{22}(x_1, x_2)$ and then omit x_1.

$$\begin{aligned}\mathfrak{a}_1^{\alpha_1}(\mathbf{x}) &= [L_{00}(x_1,x_2) + L_{20}(x_1,x_2) + \big(L_{100}(\mathbf{x}) + L_{101}(\mathbf{x}) + L_{102}(\mathbf{x})\big)] + \\ &\quad 2[L_{12}(x_1,x_2) + L_{02}(x_1,x_2) + \big(L_{220}(\mathbf{x}) + L_{221}(\mathbf{x}) + L_{222}(\mathbf{x})\big)] \\ &= [L_{00}(x_1,x_2) + L_{20}(x_1,x_2) + L_{10}(x_1,x_2)]+\end{aligned}$$

Table 4.2: Computed polynomials for updating and guard functions the DS-DES in Example 4.2

$x_1 := \mathfrak{a}_1^{\alpha_1}(\mathbf{x}) = 2(x_2+2)$
$x_1 := \mathfrak{a}_1^{\beta}(\mathbf{x}) = x_1+1$
$x_2 := \mathfrak{a}_2^{\alpha_2}(\mathbf{x}) = x_1(x_1+1)+(x_2+2)(2x_1+x_2+1)$
$x_2 := \mathfrak{a}_2^{\beta}(\mathbf{x}) = x_2+1$
$x_3 := \mathfrak{a}_3^{\alpha_3}(\mathbf{x}) = 2(x_3^2+2)$
$\mathfrak{g}_1^{\alpha_1}(\mathbf{x}) = 2(x_1+2)^2(x_2^2+2)+2x_1^2x_2(x_2+1)+2(x_1+1)^2x_2(x_2+2)$
$\mathfrak{g}_2^{\alpha_2}(\mathbf{x}) = 2x_1^2(x_2^2+2)+2(x_1+1)^2x_2(x_2+1)+2(x_1+2)^2x_2(x_2+2)$ $+2x_3(x_3+1)[x_1x_2+2x_1^2+2x_2^2+1]$
$\mathfrak{g}_{1,2}^{\beta}(\mathbf{x}) = x_1(x_1+1)(x_2^2+2)+(x_1^2+2)x_2(x_2+1)+x_1(x_1+2)x_2(x_2+2)$
$\mathfrak{g}_3^{\alpha_3}(\mathbf{x}) = 2(x_3^2+2)$

$$2[L_{12}(x_1,x_2)+L_{02}(x_1,x_2)+L_{22}(x_1,x_2)]$$
$$= L_0(x_2)+2L_2(x_2) = 2(x_2+2).$$

- Originally the polynomial expression for $\mathfrak{g}_1^{\alpha_1}(\mathbf{x})$ would be as follows:

  $\mathfrak{g}_1^{\alpha_1}(\mathbf{x}) = [L_{000}(\mathbf{x})+L_{001}(\mathbf{x})]+[L_{111}(\mathbf{x})]+[L_{201}(\mathbf{x})]+[L_{021}(\mathbf{x})]+[L_{210}(\mathbf{x})+L_{211}(\mathbf{x})]+[L_{120}(\mathbf{x})+L_{121}(\mathbf{x})]$

  To eliminate x_3, one requires to add to the six brackets in the above expression, respectively $L_{002}(\mathbf{x})$, $L_{110}(\mathbf{x})+L_{112}(\mathbf{x})$, $L_{200}(\mathbf{x})+L_{202}(\mathbf{x})$, $L_{020}(\mathbf{x})+L_{021}(\mathbf{x})$, $L_{212}(\mathbf{x})$, and $L_{122}(\mathbf{x})$. None of the states associated with these terms are reachable from any other states in $\hat{\mathbf{S}}$ and therefore we can safely add them to $\mathfrak{g}_1^{\alpha_1}$ to eliminate the dependency on x_3, i.e.

  $\mathfrak{g}_1^{\alpha_1}(\mathbf{x}) = L_{00}(x_1,x_2)+L_{11}(x_1,x_2)+L_{20}(x_1,x_2)+L_{02}(x_1,x_2)+L_{21}(x_1,x_2)+L_{12}(x_1,x_2)$. ◊

It is worth mentioning that neither the above resulting polynomials are claimed to be the most simplified ones nor the computations are necessarily efficient. Such issues can be well studied using ideals and varieties [83] which are beyond the scope of this work. Notice that the above computations can

be used for centralized supervisors in Subsection 4.2.2, too.

DSDESs as polynomial dynamical systems: The above "symbolic" formulation may be viewed as a *polynomial dynamical system* (PDS) [56] in which equations associated with updating and guard functions represent the dynamics of the DES and its algebraic constraints, respectively.

Definition 4.8 [56] Let $\mathbf{x} = (x_1, \cdots, x_n)$ and $\mathbf{y} = (y_1, \cdots, y_m)$ be the vectors of *states* and *events*, respectively. Define $\mathcal{P}(.) = \big(\mathsf{p}_1(.), \cdots, \mathsf{p}_\nu(.)\big)$ and $\mathcal{Q}(.) = \big(\mathsf{q}_1(.), \cdots, \mathsf{q}_\omega(.)\big)$ as finite dimensional vectors of polynomials over $\mathbb{F}_p[\mathbf{x}, \mathbf{y}]$. The set of polynomial equations

$$\begin{cases} \mathbf{x} := \mathcal{P}(\mathbf{x}, \mathbf{y}) \\ \mathcal{Q}(\mathbf{x}, \mathbf{y}) = \mathbf{0} \end{cases} \tag{4.13}$$

are said to be a *polynomial dynamical system* (PDS) in *state explicit* form, where $:=$ denotes the assignment of new values to its left vector, $\mathbf{x}$, and the first and second equations are referred to as *state transition* equation and *constraints* equation, respectively. □

Theorem 4.5 Every DSDES can be represented as a PDS in state explicit form.

Proof: The construction using interpolation polynomials in the Lagrange form, as explained in Algorithm 4.1, yields the following set of equations.

$$\forall i \in I. \begin{cases} x_i := \mathfrak{a}_i^\sigma(\mathbf{x}) & ; \forall \sigma \in \Sigma_{o,i} \\ \mathfrak{g}_i^\sigma(\mathbf{x}) - 1 = 0 & ; \forall \sigma \in \Sigma_{c,i} \end{cases} \tag{4.14}$$

Notice that by Remark 4.1, the $\mathbf{S}_i$'s polynomials associated with unobservable events and uncontrollable events are trivial equations and are not mentioned explicitly. Observe that the equations (4.14) are formally represented in the

same form as (4.13) except that in (4.14) we have a *separate* equation associated with each event, i.e. a system input. This is due to the fact that the DSDES framework does not assume any encoding schemes for events. ■

Example 4.4 (A PDS perspective of the polynomials in Example 4.3): As Table 4.2 reads, the PDS representation of the DSDES in Example 4.2 consists of 5 dynamic and 4 algebraic equations. In the dynamics part, each event can only affect the variable(s) owned by the supervisor(s) which can observe the occurrence of that event. Also the algebraic constraint associated with an event designates where in (here 3-dimensional) integer space that event is enabled. ◊

In [56] PDSs are used to represent uncontrolled plants and their associated given specifications for the purpose of *centralized* control design. However, the objective of this work is *decentralized* control design with an emphasis on communication among supervisors. To this end, we employ an ALM to compute a distributed representation for an already designed centralized supervisor, in which updating functions reflect the limited observational window for each decentralized supervisor and guard functions capture the control it exercises over the plant. The PDS representation of the system reveals more detailed algebraic structures and hence it suits further analysis and design purposes and more specifically, communication design, as explained in Section 5.2.

4.4.2 EFSM implementation of a DSDES

EFSM framework, which was introduced in [49] and [50], offers a bit-wise representation of supervisors' information which suits practical purposes (see [50]). The system information, which is captured by x_is, $i \in I$, in the DSDES framework, can be implemented in EFSM framework using Boolean variables.

This can be done by either direct encoding of the centralized supervisor or using the polynomials of the PDS representation of the DSDES. In the first approach [50], first the number of Boolean variables necessary for encoding the labels associated with each $\mathbf{S}_i$ is determined and then observation and control are captured by *actions* and *guards* on Boolean variables, respectively. Here, we use the second approach, which is introduced next.

Definition 4.9 ([50]-Definition 1 and Subsection III.A) *The EFSM formalism:* Let $\mathbb{B} = \{0, 1\}$, $h = \lceil log_2(p) \rceil$, $J = \{1, 2, \cdots, h\}$, and $i, j \in I$, $j \neq i$. Denote by $X_{ii} = \{x_{ii}^k \mid k \in J\}$ the set of $\mathbf{S}_i$'s *private* Boolean variables which encodes x_i such that $x_i = (x_{ii}^h \cdots x_{ii}^1)$. Denote by x_{ij}^k a copy of $x_{jj}^k \in X_{jj}$ stored by $\mathbf{S}_i$ and let $X_{ij} = \{x_{ij}^k \mid x_{jj}^k \in X_{jj}\}$, $X_{ci} = \bigcup_{j \in I \setminus \{i\}} X_{ij}$, $X_i = X_{ii} \dot{\cup} X_{ci}$, and $X = \bigcup_{i \in I} X_i$. Let G_i denote the set of all Boolean formulas over X_i and A_i denote the set of all Boolean functions $\mathsf{b} : \mathbb{B}^{nh} \rightarrow \mathbb{B}$. Associated with each $i \in I$, $\mathbf{M}_i = (\mathbf{S}, X_i, g_i, a_i)$ is an EFSM. Here $g_i : \Sigma \rightarrow \mathsf{G}$ assigns to each $\alpha \in \Sigma$ the *guard* (formula), $g_i(\alpha)|_{X_i}$, which is evaluated using the binary values of the variables in X_i and guards all the transitions labeled with α. Also $a_i : X_{ii} \times \Sigma \rightarrow \mathsf{A}$ assigns to each pair of $\alpha \in \Sigma$ and $x \in X_{ii}$ an *action* $a_i(x, \alpha) : \mathbb{B}^{nh} \rightarrow \mathbb{B}$, which is a Boolean function and, upon the occurrence of α, results in the assignment $x := a_i(x, \alpha)(X_i)$. It is assumed that for the network of EFSMs, $(\mathbf{M}_i)_{i \in I}$, X is initialized to zero. □

Remark 4.3 Following [50], for every $\sigma \in \Sigma_{uc,i}$, $\sigma' \in \Sigma_{uo,i}$, and $x \in X_i$, $g_i(\sigma)$ is an always-true formula and $a_i(x, \sigma')$ is an identity function. Hence these cases are not mentioned explicitly. □

Definition 4.10 Let PDS (4.14) be defined on the finite field $\mathbb{F}_p^n$ and h, J, and for all $i \in I$, X_i, $\mathbf{M}_i$, and $x_i = (x_{ii}^h \cdots x_{ii}^1)$ be as in Definition 4.9. The network of EFSMs, $(\mathbf{M}_i)_{i \in I}$, implements the PDS if for every $i, j \in I, i \neq j$,

$k \in J$, $\mathbf{x} \in \mathbb{F}_p^n$ and $\sigma \in \Sigma$ it holds that

$$x_{ij}^k = x_{jj}^k$$
$$\wedge\ [x_i := \mathfrak{a}_i^\sigma(\mathbf{x}) \iff \forall k \in J.\ x_{ii}^k := a_i(x_{ii}^k, \sigma)(X_i)]$$
$$\wedge\ [\mathfrak{g}_i^\sigma(\mathbf{x}) = g_i(\sigma)|_{X_i}].$$ □

Proposition 4.6 Let $\mathbf{G}$, E, $\mathbf{S}$, ℓ, be as in Proposition 4.3 and the PDS (4.14), computed using Algorithm 4.1, replaces (4.5) and (4.6). If the network of EFSMs, $(\mathbf{M}_i)_{i\in I}$, implements the PDS (4.14), it holds that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

Algorithm 4.2 *Computation of guards and actions in EFSM framework from guard and updating functions of the DSDES framework:* Given the PDS (4.14) and p from Algorithm 4.1,

1. Determine h using Definition 4.9.

2. For each $i \in I$, define X_i and encode each x_i using $(x_{ii}^h \cdots x_{ii}^1)$ as in Definition 4.9.

3. For each guard polynomial $\mathfrak{g}_i^\sigma$ $(i \in I,\ \sigma \in \Sigma_{c,i})$, compute $g_i(\sigma)$ as a function of the Boolean variables in $Y = \{x_{ii}^k, x_{ij}^k \mid k \in J, j \in I, x_j \in \arg(\mathfrak{a}_i^\sigma)\}$ such that for every $x_i \in \mathbb{F}_p$ it holds that $\mathfrak{g}_i^\sigma(\mathbf{x}) = g_i(\sigma)|_Y$. For $i, j \in I$, take all encodings $(x_{ij}^1, \cdots, x_{ij}^h)$ of numbers $p, \cdots, 2^h$ as "don't care" cases, if there exists any.

4. For each updating polynomial $\mathfrak{a}_i^\sigma$ $(i \in I,\ \sigma \in \Sigma_{o,i})$, compute $a_i(x_{ii}^k, \sigma)$ as a function of the Boolean variables in $Z = \{x_{ii}^k, x_{ij}^k \mid k \in J, j \in I, x_j \in \arg(\mathfrak{a}_i^\sigma)\}$ such that for every $x_i \in \mathbb{F}_p$, it holds that $x_i := \mathfrak{a}_i^\sigma(\mathbf{x})$ if and only if $\forall k \in J.\ x_{ii}^k := a_i(x_{ii}^k, \sigma)(Z)$. For $i, j \in I$, take all encodings $(x_{ij}^1, \cdots, x_{ij}^h)$ of numbers $p, \cdots, 2^h$ as "don't care" cases, if there exists any.

5. For every $i \in I$ and $\sigma \in \Sigma_{uo,i}$, set $a_i(x,\sigma)$ equal to identity function ($x \in X_{ii}$), and for every $i \in I$ and $\sigma \in \Sigma_{uc,i}$, set $g_i(\sigma) = 1$, i.e. an always-true formula (see Remark 4.3). □

Proposition 4.6 insures that under a correct update of copy variables, the actions and guards computed by Algorithm 4.2 implement PDS (4.14), as summarized in the following result.

Corollary 4.2 If for every $i,j \in I, i \neq j$, it holds that $x_{ij}^k = x_{jj}^k$, then the actions and the guards computed by Algorithm 4.2 implement PDS (4.14). ■

The EFSM implementation of PDS (4.14) may be regarded as a PDS over Boolean field.

Corollary 4.3 Following Definition 4.9 and Algorithm 4.2, PDS (4.14) can be represented as the following PDS in state explicit form over Boolean field.

$$\forall i \in I, \forall x \in X_{ii}. \begin{cases} x := a_i(x,\sigma)(X_i) & ;\forall \sigma \in \Sigma_{o,i} \\ g_i(\sigma) - 1 = 0 & ;\forall \sigma \in \Sigma_{c,i} \end{cases} \tag{4.15}$$

Proof: Follows directly from Definition 4.8. ■

Since the PDS polynomials are computed based on a common finite field, there is a chance of initially introducing "extra" Boolean variables for some $\mathbf{S}_i$s. Such variables will evaluate to constants, i.e. always 0 or 1, and can be removed from the final implementation (see Example 4.5).

DSDESs versus EFSMs: Corollary 4.3 states that EFSM framework may be regarded as a special case of the DSDES framework. As a result, the advantages listed in [50] for EFSM framework over the frameworks based on "possible worlds" and "state estimates" are shared by the DSDES framework, too. In general EFSM framework is the one which is implemented in practice

and offers a bit-wise approach to communication rules and content. It is then worth clarifying why one may require to use the DSDES framework.

1. Comparing to Boolean variables, integer variables offer a more compact representation[6] of the information which improves the rigor of proofs and readability of the manipulations in the same way that decimal numbers may be advantageous over binary numbers.

2. Whereas a Boolean variable seldom bears a physical meaning singly, an integer variable is a label representing a (copy of a) state of the centralized supervisor. This fact improves the tractability of analysis and design (and possibly their computational efficiency [83]).

3. Communication-oriented simplifications can be done far more easily and insightfully when computing PDS (4.14) than its EFSM counterpart (4.15) directly from the centralized supervisor, as in [50]. An example of such "high-level" simplifications is to use structural symmetries to eliminate some external variables from the polynomials (see Example 4.3 and Definitions 5.3 and 5.4), which is easier to do in a larger (than Boolean) finite field.

4. It is often easier to study the system's evolution in an n-dimensional integer space (than in an nk-dimensional Boolean space) and to relate it with the algebraic and geometric properties of the associated polynomials and codes in larger (than Boolean) finite fields.

Therefore, a blend of the DSDES and EFSM frameworks enjoy their complementary merits.

[6]Often each integer variable is encoded using more than one Boolean variable.

Example 4.5 (Implementation of the updating and guard functions of Example 4.4 in EFSM framework): Corollary 4.2 lets us employ Algorithm 4.2 to compute the EFSM implementation of the PDS in Example 4.4. Observe that $p = 3$, thus, $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ each would require $h = \lceil log_2(3) \rceil = 2$ Boolean variables to implement their labels. Following Definition 4.9, for $i \in I$ we have $X_{ii} = \{x^2_{ii}, x^1_{ii}\}$. Two variables in general can represent four integers. However there are only three labels in $\mathbb{F}_3$, i.e. 0, 1, and 2. The unused label, 3, may be used arbitrarily to simplify the expressions (see "don't care" conditions in [73]). Actions and guards are listed in Table 4.3, where the complement of of a variable x is denoted by $\overline{x}$. Examples of computations follow.

- Consider the computation of $a_1(x^1_{11}, \alpha_1)$ and $a_1(x^2_{11}, \alpha_1)$. From Table 4.2 we know that $\mathfrak{a}^{\alpha_1}_1(\mathbf{x}) = 2(x_2 + 2)$, i.e. $a_1(x^1_{11}, \alpha_1)$ and $a_1(x^2_{11}, \alpha_1)$ do not depend on x^1_{11}, x^2_{11}, x^1_{13}, nor x^2_{13}. Whereas $\mathfrak{a}^{\sigma}_1(x_2)$ has been defined for $x_2 = 0$, $x_2 = 1$, and $x_2 = 2$, i.e on $\mathbb{F}_3$, to simplify the functional forms we arbitrarily map $x_2 = 3$ to $(x^2_{11}x^1_{11}) = (10)$ which yields $a_1(x^2_{11}, \alpha_1) = x^2_{12}\overline{x}^1_{12} + x^2_{12}x^1_{12} = x^2_{12}$ and $a_1(x^1_{11}, \alpha_1) = \overline{x}^2_{12}\overline{x}^1_{12}$.

- As a second example, consider the computation of $g_1(\beta)$. From Table 4.2 we know that $x_3 \notin \arg(\mathfrak{g}^{\beta}_{1,2})$, hence $g_1(\beta)$ do not depend on x^1_{13} and x^2_{13}. On the other hand $\mathfrak{g}^{\beta}_{1,2}(\mathbf{x})$ has been defined for neither $(x^2_{11}x^1_{11}, x^2_{12}x^1_{12}) = (11, --)$ nor $(x^2_{11}x^1_{11}, x^2_{12}x^1_{12}) = (--, 11)$, where "$-$" can take any Boolean values. To simplify the functional form of $g_1(\beta)$ we arbitrarily set the guard formula equal to 1 at $(00, 11)$, $(11, 00)$, $(11, 10)$, $(11, 11)$, and $(10, 11)$ and set it to 0 at the rest of the unspecified points. This results in the following.

$$
\begin{aligned}
g_1(\beta) = {} & [\overline{x}^2_{11}\overline{x}^1_{11}\overline{x}^2_{12}x^1_{12} + \overline{x}^2_{11}\overline{x}^1_{11}x^2_{12}x^1_{12}] + [\overline{x}^2_{11}x^1_{11}\overline{x}^2_{12}\overline{x}^1_{12} + x^2_{11}x^1_{11}\overline{x}^2_{12}\overline{x}^1_{12}] + \\
& [x^2_{11}x^1_{11}x^2_{12}x^1_{12} + x^2_{11}x^1_{11}x^2_{12}\overline{x}^1_{12} + x^2_{11}\overline{x}^1_{11}x^2_{12}x^1_{12} + x^2_{11}\overline{x}^1_{11}x^2_{12}\overline{x}^1_{12} +] \\
= {} & \overline{x}^2_{11}\overline{x}^1_{11}x^1_{12} + x^1_{11}\overline{x}^2_{12}\overline{x}^1_{12} + x^2_{11}x^2_{12}
\end{aligned}
$$

Table 4.3: Actions and guards in EFSM framework

$a_1(x_{11}^2, \alpha_1) = x_{12}^2$
$a_1(x_{11}^1, \alpha_1) = \overline{x}_{12}^2 \overline{x}_{12}^1$
$a_1(x_{11}^2, \beta) = x_{11}^1$
$a_1(x_{11}^1, \beta) = \overline{x}_{11}^2 \overline{x}_{11}^1$
$a_2(x_{22}^2, \alpha_2) = \overline{x}_{22}^2 [\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1]$
$a_2(x_{22}^1, \alpha_2) = x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2$
$a_2(x_{22}^2, \beta) = x_{22}^1$
$a_2(x_{22}^1, \beta) = \overline{x}_{22}^2 \overline{x}_{22}^1$
$a_3(x_{33}^2, \alpha_3) = 0$
$a_3(x_{33}^1, \alpha_3) = \overline{x}_{33}^2 \overline{x}_{33}^1 = \overline{x}_{33}^1$
$g_1(\alpha_1) = x_{11}^2 \overline{x}_{12}^2 + \overline{x}_{11}^2 x_{12}^2 + x_{11}^1 x_{12}^1 + \overline{x}_{11}^2 \overline{x}_{11}^1 \overline{x}_{12}^1$
$g_1(\beta) = x_{11}^1 \overline{x}_{12}^2 \overline{x}_{12}^1 + \overline{x}_{11}^2 \overline{x}_{11}^1 x_{12}^1 + x_{11}^2 x_{12}^2$
$g_2(\alpha_2) = x_{21}^1 \overline{x}_{22}^2 + \overline{x}_{21}^2 x_{22}^1 + \overline{x}_{21}^1 x_{22}^2 + x_{21}^2 \overline{x}_{22}^1 + x_{33}^1$
$g_2(\beta) = x_{21}^1 \overline{x}_{22}^2 \overline{x}_{22}^1 + \overline{x}_{21}^2 \overline{x}_{21}^1 x_{22}^1 + x_{21}^2 x_{22}^2$
$g_3(\alpha_3) = \overline{x}_{33}^2 \overline{x}_{33}^1 = \overline{x}_{33}^1$

- Since $x_{33}^2 = 0$, this variable is not really required to represent the system's evolution.

4.5 A note on computational complexity

It is worth mentioning some computational facts about the DSDES and EFSM frameworks. Let $|R| = m$ and $|\Sigma| = m'$. Thus the centralized supervisor may have at most $m'm^2$ state transitions. Observe that the computation of an ALM as a Latin hypercube is linear time, i.e. $O(m)$ [50]. Also for an event σ, computations of $\mathcal{G}(\sigma)$ and $\mathcal{A}(\sigma, .)$ are linear searches in the number of states and transitions, i.e. they are $O(m)$ and $O(m'm^2)$, respectively. Moreover, computation of polynomials for updating and guard functions using (4.11) and (4.10), and for actions and guards using Algorithm 4.2, is polynomial in time. Thus obtaining the PDS representation of a DSDES is polynomial in time.

However, simplifying polynomials and ALMs are in general computationally expensive though it can be systematically performed using computational-algebraic tools [83].

To study space complexity, assume that a finite-state machine (FSM) representation for a typical decentralized supervisor i is computed using natural projection of the centralized supervisor. Accordingly, this representation may have up to 2^m states and $(m'(2^m)^2 = m'4^m)$ distinct transitions. Using Boolean variables to encode each transition would require $\lceil 2m + \log_2 m' \rceil$ memory units to store the source and target states and the label of each transition[7]. Thus, in the worst case $N_1 = nm'4^m\lceil 2m + \log_2 m' \rceil$ units of memory are required to store all n supervisors' information. To do the same analysis for an EFSM obtained from a DSDES, let us use the CNF standard form to write Boolean formulas or functions[8]. Observe that $n \log_2 m$ Boolean variables are required to encode the states of the centralized supervisor[9]. These variables give rise to $(2^{n \log_2 m} = m^n)$ minterms, each having a length of[10] $n \log_2 m$. There are at most $2^{n \log_2 m} = m^n$ such minterms forming each formula with $(2^{n \log_2 m} - 1) = m^n - 1$ disjunction operators between every two neighboring minterms[11]. As a result, each formula (or function) would take up to $\lceil nm^n \log_2 m + m^n - 1 \rceil$ memory units. There are m' formulas associated with guards and (at most) $m'n \log_2 m$ functions corresponding to actions. Thus, the EFSM consumes up to $N_2 = \lceil m' + m'n \log_2 m \rceil \lceil nm^n \log_2 m + m^n - 1 \rceil$ units of memory to store the control-related information of the closed-loop

[7]Here the comparison is made with this graph encoding which is not necessarily the most efficient one. For other encoding schemes a separate analysis should be done along the same lines mentioned here.

[8]Guards and actions are formally computed using minterms [49]. CNF stands for "conjunctive normal form." For definitions of CNF and minterm see [73], for example.

[9]In fact, m should be replaced with p here, but this does not change the asymptotic analysis.

[10]Each minterm lists all variables in the simple or negated forms.

[11]Conjunctive operators need not be stored since the CNF form assigns *a priori* fixed places to them.

system. Considering the asymptotic behavior of the two numbers we have $N_1 = O(nm4^m)$ and $N_2 = O(n^2 m^n \log^2 m)$. Therefore, for a large m, the EFSM shows an improvement in the number of memory cells to store the supervisor's information. Finally, notice that comparing to a graph-based representation, which is common to FSMs, the formula-based representation of a system's dynamical evolution makes it more structured to both observe the system's behavior and search for associated control decisions. This fact, on top of the above space reduction, may improve online computational issues, which are not studied here.

4.6 Conclusion

This chapter introduces the (distributed) SDES framework for the synthesis of communicating decentralized supervisors out of a given centralized supervisor, and to study communication among them. The DSDES framework encompasses EFSM framework as its special implementation-oriented case. The design in DSDES framework is more tractable and insightful and the two frameworks enjoy their complementary facets. The central tools employed by a DSDES are its ALM and its updating and guard functions which represent the control-related structure of the centralized supervisor in a distributed way. When recast as a polynomial dynamical system over a field, a DSDES becomes amenable to standard mathematical analysis tools. Two algorithms are introduced to compute such a polynomial representation and its EFSM implementation. The chapter ends with a complexity analysis of the associated computations.

Chapter 5

Modeling and Synthesis of Communication within SDES Framework

5.1 Introduction

For a distributed SDES (DSDES) communication naturally appears to inform a supervisor of the labels assigned by other supervisors, which it uses to reevaluate its guard and updating functions. The communication problem is defined as designing "communication events" to help the decentralized supervisors have enough "control-related" information to enforce the given specification optimally. Once a DSDES is represented as a PDS, communication is characterized and solved based on the interconnections between polynomial equations defined over algebraic structures. This chapter employs the PDS representations of Chapter 4 to derive what is formally defined as "communication policy," which is simply the collection of rules for communication among supervisors. The communication policy is itself divided into an

"information policy" and a "routing policy," corresponding to "logical informational dependency of supervisors on each other" and "rules to physically exchange this required information in the presence of unreliable communication channels and network topology," respectively. Solutions in the form of information policies are derived for communication of state information and observed events. Design of routing policies is left for future work.

Section 5.2 formalizes communication among supervisors through the introduction of communication events, defining the notion of a communication policy, and deriving five communication policies. Section 5.3 comes up with a partitioning of the class of solutions to the communication problem. In Section 5.4 some relations between behavioral properties of a DSDES and its associated state representation are established and a few preliminary results on communication-oriented state realizations are given.

5.2 Communication as reevaluation of guard and updating functions

In DSDES framework, communication among decentralized supervisors is required for reevaluation of their guard and updating functions. For example assume that the vector of values after a string s is observed is $\mathbf{v} := \mathcal{A}(s, \mathbf{0})$. Then $\sigma \in \Sigma_i$ is enabled at s if and only if $\mathbf{v} \in \mathcal{G}_i(\sigma)$. To determine if this is the case, $\mathbf{S}_i$ may need to receive the value v_j, for some $j \neq i$, from $\mathbf{S}_j$. When σ is taken, $\mathbf{S}_i$ updates v_i with the value $\mathcal{A}_i(\sigma, \mathbf{v})$. Again, to correctly evaluate $\mathcal{A}_i(\sigma, \mathbf{v})$, $\mathbf{S}_i$ may need to receive the value v_j, for some $j \neq i$, from $\mathbf{S}_j$.

Once a DSDES is formulated as a PDS, $\mathbf{S}_i$'s *informational dependencies* appear in the functional forms of its updating and guard polynomials and on

their dependencies on external variables. Thus, computation of the communication, either to correctly evaluate the effect of the occurrence of an event on its observing supervisor's variable update, or to correctly determine where an event is enabled, depends on the properties of the polynomials (for updating and guard functions, respectively) and their arguments. In the following we formalize the communication problem and propose six solutions to it inspired by a study of polynomial forms. Advanced characterizations of the problem based on finite-field tools is left for future work.

Throughout this section, let the n supervisors be connected through a network of "ideal" control channels, i.e. data is instantly transmitted without any losses. To ensure the connectedness of the network, we assume that disabling a controllable event affects none of communication-related events, which are defined next. We assume a DSDES is given in the PDS form (4.14), that its current state is $\mathbf{x}$, and that the observation and control tasks of each supervisor are represented as polynomials over a finite field $\mathbb{F}_p^n$ and implemented in EFSM framework as in (4.15).

5.2.1 Communication-related events

Denote a communication-related event which is sent from $\mathbf{S}_j$ to $\mathbf{S}_i$ by the subscript indices ji and assume that it is observable by both supervisors and controllable by $\mathbf{S}_j$. Although $\Sigma_{o,j}$, $\Sigma_{c,j}$, and $\Sigma_{o,i}$ should be enlarged by these new events, to keep the notations simpler we avoid introducing new sets and assume that this fact is clear from the context.

A first observation is that the event-driven nature of DESs limits the release of the system-related information to the occurrence of observable events such that only the supervisors, who can observe them, gain information about the system's evolution. Therefore, it is plausible to rely communication, as an

information-providing mean, on the observation of the occurrence of events, i.e. issuing a communication event would follow the occurrence of an observable event. Accordingly, the semantics of every communication event is defined by a mapping, whose domain is a subset of observable events and whose codomain is an information-containing set. The communication problem then reduces to designing these maps exactly, as defined later.

Since the system-related information is captured by Boolean variables in X, the content transferred by a communication event, i.e. the elements of its image set, is taken either from these variables or from *constant* messages, whose definition follows.

Definition 5.1 Let H be a finite set of Boolean variables such that $H \cap X = \emptyset$. A *constant message* is a finite-length word which is encoded by Boolean variables in H, may be sent by $\mathbf{S}_j$ to $\mathbf{S}_i$, and whose semantics is known to $\mathbf{S}_j$ and $\mathbf{S}_i$, where $i, j \in R, i \neq j$. □

The semantics of a constant message might be interpreted in different ways such as a handshaking between the two supervisors, a request for issuing a communication event, an update in a fixed form such as toggling the values of variables, etc.

To model the communication, we distinguish two types of communication events which are referred to as $\mathscr{I}$ and $\mathscr{R}$ events. Correspondingly, this classification divides the communication design into two levels of *information exchange* and *routing*, which, respectively address what information needs to be exchanged mutually among supervisors and how these exchanges can be performed using the available communication channels. These events are introduced next.

$\mathscr{I}$ events: An event $\mathscr{I}_{ji}$ transfers the private information of $\mathbf{S}_j$ or constant messages to $\mathbf{S}_i$. For $j \in I$, let $\Sigma_{\mathscr{I},j} \subseteq (\Sigma_{o,j} \cup \bigcup_{k \in I, k \neq j} \{\mathscr{I}_{kj} \mid \mathscr{I}_{kj} \text{ is defined}\})$

denote the set of observable events by $\mathbf{S}_j$, after which $\mathbf{S}_j$ issues an $\mathscr{I}$ event. Correspondingly, for $i, j \in I, i \neq j$, define $\mathscr{I}_{ji} : \Sigma_{\mathscr{I},j} \rightarrow pwr(X_{jj} \cup H)$ as a function which associates to an event in $\Sigma_{\mathscr{I},j}$, a piece of information stored by Boolean variables in X_{jj} or H according to a rule which will become specified upon the design of the communication. Whereas the definition of $\Sigma_{\mathscr{I},j}$ allows the firing of an $\mathscr{I}$ event after another $\mathscr{I}$ event, circular definitions should be avoided. Since I is finite, there is a finite number of distinct $\mathscr{I}$ events and this, together with the finiteness of X and H, guarantee that the information is exchanged in a finite number of communication steps. Once received by $\mathbf{S}_i$, $\mathscr{I}_{ji}(.)$ provides it with the updated copies of the variables in its image set, i.e.

$$\forall \mathbf{x} \in \mathbb{F}_p^n,\ \forall i, j \in I, i \neq j,\ \forall k \in J,\ \forall x_{jj}^k \in X_{jj}, \forall \sigma \in \Sigma_{\mathscr{I},j}.$$
$$x_{jj}^k \in \mathscr{I}_{ji}(\sigma) \Longrightarrow x_{ij}^k := x_{jj}^k. \quad (5.1)$$

The way constant messages affect the receiver or its information depend on their semantics, which is specified by the communication design.

To implement the exchange of information, as prescribed by $\mathscr{I}$ events, network constraints and channel characteristics should be considered, too. If the network is strongly connected, $\mathscr{I}$ events, on their own, can implement the exchange of information. However, it may be the case that some pairs of supervisors do not have direct communication links between them, or a supervisor $\mathbf{S}_k$ may have the updated copies of $\mathbf{S}_j$'s variables and these copies can be sent to $\mathbf{S}_i$ from $\mathbf{S}_k$, rather than $\mathbf{S}_j$, in a possibly more reliable or economic way. To account for such "indirect" data transfers, $\mathscr{R}$ events can be introduced to replace some $\mathscr{I}$ events.

$\mathscr{R}$ events: For $j \in I$ let

$\Sigma_{\mathscr{R},j} \subseteq \left(\Sigma_{o,j} \cup \bigcup_{k \in I, k \neq j} \{\mathscr{I}_{kj} |\ \mathscr{I}_{kj} \text{ is defined}\} \cup \bigcup_{k \in I, k \neq j} \{\mathscr{R}_{kj} |\ \mathscr{R}_{kj} \text{ is defined}\}\right)$

denote the set of observable events by $\mathbf{S}_j$, after which $\mathbf{S}_j$ issues an $\mathscr{R}$ event. Correspondingly, for $i, j \in I, i \neq j$, define $\mathscr{R}_{ji} : \Sigma_{\mathscr{R},j} \rightarrow pwr(X_{cj} \cup H)$ as a function which associates to an event in $\Sigma_{\mathscr{R},j}$, a piece of information stored by Boolean variables in X_{cj} or a constant message in H according to a rule which will become specified upon the design of communication. Similar to $\mathscr{I}$ events, the number of $\mathscr{R}$ events is finite and their circular definitions should be avoided. Upon its receipt by $\mathbf{S}_i$, $\mathscr{R}_{ji}$ provides the updated values such that at every state we have

$$\forall \mathbf{x} \in \mathbb{F}_p^n,\ \forall i, j, l \in I, i \neq j, l \neq j, i,\ \forall k \in J,\ \forall \sigma \in \Sigma_{\mathscr{R},j}, \forall x_{jl}^k \in X_{cj}.$$
$$x_{jl}^k \in \mathscr{R}_{ji}(\sigma) \Longrightarrow x_{il}^k := x_{jl}^k \quad (5.2)$$

Constant messages affect the receiver based on their semantics, which is specified by the communication design.

Inherently, the definitions for each of the events $\mathscr{I}_{ji}(\sigma)$, and $\mathscr{R}_{ji}(\sigma)$ determine *who* (i.e. $\mathbf{S}_j$) sends *what* (a subset of H, X_{jj}, or X_{cj}) to *whom* (i.e. $\mathbf{S}_i$). Implicitly, their dependency on an event as their argument, bears a notion of "logical" time which roughly specifies the soonest moment at which the communication can start. This might be useful in the study of time-related issues. Also notice that whereas the dependency on observed events makes the communication *event-triggered*, its content, which is a (Boolean)-variable representation of the history of the system's behavior, is *state-based*.

Although in practice $\mathscr{I}_{ji}$ and $\mathscr{R}_{ji}$ can be implemented using one communication event from $\mathbf{S}_j$ to $\mathbf{S}_i$, the distinction between them is a theoretically insightful divide-and-conquer approach to design communication. Accordingly, $\mathscr{I}$ events take an abstract viewpoint to reflect which parts of private information should be exchanged and $\mathscr{R}$ events may be required to implement this

exchange based on routing considerations, channel restrictions, and network structure. Moreover, addition of constant messages, as means of conveying system-related information which do not explicitly depend on the states of the system, on top of state-dependent information captured by Boolean variables, can model all kinds of information which may be exchanged among the supervisors. Furthermore, since $\mathscr{R}$ events can transfer the communication load of some channels to others, they may be potentially used to optimize the communication in some specified sense.

5.2.2 Communication policies

We can now formalize the "communication problem" within the DSDES framework. This is conceptually motivated by the design of the two types of events.

Definition 5.2 Let PDS (4.14) and its implementation (4.15) be given. A *handshaking policy*, an *information policy*, and a *routing policy* for (4.14) and (4.15) are respectively equivalent to designing (H, constant messages), $\left(\Sigma_{\mathscr{I},j}, \mathscr{I}_{ji}(.)\right)$, and $\left(\Sigma_{\mathscr{R},j}, \mathscr{R}_{ji}(.)\right)$ for every $i, j \in I$. Design of a *communication policy* is equivalent to the design of all three policies. □

Problem 5.1 *Communication Problem in the DSDES framework:* Associated with Problem 4.2 and Proposition 4.3, let the PDS (4.14) represent the DSDES and the network of EFSMs (4.15) implement the PDS. Find a communication policy such that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. □

Measuring the communication content: Under a communication policy, it is possible to measure the size of the communication for quantitative purposes.

Lemma 5.1 For $\sigma \in \Sigma$ and $i, j \in I, j \neq i$, let $\mathscr{I}_j[\sigma]$ and $\mathscr{R}_j[\sigma]$ denote respectively the set of all $\mathscr{I}_{ji}$ and $\mathscr{R}_{ji}$ events which happen after the observation of

σ and before the occurrence of any other event in Σ, under a given communication policy. Denote the size of the image set of a communication event θ by $|\theta|$. Then along a trajectory $s \in \Sigma^*$, where $P_j(s) = \sigma_1 \cdots \sigma_z$ and z is a finite natural number, the amount of communication initiated by $\mathbf{S}_j$ would be as follows.

$$\Delta_j(s) = \sum_{k=1}^{z} \Big(\sum_{\mathscr{I} \in \mathscr{I}_j[\sigma_k]} |\mathscr{I}| + \sum_{\mathscr{R} \in \mathscr{R}_j[\sigma_k]} |\mathscr{R}| \Big) \tag{5.3}$$

Proof: Follows simply by measuring the sizes of the image sets of communication events. ■

Communication is the third means, on top of observation and control, by which decentralized supervisors confine a plant's behavior within given specifications. By presenting the system information of the centralized supervisor in a distributed way and putting it into polynomial equations, the DSDES framework provides a general, flexible, and systematic framework for analysis and synthesis of decentralized supervisors. Within this framework, supervisors' private variables form the largest set of system information, owned by a given state representation of the centralized supervisor's behavior. Communication then appears naturally to help each supervisor reevaluate its guard and updating functions by providing the values of the external variables on which they depend. Computation of communication, as a solution to Problem 5.1, deserves a separate work of its own, which is based on the study of algebraic structures such as finite fields. However, to illustrate the applicability of the proposed approach, in the next subsection we take the first steps by proposing two communication policies and verifying their correctness.

5.2.3 Two simple communication policies

This subsection studies two simple communication policies, which are intuitively proposed by the structured way of representing the system information in the DSDES framework. We justify this fact through explaining the motivation behind the communication policies and presenting their informal descriptions which are then followed by their formal definitions and verifications. As a first study we focus only on the information policies and assume that the routing policies are void, i.e. $\mathscr{R}_{ji} = \emptyset, (i, j \in I)$. As mentioned in Subsection 5.2.1, if the network is strongly connected, the information policy itself can be reasonably implemented in practice. Whereas the second policy prescribes the exchange of fewer bits, neither policy is claimed to be content-wise minimal in the sense of (5.3). They are mainly derived to illustrate how the DSDES framework can found the basis for systematic modeling and computation of communication among supervisors. Advanced solution techniques are left for future.

The motivation behind the communication policies and their description: When $\mathbf{S}$ is represented as PDS (4.14), a supervisor $\mathbf{S}_i$ depends on $\mathbf{S}_j$'s private information if and only if x_j appears as the argument of one of $\mathbf{S}_i$'s guard or updating functions. Therefore it seems plausible that if $\mathbf{S}_i$ receives the last updated value of such x_j's, it can reevaluate its guard and updating functions correctly. With this intuition, a communication policy may be synthesized as follows: When the system starts its evolution, upon the occurrence of an event, first all supervisors $\mathbf{S}_j$, which observe it, update their x_j's. Next, every such $\mathbf{S}_j$, whose x_j is one of the arguments of a polynomial of an $\mathbf{S}_i$ $(i \neq j)$ sends the value of x_j to such an $\mathbf{S}_i$. Back to the EFSM implementation, every variable $x \in X_{jj}$ whose copy appear in X_{ij} should be sent upon update, a policy which is referred to as *policy 1*. However, intuitively it

suffices that $\mathbf{S}_j$ send $\mathbf{S}_i$ the subset of these variables which have *changed* upon the update. This second policy is referred to as *policy 2*. In any case, $\mathbf{S}_i$ then utilizes the received bits to reevaluate either its Boolean variables $y \in X_{ii}$, which implement x_i, or its guards. These policies, which assume no constant messages, can be formally defined in the following.

Definition 5.3 For the PDS (4.14) and its EFSM implementation (4.15), *communication policy 1* is defined as follows: For every $i, j \in I, i \neq j$, we have

$$H_{ji}=\emptyset \ \wedge \ \Sigma_{\mathscr{R}_{ji}}=\emptyset \ \wedge \ \Sigma_{\mathscr{I},j}=\Sigma_{o,j}$$
$$\wedge \ [\forall \sigma \in \Sigma_{o,j}. \ \mathscr{I}_{ji}(\sigma)=\{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j)$$
$$\wedge \ [\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}. \ x^k_{jj} \in \arg(a_i(x', \sigma')) \ \vee \ x^k_{jj} \in \arg(g_i(\sigma'))]\}]. \ \square$$

Definition 5.4 For the PDS (4.14) and its EFSM implementation (4.15), *communication policy 2* is defined as follows: For every $i, j \in I, i \neq j$, we have

$$H_{ji}=\emptyset \ \wedge \ \Sigma_{\mathscr{R}_{ji}}=\emptyset \ \wedge \ \Sigma_{\mathscr{I},j}=\Sigma_{o,j}$$
$$\wedge \ [\forall \sigma \in \Sigma_{o,j}. \ \mathscr{I}_{ji}(\sigma)=\{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j)$$
$$\wedge \ \hat{x}^k_{jj} \neq x^k_{jj} \wedge [\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}. \ x^k_{jj} \in \arg(a_i(x', \sigma')) \vee x^k_{jj} \in \arg(g_i(\sigma'))]\}]. \square$$

Proposition 5.1 *A solution to Problem 5.1:* Associated with Problem 5.1, if the network of supervisors is strongly connected with lossless channels and if communication is instantaneous, each of communication policies 1 and 2 insures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

Remark 5.1 Definitions 5.3 and 5.4 imply that the communication content may be reduced by eliminating from polynomials as many external variables as possible (see Remark 4.2). □

Example 5.1 (Using communication policies 1 and 2 to compute communication for Example 4.5): For the system in Example 4.2 with guards and actions in Table 4.3, two sets of communication events are derived

Table 5.1: Communication amongst supervisors based on policy 1

$\mathscr{I}_{12}(\alpha_1) = \mathscr{I}_{12}(\beta) = \{x_{11}^1, x_{11}^2\}$ $\mathscr{I}_{13}(\alpha_1) = \mathscr{I}_{13}(\beta) = \emptyset$
$\mathscr{I}_{21}(\alpha_2) = \mathscr{I}_{21}(\beta) = \{x_{22}^1, x_{22}^2\}$ $\mathscr{I}_{23}(\alpha_2) = \mathscr{I}_{23}(\beta) = \emptyset$
$\mathscr{I}_{31}(\alpha_3) = \emptyset$ $\mathscr{I}_{32}(\alpha_3) = \{x_{33}^2\}$

using communication policies 1 and 2 (Definitions 5.3 and 5.4) and listed in Tables 5.1 and 5.2, respectively. Examples of the computations are as follows.

- In Table 4.3, observe that $x_{22}^1, x_{22}^2 \in \arg(a_1(x_{11}^1, \alpha_1))$. As a result, communication policy 1 requires that $\mathbf{S}_2$ send these two variables to $\mathbf{S}_1$ when they get updated. On the other hand, the two variables are updated when either α_2 or β happens. Thus, as shown in Table 5.1, we have $\mathscr{I}_{21}(\alpha_2) = \mathscr{I}_{21}(\beta) = \{x_{22}^1, x_{22}^2\}$.

- Following the previous observation, communication policy 2 requires that each of x_{22}^1 and x_{22}^2 be sent only when its value changes. As for α_2 observe that $a_2(x_{22}^1, \alpha_2) = x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2$, i.e. once α_2 happens, x_{22}^1 changes if either $x_{22}^1 = 0$ and $x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2 = 1$ or $x_{22}^1 = 1$ and $x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2 = 0$. This can be shortly denoted by the exclusive-or function of two Boolean variables y and z which is defined as $y \oplus z = y\overline{z} + \overline{y}z$. Therefore if $(x_{22}^1 \oplus [x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2])$ is logically true, $x_{22}^1 \in \mathscr{I}_{21}(\alpha_2)$. Similarly if $\left(x_{22}^2 \oplus (\overline{x}_{22}^2 [\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1])\right)$ is true, $x_{22}^2 \in \mathscr{I}_{21}(\alpha_2)$. The two conditions result in 4 different possibilities in Table 5.2.

- To see how the network of 3 communicating supervisors evolves, let us consider the execution of a sample string $s = \alpha_1\beta \in \Sigma^*$. Since X is

Table 5.2: Communication amongst supervisors based on policy 2

$$\mathscr{I}_{12}(\alpha_1) = \begin{cases} \{x_{11}^1, x_{11}^2\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{12}^2 \overline{x}_{12}^1]) \wedge (x_{11}^2 \oplus x_{12}^2) \\ \{x_{11}^1\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{12}^2 \overline{x}_{12}^1]) \wedge \neg(x_{11}^2 \oplus x_{12}^2) \\ \{x_{11}^2\} & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{12}^2 \overline{x}_{12}^1]) \wedge (x_{11}^2 \oplus x_{12}^2) \\ \emptyset & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{12}^2 \overline{x}_{12}^1]) \wedge \neg(x_{11}^2 \oplus x_{12}^2) \end{cases}$$

$$\mathscr{I}_{12}(\beta) = \begin{cases} \{x_{11}^1, x_{11}^2\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge (x_{11}^2 \oplus x_{11}^1) \\ \{x_{11}^1\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge \neg(x_{11}^2 \oplus x_{11}^1) \\ \{x_{11}^2\} & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge (x_{11}^2 \oplus x_{11}^1) \\ \emptyset & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge \neg(x_{11}^2 \oplus x_{11}^1) \end{cases}$$

$$\mathscr{I}_{13}(\alpha_1) = \mathscr{I}_{13}(\beta) = \emptyset$$

$$\mathscr{I}_{21}(\alpha_2) = \begin{cases} \{x_{22}^1, x_{22}^2\} & ; \text{ if } (x_{22}^1 \oplus [x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2]) \\ & \quad \wedge (x_{22}^2 \oplus [\overline{x}_{22}^2 (\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1)]) \\ \{x_{22}^1\} & ; \text{ if } (x_{22}^1 \oplus [x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2]) \\ & \quad \wedge \neg(x_{22}^2 \oplus [\overline{x}_{22}^2 (\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1)]) \\ \{x_{22}^2\} & ; \text{ if } \neg(x_{22}^1 \oplus [x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2]) \\ & \quad \wedge (x_{22}^2 \oplus [\overline{x}_{22}^2 (\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1)]) \\ \emptyset & ; \text{ if } \neg(x_{22}^1 \oplus [x_{22}^2 x_{21}^1 + \overline{x}_{22}^1 x_{21}^2]) \\ & \quad \wedge \neg(x_{22}^2 \oplus [\overline{x}_{22}^2 (\overline{x}_{21}^2 \overline{x}_{22}^1 + x_{21}^1)]) \end{cases}$$

$$\mathscr{I}_{21}(\beta) = \begin{cases} \{x_{22}^1, x_{22}^2\} & ; \text{ if } (x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge (x_{22}^2 \oplus x_{22}^1) \\ \{x_{22}^1\} & ; \text{ if } (x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge \neg(x_{22}^2 \oplus x_{22}^1) \\ \{x_{22}^2\} & ; \text{ if } \neg(x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge (x_{22}^2 \oplus x_{22}^1) \\ \emptyset & ; \text{ if } \neg(x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge \neg(x_{22}^2 \oplus x_{22}^1) \end{cases}$$

$$\mathscr{I}_{23}(\alpha_1) = \mathscr{I}_{23}(\beta) = \emptyset$$

$$\mathscr{I}_{31}(\alpha_3) = \emptyset$$

$$\mathscr{I}_{32}(\alpha_3) = \{x_{33}^1\} \ (x_{33}^1 \oplus \overline{x}_{33}^1 \text{ is always true})$$

initialized to zero, from Table 4.3 we have $g_1(\alpha_1) = 1$, therefore α_1 can happen. Upon its execution, $(x_{11}^2 x_{11}^1)$ is updated to (01), leading the PDS to state $(1, 0, 0) \in \ell(r_1)$. By communication policy 1, $\mathscr{I}_{12}(\alpha_1) = \{x_{11}^1, x_{11}^2\}$ and following (5.1), it results in $(x_{21}^2 x_{21}^1) = (01)$. Also, since only the value of x_{11}^1 has changed, the conditions of policy 2 in Table 5.2 require that $\mathscr{I}_{12}(\alpha_1) = \{x_{11}^1\}$, upon which $x_{21}^1 = 1$. This, together with the fact that x_{11}^2 has not changed (and so $x_{21}^2 = x_{11}^2 = 0$), again yields $(x_{21}^2 x_{21}^1) = (01)$. The new values lead to $g_1(\beta) = g_2(\beta) = 1$. Once β happens, both $\mathbf{S}_1$ and $\mathbf{S}_2$ update their private variables so that

$(x_{11}^2 x_{11}^1) = (10)$ and $(x_{22}^2 x_{22}^1) = (01)$, leading the PDS to state $(2, 1, 0) \in \ell(r_0)$. By communication policy 1 we have $\mathscr{I}_{12}(\beta) = \{x_{11}^1, x_{11}^2\}$ and $\mathscr{I}_{21}(\beta) = \{x_{22}^1, x_{22}^2\}$ resulting in $(x_{21}^2 x_{21}^1) = (10)$ and $(x_{12}^2 x_{12}^1) = (01)$. By policy 2 we have $\mathscr{I}_{12}(\beta) = \{x_{11}^1, x_{11}^2\}$ and $\mathscr{I}_{21}(\beta) = \{x_{22}^2\}$ which, together with the fact that x_{22}^2 has not changed (and so $x_{12}^2 = x_{22}^2$), again result in $(x_{21}^2 x_{21}^1) = (10)$ and $(x_{12}^2 x_{12}^1) = (01)$.

- As an application of (5.3), for $s = \alpha_1\beta$ we have $P_1(s) = \alpha_1\beta$, $P_2(s) = \beta$, and $P_3(s) = \emptyset$. For both policies $\mathscr{I}_1[\alpha_1] = \{\mathscr{I}_{12}(\alpha_1)\}$, $\mathscr{I}_1[\beta] = \{\mathscr{I}_{12}(\beta)\}$, and $\mathscr{I}_2[\beta] = \{\mathscr{I}_{21}(\beta)\}$. Policy 1 yields $|\mathscr{I}_{12}(\alpha_1)| = |\mathscr{I}_{12}(\beta)| = |\mathscr{I}_{21}(\beta)| = 2$, resulting in $\Delta_1 = 2 + 2 = 4$ and $\Delta_2 = 2$. For policy 2 we have $|\mathscr{I}_{12}(\alpha_1)| = |\mathscr{I}_{21}(\beta)| = 1$ and $|\mathscr{I}_{12}(\beta)| = 2$, resulting in $\Delta_1 = 1 + 2 = 3$ and $\Delta_2 = 1$. Hence, policy 2 reduces the communication content by 2 bits. $\Diamond$

5.2.4 Communication policies which prescribe less communication

Comparing to communication policy 1, communication policy 2 reduces the communication content by communicating only those private variables whose values change upon update. However, not every change in the variables of a function would change the value of that function. As a result, it is only required to communicate only those changed variables whose new value would change the value of the guard or updating function (or guards and actions) which depend on these variables. The next example illustrates this point.

Example 5.2 Let $I = \{1, 2\}$, $\Sigma_{o,1} = \Sigma_{c,1} = \{\alpha_1\}$, $\Sigma_{o,2} = \Sigma_{c,2} = \{\alpha_2\}$, and $\Sigma = \Sigma_1 \cup \Sigma_2$. Parts (a) and (b) of Fig. 5.1 shows a plant $\mathbf{G}$ and a specification $\mathbf{S}$, which is also a centralized supervisor. Shown in part (c) is the

graphical representation of an ALM for $\mathbf{S}$ where $\ell(r_0) = \{00\}$, $\ell(r_1) = \{10\}$, $\ell(r_2) = \{20\}$, $\ell(r_3) = \{30\}$, and $\ell(r_4) = \{31\}$. Updating functions are listed in Table 5.3 and guard functions are as follows.

$\mathcal{G}_1(\alpha_1) = \ell(r_0) \cup \ell(r_1) \cup \ell(r_2) \cup \ell(r_3) = \{00, 10, 20, 30\}$,

$\mathcal{G}_2(\alpha_2) = \ell(r_3) \cup \ell(r_4) = \{30, 31\}$

Using Algorithm 4.1 we have $\mathbf{x} = (x_1, x_2)$, $p = 5$. To come up with simplified polynomials, we extend arbitrarily $\mathcal{A}_1(\alpha_1, \mathbf{v})$ to map $(x_1, x_2) = (0, j)$ to $(1, j)$, $(x_1, x_2) = (1, j)$ to $(2, j)$, $(x_1, x_2) = (2, j)$ to $(3, j)$, where $j \in \{1, 2, 3, 4\}$. Similarly, we extend $\mathcal{A}_2(\alpha_2, \mathbf{v})$ to map $(x_1, x_2) = (j, 0)$ to $(j, 1)$, where $j \in \{0, 1, 2, 4\}$. Also, we assume that α_1 is enabled at $(x_1, x_2) = (4, 0)$ and α_2 is enabled at $(x_1, x_2) \in \{(3, 2), (3, 3), (3, 4)\}$ [1]. Accordingly, guard and updating functions on $\mathbb{F}_5^2$ are as in Table 5.4. Observe that neither $\mathfrak{a}_1^{\alpha_1}$ nor $\mathfrak{a}_2^{\alpha_2}$ depends on external variables, i.e. communication is required only to help $\mathbf{S}_1$ and $\mathbf{S}_2$ reevaluate their guard functions, correctly.

An inspection of $\mathfrak{g}_2^{\alpha_2}$ reveals that it is only a function of x_1 and its value is 1 at $x_1 = 3$ and 0 elsewhere. This implies that unless x_1 changes between the two sets $\{3\}$ and $\{0, 1, 2, 4\}$, its new value does not change the value of $\mathfrak{g}_2^{\alpha_2}$ and thus need not be sent by $\mathbf{S}_1$ to $\mathbf{S}_2$. Knowing the fact that $\mathfrak{g}_2^{\alpha_2}$ depends solely on x_1, this would reduce the number of times $\mathscr{I}_{12}(\alpha_1)$ should be issued. Such an observation can be made for $\mathfrak{g}_1^{\alpha_1}$, too, though since x_2 just toggles between 0 and 1 and this changes the value of $\mathfrak{g}_1^{\alpha_1}$, the number of times $\mathscr{I}_{21}(\alpha_2)$ should be issued is not decreased. $\Diamond$

[1] All arbitrary points of updating and guard functions are non-reachable.

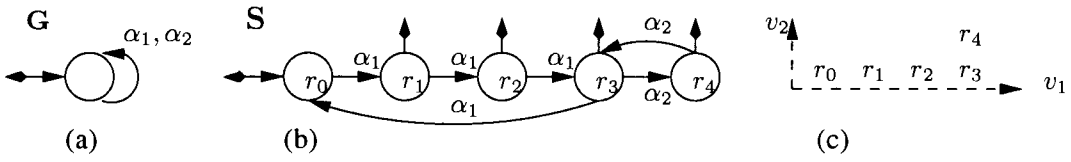


Figure 5.1: (a) Plant **G**, (b) specification and centralized supervisor **S**, and (c) a graphical representation of the ALM for **S**.

Table 5.3: Updating functions for the DSDES in Example 5.2

$\mathbf{v}$	00	10	20	30	else
$\mathcal{A}(\alpha_1, \mathbf{v})$	10	20	30	00	–
$\mathbf{v}$	30	31	else		
$\mathcal{A}(\alpha_2, \mathbf{v})$	31	30	–		

In the following we formalize the above argument which leads to a reduction in the number of occurrence of some of $\mathscr{I}$ events.

Notation 5.1 For $m \in \mathbb{N}$, an m-set is a set of m objects taken from the universe of discourse. The power set of a set Z is denoted by $\mathscr{P}(Z)$ or 2^Z and is the set of all subsets of Z. Let Z and Z' be two finite sets and $f : Z \to Z' : z \mapsto z'$. Denote by $Im(f)$ the image of Z under f, i.e. $Im(f) = \{z' \in Z' \mid \exists z \in Z.\ f(z) = z'\}$. Let $\mathcal{E}(Z)$ denote the set of equivalence relations on Z. For every $z_1, z_2 \in Z$ and $E \in \mathcal{E}$ let $z_1 \equiv_E z_2$ denote the fact that z_1 and z_2 belong to the same equivalence class. Write the equivalence class of $z \in Z$ as $[z]_E$ and the number of cosets of E as $|E|$. For $E_1, E_2 \in \mathcal{E}$ denote by $E_1 \wedge E_2$ the meet of E_1 and E_2, i.e. for every $z_1, z_2 \in Z.\ z_1 \equiv_{E_1 \wedge E_2} z_2$ if and only if $z_1 \equiv_{E_1} z_2$ and $z_1 \equiv_{E_2} z_2$. By $E_1 \leq E_2$ we mean that E_1 is finer than E_2, i.e. $|E_1| \leq |E_2|$. When this inequality is strict, we use $<$ instead of $\leq$. Denote by $\ker(f) \in \mathcal{E}$ the equivalence kernel of f, i.e. for every $z_1, z_2 \in Z.\ z_1 \equiv_{\ker(f)} z_2 \iff f(z_1) = f(z_2)$.

Let $i, j \in I$, $j \neq i$, and $\Lambda \subseteq I$ be a nonempty subset of I. Following

Table 5.4: Computed polynomials for updating and guard functions of the DSDES in Example 5.2

$x_1 := \mathfrak{a}_1^{\alpha_1}(\mathbf{x}) = 4(x_1+1)(x_1+2)(x_1^2+2)$
$x_2 := \mathfrak{a}_2^{\alpha_2}(\mathbf{x}) = 4(x_2+1)(x_2+2)(x_2+3)(x_2+4)$
$\mathfrak{g}_1^{\alpha_1}(\mathbf{x}) = 4(x_2+1)(x_2+2)(x_2+3)(x_2+4)$
$\mathfrak{g}_2^{\alpha_2}(\mathbf{x}) = 4x_1(x_1+1)(x_1+3)(x_1+4)$

Notation 4.3, for a function $f(\mathbf{x})$, defined over $\mathbb{F}_p^n$, and for $v \in \mathbb{F}_p$, by writing $f(\mathbf{x})|_{x_i=v}$ or $f(v, \mathbf{x}_{-i})$ we mean $f(x_1, \cdots, x_{i-1}, v, x_{i+1}, \cdots, x_n)$, i.e. the value of f when $x_i = v$ and x_js ($j \in I \setminus \{i\}$) are left arbitrary. Also for $\mathbf{v} \in \mathbb{F}_p^{|\Lambda|}$, by writing $f(\mathbf{x})|_{\mathbf{x}_\Lambda=\mathbf{v}}$ or $f(\mathbf{v}, \mathbf{x}_{-\Lambda})$ we mean the value of $f(\mathbf{x})$ when for all $j \in I \setminus \Lambda$, x_j is arbitrary and for all $k \in \Lambda$, x_k is assigned a value which is provided by the corresponding component of $\mathbf{v}$. Recalling Definition 4.9, let $X_{i-j} = X_i \setminus \{x_{ij}^h, \cdots, x_{ij}^1\}$ and v have a binary representation $(v^h, \cdots, v^1)$. For a function (or formula) $f(X_i)$, which is defined over $\mathbb{B}^{nh}$, by writing $f((v^h, \cdots, v^1), X_{i-j})$ we mean $f(X_i)$ when $(x_{ij}^h, \cdots, x_{ij}^1) = (v^h, \cdots, v^1)$ and all other variables in X_i can take arbitrary values in $\mathbb{B}$. Let $id_{\mathbb{F}_p} : \mathbb{F}_p \to \mathbb{F}_p : n \mapsto n$ be the identity function on $\mathbb{F}_p$. □

Definition 5.5 Let $f : \mathbb{F}_p^n \to \mathbb{F}_p : \mathbf{x} \mapsto f(\mathbf{x})$ and $i \in I$. Define binary relation $\equiv_{f,i}$ on $\mathbb{F}_p$ as follows.

$\forall v, v' \in \mathbb{F}_p.\quad v \equiv_{f,i} v' \iff [\forall j \in I \setminus \{i\}, \forall x_j \in \mathbb{F}_p.\ \ f(\mathbf{x})|_{x_i=v} = f(\mathbf{x})|_{x_i=v'}]$ □

It can be verified that $\equiv_{f,i}$ is reflexive, symmetric, and transitive, proving the following.

Lemma 5.2 Let f be as in Definition 5.5 and $i \in I$. Then $\equiv_{f,i}$ is an equivalence relation. ■

As a result, for each $i \in I$, under $\equiv_{f,i}$ the set $\mathbb{F}_p$, from which x_i takes its value, is partitioned into equivalence classes, whose number is finite because of the

finiteness of $\mathbb{F}_p$. For each $\mathbf{v} \in \mathbb{F}_p$, denote its equivalence class by $[\mathbf{v}]_{f,i}$ and let the quotient set thus obtained be written as $\mathbb{F}_p / \equiv_{f,i}$.

Example 5.3 Consider function $f(x_1, x_2) = 2x_1[x_1 + 1 + 2(x_1 + 2)(x_2 + 1)^2]$, which is defined on $\mathbb{F}_3^2$ and its image is $Im(f) = \{0, 1\}$. Figure 5.2 shows a graphical representation of f by labeling each point in its domain with its image. It can be seen that $f(0,0) \neq f(1,0), f(0,0) \neq f(2,0)$ and $f(1,1) \neq f(2,1)$. These imply that $[0]_{f,1} \neq [1]_{f,1}, [0]_{f,1} \neq [2]_{f,1}$, and $[1]_{f,1} \neq [2]_{f,1}$. On the other hand $f(2,0) \neq f(2,1)$ and $f(2,2) \neq f(2,1)$, but for every $x_1 \in \mathbb{F}_3$ we have $f(x_1, 0) = f(x_1, 2)$, implying that $[0]_{f,2} = [2]_{f,2}$ and $[0]_{f,2} \neq [1]_{f,2}$. Following Definition 5.5 and this observation we may distinguish the following quotient sets on $\mathbb{F}_3$ induced by $\equiv_{f,1}$ and $\equiv_{f,2}$.

$$\mathbb{F}_3 / \equiv_{f,1} = \{[0]_{f,1}, [1]_{f,1}, [2]_{f,1}\}, \quad [0]_{f,1} = \{0\}, \quad [1]_{f,1} = \{1\}, \quad [2]_{f,1} = \{2\},$$

$$\mathbb{F}_3 / \equiv_{f,2} = \{[0]_{f,2}, [1]_{f,2}\}, \quad [0]_{f,2} = [2]_{f,2} = \{0, 2\}, \quad [1]_{f,2} = \{1\}$$

Equivalent classes are shown in the figure. Observe that when x_2 changes between 0 to 2, the value of f does not vary. Hence, if assume x_1 and x_2 are owned by two supervisors $\mathbf{S}_1$ and $\mathbf{S}_2$ and f is one of $\mathbf{S}_1$'s guard functions, $\mathbf{S}_1$ does not require to get informed of a change between $[0]_{f,2}$ and $[2]_{f,2}$ for x_2, thereby reducing the number of occurrences of communication event $\mathscr{I}_{21}$. $\Diamond$

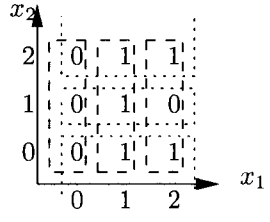


Figure 5.2: Graphical representation of $f(x_1, x_2) = 2x_1[x_1 + 1 + 2(x_1 + 2)(x_2 + 1)^2]$ and equivalent classes.

In general, since $\mathbb{F}_p$ has p elements, $\mid \equiv_{f,i} \mid \leq p$, i.e. there can exist up to p equivalents classes. In particular, for an updating or a guard function computed by Algorithm 4.1 the following holds.

Lemma 5.3 Let the updating and guard functions in PDS (4.14) be computed by Algorithm 4.1. Then the following holds.

$$\forall i, j \in I, i \neq j, \forall \alpha \in \Sigma_{o,i}, \forall \beta \in \Sigma_{c,i}. \ \mid \equiv_{\mathfrak{a}_i^\alpha, j} \mid \leq p \ \wedge \ \mid \equiv_{\mathfrak{g}_i^\beta, j} \mid \leq 2$$

Proof: Immediate by identifying that the images of an updating (respectively, a guard) function is a subset of $\{0.1, \cdots, p-1\}$ (respectively, $\{0, 1\}$). ■

If function f is an updating or a guard function, its EFSM implementation inherits the partitioning induced by $\equiv_{f,i}$, as stated in the following result.

Lemma 5.4 Let $\alpha \in \Sigma_o, \beta \in \Sigma_c$, and $i \in I$, and $\mathfrak{a}_i^\alpha$ and $\mathfrak{g}_i^\beta$ be, respectively, updating and guard functions in DSDES framework, which are implemented in EFSM framework in the sense of Definition 4.10. Let $v, w \in \mathbb{F}_p$ be integers whose binary representations are denoted by $(v^h, \cdots, v^1)$ and $(w^h, \cdots, w^1)$, respectively, and $j \in I \setminus \{i\}$. Then we have the following.

$$v \equiv_{\mathfrak{a}_i^\alpha, j} w \iff \forall k \in J.$$
$$a_i(x_{ii}^k, \beta)((v^h, \cdots, v^1), X_{i-j}) = a_i(x_{ii}^k, \alpha)((w^h, \cdots, w^1), X_{i-j})$$
$$v \equiv_{\mathfrak{g}_i^\beta, j} w \iff g_i(\beta)((v^h, \cdots, v^1), X_{i-j}) = g_i(\beta)((w^h, \cdots, w^1), X_{i-j})$$

Proof: Follows directly from Definition 4.10. ■

The observation we made in Example 5.3 forms the intuition behind communication policy 3. For $i, j \in I, i \neq j$, this policy requires that the updated value of x_j be sent to a supervisor $\mathbf{S}_i$, whose (at least) one updating or guard function f depends on x_j, if the updated value results in a change in the equivalence class, induced by $\equiv_{f,j}$. This policy is formally defined next.

Definition 5.6 Associated with PDS (4.14) and for each $\mathfrak{a}_i^\alpha$ and $j \in I$, such that $\alpha \in \Sigma_{o,i}, i \in I, j \neq i$, and $x_j \in \arg(\mathfrak{a}_i^\alpha)$, consider a partitioning of $\mathbb{F}_p$ induced by $\equiv_{\mathfrak{a}_i^\sigma,j}$. Similarly for each $\mathfrak{g}_i^\beta$ and $k \in I$ such that $\beta \in \Sigma_{c,i}, i \in I, k \neq i$, and $x_k \in \arg(\mathfrak{g}_i^\beta)$, consider a partitioning of $\mathbb{F}_p$ induced by $\equiv_{\mathfrak{g}_i^{\sigma'},k}$. For PDS (4.14) and its EFSM implementation (4.15), *communication policy 3* is as follows: For every $i, j \in I, i \neq j$, we have $H_{ji}=\emptyset$, $\Sigma_{\mathscr{R}_{ji}}=\emptyset$, $\Sigma_{\mathscr{I},j}=\Sigma_{o,j}$, and

$$\forall \sigma \in \Sigma_{o,j}.\ \mathscr{I}_{ji}(\sigma)=\{\hat{x}_{jj}^k \in X_{jj} \mid \hat{x}_{jj}^k := a_j(x_{jj}^k,\sigma)(X_j)\ \wedge\ \hat{x}_{jj}^k \neq x_{jj}^k$$
$$\wedge\ \big[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\ \big(x_{ij}^k \in \arg(a_i(x',\sigma'))\ \wedge\ [\hat{x}_j]_{\mathfrak{a}_i^{\sigma'},j} \neq [x_j]_{\mathfrak{a}_i^{\sigma'},j}\big)$$
$$\vee \big(x_{ij}^k \in \arg(g_i(\sigma'))\ \wedge\ [\hat{x}_j]_{\mathfrak{g}_i^{\sigma'},j} \neq [x_j]_{\mathfrak{g}_i^{\sigma'},j}\big)\big]\}. \qquad \square$$

Proposition 5.2 *A solution to Problem 5.1:* Associated with Problem 5.1, if the network of supervisors is strongly connected with lossless channels and if communication is instantaneous, communication policy 3 insures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

By Definition 5.6, for $i, j \in I, i \neq j$, communication policy 3 prescribes the transfer of the *new* values of a bit $x \in X_{jj}$ through $\mathscr{I}_{ji}$, whose associated *new* integer value of x_j changes the equivalence class induced by (at least) one of the updating or guard functions of $\mathbf{S}_i$. In other words, if the new integer value entails no change in the equivalent classes in the integer space, the new bit values are not sent. As a result, comparing to policy 2 of Definition 5.4, fewer $\mathscr{I}$ events would be issued if the partition induced by all updating and guard functions (of $\mathbf{S}_i$) is coarser than the integer space. This is summarized in the following result.

Definition 5.7 Let $i, j \in I$ and $i \neq j$. Associated with PDS (4.14) define $\mathfrak{a}_i = \{\mathfrak{a}_i^\sigma \mid \sigma \in \Sigma_{o,i}\}$ and $\mathfrak{g}_i = \{\mathfrak{g}_i^\sigma \mid \sigma \in \Sigma_{c,i}\}$ as the sets of updating and guard functions, respectively. Define $\equiv_{i,j}$ as $\bigwedge_{f \in \mathfrak{a}_i \cup \mathfrak{g}_i} \equiv_{f,j}$, i.e. the meet of all equivalence relations on the integer space of x_j, each induced by an updating or a guard function of $\mathbf{S}_i$. $\square$

Lemma 5.5 Consider PDS (4.14) and its EFSM implementation (4.15). Let $i, j \in I$, $i \neq j$, $\sigma \in \Sigma_o$, and ${}^2\mathscr{I}_{ji}(\sigma)$ and ${}^3\mathscr{I}_{ji}(\sigma)$ denote $\mathscr{I}_{ji}(\sigma)$ event under communication policies 2 and 3, respectively.

1. If $|{}^3\mathscr{I}_{ji}(\sigma)| < |{}^2\mathscr{I}_{ji}(\sigma)|$, then $\ker(id_{\mathbb{F}_p}) < \equiv_{i,j}$ and $|\equiv_{i,j}| < p$.

2. Let $\sigma \in \Sigma_{o,j}$ and for some $\sigma' \in \Sigma_i$ either $x_j \in \arg(\mathfrak{a}_i^{\sigma'})$ or $x_j \in \arg(\mathfrak{g}_i^{\sigma'})$. If σ moves the PDS from $\mathbf{x}$ to $\hat{\mathbf{x}}$ and $x_j \equiv_{i,j} \hat{x}_j$, then $|{}^3\mathscr{I}_{ji}(\sigma)| < |{}^2\mathscr{I}_{ji}(\sigma)|$.

■

Notice that the assumptions in part 2 of Lemma 5.5 requires that the PDS move from state $\mathbf{x}$ to $\hat{\mathbf{x}}$, which entails $x_j \neq \hat{x}_j$, and that $x_j \equiv_{i,j} \hat{x}_j$, implying that $\ker(id_{\mathbb{F}_p}) < \equiv_{i,j}$. Therefore, in a sense part 2 is the "converse" of part 1. Accordingly, the condition $\ker(id_{\mathbb{F}_p}) < \equiv_{i,j}$ is related to the reduction of communication content in the sense of Lemma 5.5.

Remark 5.2 Implied by the proof of Lemma 5.5 and Definition 5.7 is that communication policy 3 can be written in the following way, too. Associated with PDS (4.14) and its EFSM implementation (4.15), for every $i, j \in I, i \neq j$, we have $H_{ji}=\emptyset$, $\Sigma_{\mathscr{R}_{ji}}=\emptyset$, $\Sigma_{\mathscr{I},j}=\Sigma_{o,j}$, and

$$\forall \sigma \in \Sigma_{o,j}.\ \mathscr{I}_{ji}(\sigma) = \{\hat{x}_{jj}^k \in X_{jj} \mid \hat{x}_{jj}^k := a_j(x_{jj}^k, \sigma)(X_j) \ \wedge \ \hat{x}_{jj}^k \neq x_{jj}^k \wedge$$
$$\big[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\ \big(x_{ij}^k \in \arg(a_i(x', \sigma')) \vee x_{ij}^k \in \arg(g_i(\sigma'))\big)$$
$$\wedge [\hat{x}_j]_{i,j} \neq [x_j]_{i,j}\big]\}.$$ □

Example 5.4 Let us redo Example 5.2 using the above machinery. First, we have the following.

$(\mathbb{F}_5/\equiv_{\mathfrak{a}_1^{\alpha_1},2}) = \{[0]_{\mathfrak{a}_1^{\alpha_1},2}\}$, where $[0]_{\mathfrak{a}_1^{\alpha_1},2} = \{0,1,2,3,4\}$;

$(\mathbb{F}_5/\equiv_{\mathfrak{g}_1^{\alpha_1},2}) = \{[0]_{\mathfrak{g}_1^{\alpha_1},2}, [1]_{\mathfrak{g}_1^{\alpha_1},2}\}$, where $[0]_{\mathfrak{g}_1^{\alpha_1},2} = \{1\}$, $[1]_{\mathfrak{g}_1^{\alpha_1},2} = \{1,2,3,4\}$;

$(\mathbb{F}_5/\equiv_{\mathfrak{a}_2^{\alpha_2},1}) = \{[0]_{\mathfrak{a}_2^{\alpha_2},1}\}$, where $[0]_{\mathfrak{a}_2^{\alpha_2},1} = \{0,1,2,3,4\}$;

$(\mathbb{F}_5/\equiv_{\mathfrak{g}_2^{\alpha_2},1}) = \{[0]_{\mathfrak{g}_2^{\alpha_2},1}, [3]_{\mathfrak{g}_2^{\alpha_2},1}\}$, where $[0]_{\mathfrak{g}_2^{\alpha_2},1} = \{0,1,2,4\}$, $[3]_{\mathfrak{g}_2^{\alpha_2},1} = \{3\}$.

This would lead to $(\equiv_{1,2}) = (\equiv_{\mathfrak{a}_1^{\alpha_1},2} \wedge \equiv_{\mathfrak{g}_1^{\alpha_1},2})$, $(\equiv_{2,1}) = (\equiv_{\mathfrak{a}_2^{\alpha_2},1} \wedge \equiv_{\mathfrak{g}_2^{\alpha_2},1})$, and the following.

1) $(\mathbb{F}_5/\equiv_{1,2}) = \{[0]_{1,2}, [1]_{1,2}\}$, where $[0]_{1,2} = \{1\}$ and $[1]_{1,2} = \{1,2,3,4\}$;

2) $(\mathbb{F}_5/\equiv_{2,1}) = \{[0]_{2,1}, [3]_{2,1}\}$, where $[0]_{2,1} = \{0,1,2,4\}$ and $[3]_{2,1} = \{3\}$.

Using Definition 5.6, 1) insures that while x_2 is altering within the set $[1]_{1,2} = \{1,2,3,4\}$, $\mathbf{S}_2$ need not inform $\mathbf{S}_1$ of its exact value. However, a closer observation of $\mathfrak{a}_2^{\alpha_2}$ reveals that $Im(\mathfrak{a}_2^{\alpha_2}) = \{0,1\}$, which in turn implies that x_2 is actually altering between 0 and 1 all the time, i.e. it is always moving from one equivalence class to another, therefore in this case its latest value should always be communicated upon changing. Thus, as far as $\mathscr{I}_{21}(\alpha_2)$ is concerned, policy 3 performs in the same manner as policy 2 and cannot decrease its number of occurrence any further. When it comes to $\equiv_{2,1}$, 2) insures that while x_1 is moving within $[0]_{2,1} = \{0,1,2,4\}$, its exact value is not required by $\mathbf{S}_2$. On the other hand, $Im(\mathfrak{a}_1^{\alpha_1}) = \{0,1,2,3\}$ which implies that x_1 indeed takes values within the set $\{0,1,2\}$, for which, policy 3 decreases the number of $\mathscr{I}_{12}(\alpha_1)$ events.

To see how the above argument is implemented in practice, let us compute actions and guards in EFSM framework. To this end, notice that Algorithm 4.2 initiates from $h = \lceil \log p \rceil = \lceil \log 5 \rceil = 3$ binary variables for each supervisor, i.e. x_i is represented by $(x_{ii}^3, x_{ii}^2, x_{ii}^1)$, where $i \in \{1,2\}$. To simplify the expressions for $a_1(x_{11}^k, \alpha_1)$, $k \in J = \{1,2,3\}$, we have assumed that points $(x_{11}^3 x_{11}^2 x_{11}^1) \in \{(101), (110)\}$ and (111), which do not exist in $\mathbb{F}_5$ but can be encoded using three Boolean variables, are arbitrarily mapped to (010) and (000), respectively. Also, to simplify the expressions for $a_1(x_{22}^k, \alpha_2)$, $k \in J$, all three points $(x_{22}^3 x_{22}^2 x_{22}^1) \in \{(101), (110), (111)\}$, are mapped to (000). When it comes to compute guards $g_1(\alpha_1)$ and $g_2(\alpha_2)$, we assume that α_1 and α_2 are all disabled at these three points. Following these simplifications, Table 5.5

Table 5.5: Actions and guards in EFSM framework for supervisors in Example 5.2

$a_1(x_{11}^3, \alpha_1)(\mathbf{x}) = 0$
$a_1(x_{11}^2, \alpha_1)(\mathbf{x}) = \overline{x}_{11}^2 x_{11}^1$
$a_1(x_{11}^1, \alpha_1)(\mathbf{x}) = \overline{x}_{11}^3 \overline{x}_{11}^1 = \overline{x}_{11}^1$
$a_2(x_{22}^3, \alpha_2)(\mathbf{x}) = 0$
$a_2(x_{22}^2, \alpha_2)(\mathbf{x}) = 0$
$a_2(x_{22}^1, \alpha_2)(\mathbf{x}) = \overline{x}_{22}^3 \overline{x}_{22}^2 \overline{x}_{22}^1 = \overline{x}_{22}^1$
$g_1(\alpha_1) = \overline{x}_{12}^3 \overline{x}_{12}^2 \overline{x}_{12}^1 = \overline{x}_{12}^1$
$g_2(\alpha_2) = x_{21}^2 x_{21}^1$

lists the actions and guards. Observe that out of the six introduced Boolean variables, x_{11}^3, x_{22}^3, and x_{22}^2 are constantly equal to zero and need not be communicated at all.

Regarding the other three variables, the copies of x_{11}^2, x_{11}^1, i.e. x_{21}^2, x_{21}^1 belong to $g_2(\alpha_2)$, and the copy of x_{22}^1, i.e. x_{12}^1 belongs to $g_1(\alpha_1)$. Communication policy 2 then requires that when each of these three variables be sent by its owning supervisor to the other supervisor upon getting changed. Denoting by $\oplus$ the *exclusive OR* operation of two binary expressions[2], x_{11}^2, x_{11}^1, and x_{22}^1 will change if $x_{11}^2 \oplus a_1(x_{11}^2, \alpha_1)(\mathbf{x}) = 1$, $x_{11}^1 \oplus a_1(x_{11}^1, \alpha_1)(\mathbf{x}) = 1$, and $x_{22}^1 \oplus a_1(x_{22}^1, \alpha_2)(\mathbf{x}) = 1$, respectively. Whereas the first condition reduces to $x_{11}^1 = 1$, the last two are always 1, i.e. true. As a result, x_{11}^1 and x_{22}^1 are always communicated (because they are always changed upon each occurrence of α_1 and α_2, respectively), and x_{11}^2 is communicated only if $x_{11}^1 = 1$. These results are summarized in Table 5.6.

On top of conditions imposed by policy 2, policy 3 requires that the value of a changed binary variables not be communicated unless its new value

[2]For every two Boolean variables c and d, $c \oplus d$ is 1 if and only if $c \neq d$.

Table 5.6: Communication amongst supervisors in Example 5.2 based on policy 2

$\mathscr{I}_{12}(\alpha_1) = \begin{cases} \{x_{11}^1, x_{11}^2\} & ; \text{ if } (x_{11}^1 = 1) \\ \{x_{11}^1\} & ; \text{ if } (x_{11}^1 = 0) \end{cases}$
$\mathscr{I}_{21}(\alpha_2) = \{x_{22}^1\};$

alters the value of the function which depends on the corresponding integer variable of the binary variable. Considering x_1, we have $Im(\mathfrak{a}_1^{\alpha_1}) = \{0,1,2,3\}$ partitioned into $[0]_{\mathfrak{a}_1^{\alpha_1}} = \{0,1,2\}$ and $[1]_{\mathfrak{a}_1^{\alpha_1}} = \{3\}$. Therefore, communication of a changed x_{11}^2 or x_{11}^1 to $\mathbf{S}_2$ is required only when the new binary value changes $[0]_{\mathfrak{a}_1^{\alpha_1}}$ to $[1]_{\mathfrak{a}_1^{\alpha_1}}$ and vice versa. Encoding $\hat{x_1}$ as $(\hat{x}_{11}^3\hat{x}_{11}^2\hat{x}_{11}^1)$, this condition is translated into $\hat{x}_{11}^2\hat{x}_{11}^1 \oplus x_{11}^2 x_{11}^1 = 1$, which is then equal to $x_{11}^2 = 1$ upon substituting $\hat{x}_{11}^2$ and $\hat{x}_{11}^1$ with their equivalent expressions of the actions from Table 5.5. As for x_2, we have $Im(\mathfrak{a}_2^{\alpha_2}) = \{0,1\}$ partitioned into $[0]_{\mathfrak{a}_2^{\alpha_2}} = \{0\}$ and $[1]_{\mathfrak{a}_2^{\alpha_2}} = \{1\}$. Therefore, communication of a changed x_{11}^1 to $\mathbf{S}_1$ is required only when the new binary value changes $[0]_{\mathfrak{a}_2^{\alpha_2}}$ to $[1]_{\mathfrak{a}_2^{\alpha_2}}$ and vice versa. Encoding $\hat{x_2}$ as $(\hat{x}_{22}^3\hat{x}_{22}^2\hat{x}_{22}^1)$, this condition is translated into $\hat{x}_{22}^1 \oplus x_{22}^1 = 1$, which is always true upon substituting $\hat{x}_{22}^1$ with its $a_2(x_{22}^1, \alpha_2)$ from Table 5.5. From the shown results in Table 5.7 and comparing them with those in Table 5.6. Observe that policy 3 indeed decreases the number of times $\mathscr{I}_{12}$ occurs by imposing the new condition $x_{11}^2 = 1$ such that now there are cases where this communication event is void, i.e. does not occur. $\Diamond$

The formalization developed so far is based on the equivalence relation $\equiv_{f,j}$, where f is an updating or a guard function owned by $\mathbf{S}_i$ and $i, j \in I, j \neq i$. Recalling Definition 5.5, this relation requires that the value of function f remain unchanged for two different values of x_j regardless of values of other x_ks, where $k \in I \setminus \{j\}$. In other words, there are $n-1$-dimensional

Table 5.7: Communication amongst supervisors in Example 5.2 based on policy 3

$\mathscr{I}_{12}(\alpha_1) = \begin{cases} \{x^1_{11}, x^2_{11}\} & ; \text{ if } (x^1_{11} = 1 \wedge x^2_{11} = 1) \\ \{x^1_{11}\} & ; \text{ if } (x^1_{11} = 0 \wedge x^2_{11} = 1) \\ \emptyset & ; \text{ otherwise} \end{cases}$
$\mathscr{I}_{21}(\alpha_2) = \{x^1_{22}\};$

subspaces which share the same pattern of values of f. However, this might be a strong requirement for a PDS to satisfy and instead, there might be *subsets* of $n-1$-dimensional subspaces which share the same pattern of values of f. To formalize such cases, the concept of *invariance* which was introduced before, can be extended to invariance with respect to such subsets, as defined next.

Definition 5.8 Let $f : \mathbb{F}_p^n \to \mathbb{F}_p : \mathbf{x} \mapsto f(\mathbf{x})$, $j \in I$, for every $k \in I \setminus \{j\}$ it holds that $\emptyset \neq \Lambda_k \subseteq \mathbb{F}_p$, and $\Lambda = \prod_{k \in I \setminus \{j\}} \Lambda_k$. Define the binary relation $\equiv_{f,j,\Lambda}$ on $\mathbb{F}_p$ as follows.

$\forall v, v' \in \mathbb{F}_p.\ v \equiv_{f,j,\Lambda} v' \iff [\forall k \in I \setminus \{j\}, \forall \mathbf{x}_{-j} \in \Lambda.\ \ f(v, \mathbf{x}_{-j}) = f(v', \mathbf{x}_{-j})]$ □

Relation $\equiv_{f,j,\Lambda}$ is also reflexive, symmetric, and transitive implying the following result.

Lemma 5.6 The binary relation $\equiv_{f,j,\Lambda}$ is an equivalence relation. ■

Lemma 5.7 For every $j \in I$, function f, and Λ as in Definition 5.8, we have

$$\equiv_{f,j} \leq \equiv_{f,j,\Lambda}.$$

Proof: For every $v, v' \in \mathbb{F}_p$, $v \equiv_{f,j} v'$ holds for the whole $n-1$-dimensional subspace, including Λ as its subset. ■

Example 5.5 Figure 5.3 shows the graphical representation of a function f, defined on $\mathbb{F}_3^3$. Observe that $0 \equiv_{f,2,\Lambda} 1$, where $\Lambda = \Lambda_1 \times \Lambda_3$, $\Lambda_1 = \mathbb{F}_3$, and

$\Lambda_3 = \{0, 1\}$. As a result of partitioning the set of point of v_2 axis, there are two equivalence classes $[0]_{f,2,\Lambda} = \{0, 1\}$ and $[2]_{f,2,\Lambda} = \{2\}$. $\Diamond$

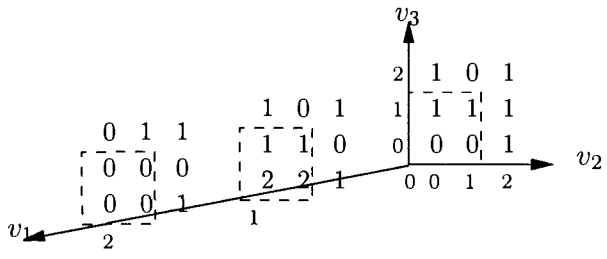


Figure 5.3: Graphical representation of a function f with symmetric pattern with respect to v_2 axis.

Assume that f is an updating or a guard functions owned by $\mathbf{S}_i$. Definition 5.8 then suggests that while $\mathbf{x}_{-j} \in \Lambda$, the new value of x_j need not be sent to $\mathbf{S}_i$ as long as this value belongs to the same equivalence class as the old value. To insure that $\mathbf{x}_{-j} \in \Lambda$, $\mathbf{S}_j$ requires to be informed by all $\mathbf{S}_k$s, $k \in I \backslash \{j\}$, for which $\Lambda_k \neq \mathbb{F}_p$. In other words to reduce the amount of communication associated with $\mathscr{I}_{ji}(\sigma)$, a communication event $\mathscr{I}_{kj}(\sigma')$ should have been sent by $\mathbf{S}_k$ to $\mathbf{S}_j$ each time x_k enters or leaves Λ_k, where $k \in I \subseteq \{j\}$, $\sigma \in \Sigma_{o,j}$, and $\sigma' \in \Sigma_{o,k}$. Whether using these $\mathscr{I}_{kj}$s to avoid some $\mathscr{I}_{ji}$s can reduce the total amount of communication in the network, depends on many factors and cannot be guaranteed in general. As an example, whereas the fact that $\Lambda_k = \mathbb{F}_p$ eliminates $\mathscr{I}_{kj}$, a Λ with larger cardinality may provide larger equivalence classes, thereby omitting more $\mathscr{I}_{ji}$s. Therefore it might not be even possible to tell which Λs reduce the communication content more *a priori*, i.e. before analyzing different possibilities. The only cases where one can trivially say that these $\mathscr{I}_{kj}$s are redundant and should be avoided are when $| \equiv_{f,j,\Lambda} | = p$, i.e. the number of partitions equal to $|\mathbb{F}_p|$, itself. In such cases

no symmetries exist and therefore, no $\mathscr{I}_{jk}$ events will help reduce (the content of) $\mathscr{I}_{ji}$s. We do not investigate this issue furthermore here and in what follows formulate the solution to communication problem subject to a given set Λ.

Similar to what we did for $\equiv_{f,j}$ in Definition 5.7, when dealing with all updating and guard functions of $\mathbf{S}_i$, the meet of the equivalence relations $\equiv_{f,j,\Lambda}$, each induced by one such function, should be utilized. To this end, notice that the subset of $\mathbb{F}_p^{n-1}$ for which $\equiv_{f,j,\Lambda}$ holds is specific to f. Therefore, for the meet to hold, its subset of $\mathbb{F}_p^{n-1}$ is defined as the intersection of all Λs, each for an equivalence relation $\equiv_{f,j,\Lambda}$.

Definition 5.9 Let $i,j,k \in I, j \neq i, k \neq j$, and recall Definition 5.8. Associated with PDS (4.14) define $\mathfrak{a}_i = \{\mathfrak{a}_i^\sigma \mid \sigma \in \Sigma_{o,i}\}$ and $\mathfrak{g}_i = \{\mathfrak{g}_i^\sigma \mid \sigma \in \Sigma_{c,i}\}$ as the sets of updating and guard functions, respectively. For each $f \in \mathfrak{a}_i \cup \mathfrak{g}_i$, j such that $x_j \in \arg(f)$, and every k, assume that the nonempty set $\Lambda_{f,-j,k} \subseteq \mathbb{F}_p$ exists such that $\Lambda_{f,-j} = \prod_{k \in I\setminus\{j\}} \Lambda_{f,-j,k}$ and $\equiv_{f,j,\Lambda_{f,-j}}$ can be defined on $\mathbb{F}_p$. Define $\Lambda_{i,-j,k} = \bigcap_{f \in \mathfrak{a}_i \cup \mathfrak{g}_i} \Lambda_{f,-j,k}$, $\Lambda_{i,-j} = \prod_{k \in I\setminus\{j\}} \Lambda_{i,-j,k}$, and let $\equiv_{f,j,\Lambda_{i,-j}}$ be the restriction of $\equiv_{f,j,\Lambda_{f,-j}}$ to $\mathbf{x}_{-j} \in \Lambda_{i,-j} \subseteq \Lambda_{f,-j}$. Define $\equiv_{i,j,\Lambda_{i,-j}}$ as $\bigwedge_{f \in \mathfrak{a}_i \cup \mathfrak{g}_i} \equiv_{f,j,\Lambda_{i,-j}}$, i.e. the meet of all equivalence relations on $\mathbb{F}_p$, each induced by an updating or a guard function of $\mathbf{S}_i$. □

Lemma 5.8 Relations $\equiv_{f,j,\Lambda_{i,-j}}$ and $\equiv_{i,j,\Lambda_{i,-j}}$ in Definition 5.9 are well defined.

■

In the following we formalize a communication policy which is inspired by equivalence relations $\equiv_{i,j,\Lambda_{i,-j}}$ and prove its correctness.

Definition 5.10 Associated with PDS (4.14) and following Definition 5.9, for every $i,j,k \in I$, such that $j \neq i$ and $k \neq j$, let $\equiv_{i,j,\Lambda_{i,-j}}$ be defined on $\mathbb{F}_p$, where $\Lambda_{i,-j,k} \subseteq \mathbb{F}_p$ and $\Lambda_{i,-j} = \prod_{k \in I\setminus\{j\}} \Lambda_{i,-j,k}$. For PDS (4.14) and its EFSM implementation (4.15), *communication policy 4* is as follows:

$\forall i,j \in I, i \neq j, H_{ji} = \emptyset$, $\Sigma_{\mathscr{R}_{ji}} = \emptyset$, $\Sigma_{\mathscr{I},j} = \Sigma_{o,j} \cup \{\{\mathscr{I}_{mj}(\sigma) |\ m \in I, m \neq j, \sigma \in \Sigma_{o,m}\}\}$,

and

1) $\forall \sigma \in \Sigma_{o,j}.\ \mathscr{I}_{ji}(\sigma) = \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j)\ \wedge\ \hat{x}^k_{jj} \neq x^k_{jj}\ \wedge$
$\big[\exists\ l \in I \setminus \{i\}.\ (|\equiv_{l,i,\Lambda_{l,-i}}| < p) \wedge (x_j \in \Lambda_{l,-i,j} \Longrightarrow \hat{x}_j \notin \Lambda_{l,-i,j}) \wedge (x_j \notin \Lambda_{l,-i,j}$
$\Longrightarrow \hat{x}_j \in \Lambda_{l,-i,j})\big]\}$,

2) $\mathscr{I}_{ji}(\{\mathscr{I}_{mj}(\sigma) |\ m \in I, m \neq j, \sigma \in \Sigma_{o,m}\}) = \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j)$
$\wedge\ \hat{x}^k_{jj} \neq x^k_{jj}\ \wedge [\forall \sigma \in \Sigma_{o,j}.\ \hat{x}^k_{jj} \notin \mathscr{I}_{ji}(\sigma)]$
$\wedge \big[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\ \big(x^k_{ij} \in \arg(a_i(x', \sigma'))\ \vee\ x^k_{ij} \in \arg(g_i(\sigma'))\big) \wedge$
$\big((|\equiv_{i,j,\Lambda_{i,-j}}| < p)\ \Longrightarrow\ \neg(\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j)\big)\big]\}$. □

In simple words, communication policy 4 suggests that communication among supervisors be performed in 2 steps. In the first step each $\mathbf{S}_j$ provides other $\mathbf{S}_i$s with the information to help them recognize their associated $\Lambda_{l,-i}$s, where $l \in I \setminus \{i\}$ (it can be equal to j, too). This is done if this information is necessary, which is when either $x_j \in \Lambda_{l,-i,j}$ or $\hat{x}_j \in \Lambda_{l,-i,j}$, exclusively. Using the information which is received through all communication events issued in the first step, in the second step each $\mathbf{S}_j$ sends its private information to a typical $\mathbf{S}_i$ if this information changes the value of (at least) one of $\mathbf{S}_i$'s guard or updating functions, as reflected in $\neg(\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j)$.

Condition $(|\equiv_{l,i,\Lambda_{l,-i}}| < p)$, in the first step, insures that no communication is done when $(|\equiv_{l,i,\Lambda_{l,-i}}| = p)$, i.e. no symmetry exists. For such cases, actually every change in a supervisor's private variable should be sent to supervisors whose guard or updating functions depend on this variable. This explains why in step 2 the same condition appears as a premise of checking different classes. In other words, the existence of this condition excludes those relations which induce exactly p partitions. Also condition $[\forall \sigma \in \Sigma_{o,j}.\ \hat{x}^k_{jj} \notin \mathscr{I}_{ji}(\sigma)]$ insures that if an $\mathbf{S}_j$'s bit has already been sent in the first step (to help $\mathbf{S}_i$ distinguish partitions), it is not sent in the second step any more, thus avoiding

redundant communication. This is justified if one notices that between steps 1 and 2 no observable event in Σ_o is assumed to occur, and the contents of private variables remain unchanged.

Proposition 5.3 *A solution to Problem 5.1:* Associated with Problem 5.1, if the network of supervisors is strongly connected with lossless channels and if communication is instantaneous, communication policy 4 ensures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

5.2.5 Communication policies which prescribe communication of encoded events

Previous communication policies all prescribe communication of a subset of Boolean variables which encode the states of the centralized supervisor in a decentralized manner as represented by PDS (4.14) and its EFSM implementation (4.15). The PDS representation also suggests communication of (a subset of Boolean variables which encode) *observed events* by observing supervisors. In the following we formalize this idea, thereby demonstrating the capability of SDES framework in modeling communication of observed events, too.

Definition 5.11 For every $i \in I$, define

$$\mathcal{N}_{\mathfrak{a}_i} = \{j \in I \mid \exists \sigma \in \Sigma_{o,i}.\ x_j \in \arg(\mathfrak{a}_i^\sigma)\} \text{ and}$$

$$\mathcal{N}_i = \{j \in I \mid [\exists \sigma \in \Sigma_{o,i}.\ x_j \in \arg(\mathfrak{a}_i^\sigma)] \vee [\exists \sigma' \in \Sigma_{c,i}.\ x_j \in \arg(\mathfrak{g}_i^{\sigma'})]\}.$$

Define also set $\mathcal{O}_i \subseteq I$ recursively as follows.

$$\mathcal{N}_i \subseteq \mathcal{O}_i \quad \wedge \quad [\forall j \in I.\ j \in \mathcal{O}_i \implies \forall k \in \mathcal{N}_{\mathfrak{a}_j}.\ k \in \mathcal{O}_i] \qquad \square$$

Lemma 5.9 We have $\forall i \in I, j \in \mathcal{O}_i, \sigma \in \Sigma_{o,j}.\ x_k \in \arg(\mathfrak{a}_j^\sigma) \implies k \in \mathcal{O}_i$.

Proof: Follows directly from Definition 5.11. ■

In simple words, what Definition 5.11 means is that $\mathcal{O}_i$ is the largest set of $\mathbf{S}_j$s, on whose private information, i.e. x_js, $\mathbf{S}_i$ depends, where $i, j \in I$. This dependency is either explicit, i.e. when x_j appears as the argument of an updating or a guard function of $\mathbf{S}_i$, or implicit, i.e. when the updating function corresponding to such an $x_j, j \neq i$, depends on $x_k, k \neq j, i$.

Lemma 5.10 For every $i \in I$, $\mathcal{O}_i$ is a fixed point of $F : \mathscr{P}(I) \rightarrow \mathscr{P}(I) : A \mapsto \big(A \cup \bigcup_{j \in A} \mathcal{N}_{\mathfrak{a}_j}\big)$.

Clearly F is monotonic and the finiteness of I implies that for every $\mathcal{N}_i \subseteq I$, $F(\mathcal{N}_i)$ converges to fixed point $\mathcal{O}_i$. This can be regarded as the basis of an algorithm to compute $\mathcal{O}_i$.

Algorithm 5.1 *Computation of* $\mathcal{O}_i$: Associated with PDS (4.14), for every $i \in I$ do the following.

1. Compute $\mathcal{N}_i = \{j \in I \mid [\exists \sigma \in \Sigma_{o,i}.\ x_j \in \arg(\mathfrak{a}_i^\sigma)] \vee [\exists \sigma' \in \Sigma_{c,i}.\ x_j \in \arg(\mathfrak{g}_i^{\sigma'})]\}$.
2. Set $A = \mathcal{N}_i$ and $B = F(A)$.
3. While $A \neq B$ do the following: $[A = B$ and $B = F(A)]$.
4. Report $\mathcal{O}_i = A$. □

Following the concept of $\mathcal{O}_i$, if supervisor $\mathbf{S}_i$ stores a copy of the private variables it requires (directly or indirectly), to compute its guard and updating functions together with the copies of updating functions, which are used to reevaluate those copy variables, it can independently compute the new values of the copy variables if it is informed of what observable events occur. The next definition states what we mean by copy variables and updating equations.

Definition 5.12 Associated with PDS (4.14) and for each $i \in I$ and every $j \in \mathcal{O}_i \setminus \{i\}$, let x_j^i be the copy of x_j which is stored by $\mathbf{S}_i$. Let $\mathbf{x}^i$ be the

vector of all x_j^is such that they are sorted from left to right based on their index j (i.e. in the same order they appear in $\mathbf{x}$). Write $\mathbf{x}^i = \mathbf{x}$ to state $\forall i \in I, \forall j \in \mathcal{O}_i \setminus \{i\}.\ x_j^i = x_j$. Also assume that all x_j^is are initialized to 0. Correspondingly, denote by $\mathfrak{a}_{ij}^\sigma(\mathbf{x}^i)$ the copy of updating function $\mathfrak{a}_j^\sigma(\mathbf{x})$ stored by $\mathbf{S}_i$. □

Lemma 5.11 For every $i \in I$, $j \in \mathcal{O}_i \setminus \{i\}$, and $\sigma \in \Sigma_{o,j}$, $\mathfrak{a}_{ij}^\sigma(\mathbf{x}^i)$ in Definition 5.12 is well defined.

Proof: It is enough to show that $\mathbf{x}^i$ includes $\arg(\mathfrak{a}_j^\sigma)$ in it. To this end, let $x_k \in \arg \mathfrak{a}_j^\sigma(\mathbf{x}^i)$. By Lemma 5.9, $k \in \mathcal{O}_i$ which, by Definition 5.12, means that x_k is a component of $\mathbf{x}^i$. ■

To be able to compute copies of other supervisors' private variables, each $\mathbf{S}_i$ should store a copy of their updating functions. We take this point for granted in the following.

Assumption 5.1 Assume that for each $i \in I$, each $j \in \mathcal{O}_i \setminus \{i\}$, and each $\sigma \in \Sigma_{o,j}$, $\mathbf{S}_i$ stores $\mathfrak{a}_{ij}^\sigma(\mathbf{x}^i)$, i.e. it stores a copy of each of $\mathbf{S}_j$'s *non-identity updating functions.* □

Under Assumption 5.1, each supervisor $\mathbf{S}_i$ can independently compute the new values of the private variables of other $\mathbf{S}_j$s upon being informed of the occurrence of every observable event by those $\mathbf{S}_j$s. This paves the way to define a new communication policy which communicates the observable events, rather than state-based information. Whereas in DSDES framework, every pieces of information is represented by binary variables before being communicated, there is no symbolic representation of "events". In the following, we first provide such an encoding scheme which is then used to define the fifth communication policy.

Definition 5.13 Associated with PDS (4.14), let $e = \lceil log_2(\max_{i\in I} |\Sigma_{o,i}|+1)\rceil$, $\mho = \{1, 2, \cdots, e\}$, and $i, j \in I,\ j \neq i$. Denote by $Y_{ii} = \{y_{ii}^k \mid k \in \mho\}$ the set of $\mathbf{S}_i$'s *private event-encoding* Boolean variables and let every event $\sigma \in \Sigma_{o,i} \cup \{\epsilon\}$ be encoded[3] arbitrarily as $\sigma = (y_{ii}^e \cdots y_{ii}^1)$. The extra artificial event, "ϵ," is used to distinguish the case where no event has occurred initially. Denote by y_{ij}^k a copy of $y_{jj}^k \in Y_{jj}$ stored by $\mathbf{S}_i$ and let $Y_{ij} = \{y_{ij}^k \mid y_{jj}^k \in Y_{jj}\}$, $Y_{ci} = \bigcup_{j\in I\setminus\{i\}} Y_{ij}$, $Y_i = Y_{ii} \dot\cup Y_{ci}$, and $Y = \bigcup_{i\in I} Y_i$. Assume that all variables in Y are initially equal to 0. □

As Definition 5.13 reads, every supervisor assigns a code to each of its observable events. To decode these codes upon their arrival, the target supervisor should have a look up table which stores the codes. This is stated in the following assumption, where one should notice that for each $\sigma \in \Sigma_o$, every supervisor whose index is in $I_o(\sigma)$ assigns its own code to σ.

Assumption 5.2 For each $i \in I$, each $j \in \mathcal{O}_i$, and every $\sigma \in \Sigma_{o,j}$, $\mathbf{S}_i$ has a look up table which stores the code(s) for σ (as assigned by different supervisors whose index is in $I_o(\sigma)$). □

Once a supervisor observes the occurrence of an event, on top of updating its own private variable, it updates copies of all other private variables which it keeps and are affected by that event, as summarized in the following assumption.

Assumption 5.3 Let Assumption 5.1 holds. Then for every $i \in I$, each $j \in \mathcal{O}_i$, and each $\sigma \in \Sigma_{o,i} \cap \Sigma_{o,j}$, $\mathbf{S}_i$ computes $\hat{x}_j^i := \mathfrak{a}_{ij}^\sigma(\mathbf{x}^i)$ upon the occurrence of σ. □

[3]Although some $\Sigma_{o,i}$s may have less then $e-1$ elements, we choose a common e for the sake of notational simplicity. Clearly, if $|\Sigma_{o,i}| < e-1$, some higher significant bits in Y_{ii} would be constantly equal to 0.

We have now enough means to introduce the fifth communication policy. The following definition will be helpful in defining the new policy.

Definition 5.14 For each $\sigma \in \Sigma_o$ define $I_{com}(\sigma) = k$, where $k \in I_o(\sigma)$. □

Notice that for each $\sigma \in \Sigma_o$, $I_o(\sigma) \neq \emptyset$, and thus $I_{com}(\sigma)$ is well defined.

Definition 5.15 Considering PDS (4.14) and following Definitions 5.12 and 5.13, for each $i \in I$, let $\alpha_i \in \Sigma_{o,i} \cup \{\epsilon\}$, encoded as $\alpha_i = (y^e_{ii}, \cdots, y^1_{ii})$, be the last event observed by $\mathbf{S}_i$. Assume that for each $\sigma \in \Sigma_o$, $I_{com}(\sigma)$ is given as in Definition 5.14. Under Assumption 5.1, for PDS (4.14) *communication policy 5* is as follows:

For all $i, j \in I$ such that $i \neq j$ we have $H_{ji} = Y_{jj}$, $\Sigma_{\mathscr{R}_{ji}} = \emptyset$, $\Sigma_{\mathscr{I},j} = \Sigma_{o,j}$, and

$$\forall \sigma \in \Sigma_{o,j}.\ \mathscr{I}_{ji}(\sigma) = \{\hat{y}^k_{jj} \in Y_{jj} \mid \sigma = (\hat{y}^e_{jj}, \cdots, \hat{y}^1_{jj}) \wedge \hat{y}^k_{jj} \neq y^k_{jj} \wedge I_{com}(\sigma) = j \wedge i \notin I_o(\sigma) \wedge [\exists l \in \mathcal{O}_i \backslash \{i\}.\ l \in I_o(\sigma)]\} \quad (5.4)$$

Once $\mathscr{I}_{ji}$ is received by $\mathbf{S}_i$, the copies of event-encoding Boolean variables will be updated, i.e.

$$\begin{aligned} &\forall i, j \in I, i \neq j, \forall k \in \mathcal{U}, \forall \hat{y}^k_{jj} \in Y_{jj}, \forall \sigma \in \Sigma_{\mathscr{I},j}. \\ &\quad (1)\ [\hat{y}^k_{jj} \in \mathscr{I}_{ji}(\sigma) \Longrightarrow \hat{y}^k_{ij} := \hat{y}^k_{jj}] \wedge [\hat{y}^k_{jj} \notin \mathscr{I}_{ji}(\sigma) \Longrightarrow \hat{y}^k_{ij} := y^k_{ij}] \\ &\wedge (2)\ [\forall \beta \in \Sigma_{o,j}, \forall m \in \mathcal{O}_i \backslash \{i\}, \forall \mathbf{x}^i \in \mathbb{F}_p^{|\mathcal{O}_i|}. \\ &\qquad (\hat{y}^e_{ij}, \cdots, \hat{y}^1_{ij}) = \beta \wedge m \in I_o(\beta) \Longrightarrow \hat{x}^i_m := \mathfrak{a}^\beta_{im}(\mathbf{x}^i)] \end{aligned} \quad (5.5)$$

□

In simple words, upon the occurrence of $\sigma \in \Sigma_o$, communication policy 5 requires that one of the supervisors, that is the one specified by $I_{com}(\sigma)$ (which observes the occurrence of σ), informs the supervisors which cannot observe σ

and whose guards or updating functions are affected by its occurrence directly or indirectly, of this occurrence, as reflected in (5.4). When the sent bits arrive at the target supervisor(s), as (5.5) reads, first they are decoded by the receivers to find out which event has been observed by the sender (Conjunct 1), and then all copies of the private variables, which are kept by each receiver and are affected by the decoded event, are updated using the copies of the corresponding updating functions (Conjunct 2). Notice that policy 5 employs *constant messages*, introduced in Definition 5.1, as the content of $\mathscr{I}$ events.

Proposition 5.4 *A solution to Problem 5.1:* Associated with Problem 5.1, if the network of supervisors is strongly connected with lossless channels and if communication is instantaneous, communication policy 5 insures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

Definition 5.15 assumes a given $I_{com}(\sigma)$ for every $\sigma \in \Sigma_o$ and introduces communication policy 5 based on it. As (5.4) reads, $\mathbf{S}_{I_{com}(\sigma)}$ is responsible for issuing communication event $\mathscr{I}_{I_{com}(\sigma)i}$ to all $\mathbf{S}_i$s which cannot observe σ, but have a copy variable which can be affected by σ. Following Definition 5.14, when $I_o(\sigma)$ is a singleton, there is just one choice for $I_{com}(\sigma)$. However, if $|I_o(\sigma)| > 1$, there are different supervisors which can be the issuer of $\mathscr{I}_{I_{com}(\sigma)i}$. By Conjunct 2 of (5.5), once the communication is received, regardless of which supervisor (among those whose index is in $I_o(\sigma)$) has issued it, it results in the same process, i.e. reevaluation of all copy variables, stored by $\mathbf{S}_i$, which are affected by σ.

However, by (5.4) and Conjunct 1 of (5.5), each choice of $I_{com}(\sigma)$ would entail communication of the changed bits of $Y_{I_{com}(\sigma)I_{com}(\sigma)}$. Therefore, it is plausible to ask if there are choices of $I_{com}(\sigma)$s which entail a minimal communication in the sense of (5.3). Whereas this issue is not formally investigated here, it is worth mentioning, as a rule of thumb, that if a supervisor $\mathbf{S}_l$ has fewer number

of observable events, it has probably more constant bits in Y_{ll}. Therefore, choosing $I_{com}(\sigma) = l$ can reduce the content of $\mathscr{I}_{li}(\sigma)$. Furthermore, for $l, i \in I$ and $\sigma \in \Sigma_{o,l}$ such that $I_{com}(\sigma) = l$, if the employed event-encoding scheme for a supervisor $\mathbf{S}_l$ is such that along the system's evolution, fewer bits in Y_{ll} change with the occurrence of an event σ, the content of $\mathscr{I}_{li}(\sigma)$ would be decreased, too. This point is explained in the next example.

Example 5.6 Let us apply communication policy 5 for the DSDES in Example 4.3. From Table 4.2, we have the following.

$$\mathcal{N}_{a_1} = \mathcal{N}_{a_2} = \{1,2\},\ \mathcal{N}_{a_3} = \{3\},\ \mathcal{N}_1 = \{1,2\},\ \mathcal{N}_2 = \{1,2,3\},\ \mathcal{N}_3 = \{3\}$$

Application of Algorithm 5.1 then leads to $\mathcal{O}_1 = \{1,2\}$, $\mathcal{O}_2 = \{1,2,3\}$, and $\mathcal{O}_3 = \{3\}$.

Observe that $|\Sigma_{o,1}| = |\Sigma_{o,2}| = 2$ and $|\Sigma_{o,3}| = 1$, therefore following Definition 5.13 we have $e = \lceil log_2(2+1) \rceil = 2$. Also it holds that $I_o(\alpha_1) = \{1\}, I_o(\alpha_2) = \{2\}, I_o(\beta) = \{1,2\}$, and $I_o(\alpha_3) = \{3\}$. Following Definition 5.14 we have $I_{com}(\alpha_1) = 1, I_{com}(\alpha_2) = 2$, and $I_{com}(\alpha_3) = 3$. However, $I_{com}(\beta)$ can be either 1 or 2 and, based on the fact that $\mathbf{S}_1$ and $\mathbf{S}_2$ have each two observable events, there does not seem to be any difference between choosing either supervisors, hence we set $I_{com}(\beta) = 1$.

Codes for the events are shown in Table 5.8. Here, all supervisors encode ϵ as (00), and we have $\alpha_1 = (01), \alpha_2 = (01)$, and $\alpha_3 = (01)$ [4]. To encode β, we notice that if $\beta = (10)$, then between $\alpha_1 = (01)$ and $\beta = (10)$ there would be a difference of 2 bits, i.e. upon the occurrence of one of these events after the other one, $\mathbf{S}_1$ should send 2 bits. However, if $\beta = (11)$, the difference would be between (01) and (11), i.e. just 1 bit. Therefore, this latter choice is taken

[4]Notice that these codes are assigned to different variables (y^2_{ii}, y^1_{ii}) for $i = 1, 2$, and 3, respectively, as shown in Table 5.8.

as the code of β assigned by $\mathbf{S}_1$. Although, $\mathbf{S}_2$ need not inform others of the occurrence of β, it encodes β as (11), because (10) would require $\mathbf{S}_2$ to send 2 bits when reporting the occurrence of α_2 after β. Moreover, a common code for β can save some memory space, by reducing from two different spaces to one in the look up table of the events, and helps supervisors distinguish β consistently in the case of any faults.

Using the above codes and Definition 5.15, communication events can be computed. To this end, assume that the last event observed by $\mathbf{S}_i$ is $\sigma_i \in \Sigma_{o,i} \cup \{\epsilon\}$, encoded as (y_{ii}^2, y_{ii}^1) and the new event is encoded as $(\hat{y}_{ii}^2, \hat{y}_{ii}^1)$, where $i \in \{1, 2, 3\}$. The communication events are shown in Table 5.9, where conditions of selecting communication content are simply formulated as whether corresponding $\hat{y}$s and ys are equal to each other or not. For $\mathbf{S}_3$, it can be seen that $y_{33}^2 = 0$ holds all the time, hence simplifying the content condition. $\Diamond$

Table 5.8: Encoding of Events for Example 5.6

$\mathbf{S}_1 : (y_{11}^2, y_{11}^1)$	$\epsilon = (00)$	$\alpha_1 = (01)$	$\beta = (11)$
$\mathbf{S}_2 : (y_{22}^2, y_{22}^1)$	$\epsilon = (00)$	$\alpha_2 = (01)$	$\beta = (11)$
$\mathbf{S}_3 : (y_{33}^2, y_{33}^1)$	$\epsilon = (00)$	$\alpha_3 = (01)$	

Remark 5.3 *Communication of observed states (State-based communication) versus communication of observed events (event-based communication)*: State-based and event-based communication policies were shown to maintain the correctness of the close-loop behavior, in the sense that the centralized supervisor, which enforces the desired system specifications, is implemented correctly in

Table 5.9: Communication amongst supervisors in Example 5.6 based on policy 5

$\mathscr{I}_{12}(\alpha_1) = \begin{cases} \{\hat{y}^1_{11}, \hat{y}^2_{11}\} & ; \text{ if } (\hat{y}^1_{11} \neq y^1_{11}) \wedge (\hat{y}^2_{11} \neq y^2_{11}) \\ \{\hat{y}^1_{11}\} & ; \text{ if } (\hat{y}^1_{11} \neq y^1_{11}) \wedge (\hat{y}^2_{11} = y^2_{11}) \\ \{\hat{y}^2_{11}\} & ; \text{ if } (\hat{y}^1_{11} = y^1_{11}) \wedge (\hat{y}^2_{11} \neq y^2_{11}) \\ \emptyset & ; \text{ if } (\hat{y}^1_{11} = y^1_{11}) \wedge (\hat{y}^2_{11} = y^2_{11}) \end{cases}$
$\mathscr{I}_{13}(\alpha_1) = \mathscr{I}_{13}(\beta) = \mathscr{I}_{12}(\beta) = \emptyset$
$\mathscr{I}_{21}(\alpha_2) = \begin{cases} \{\hat{y}^1_{22}, \hat{y}^2_{22}\} & ; \text{ if } (\hat{y}^1_{22} \neq y^1_{22}) \wedge (\hat{y}^2_{22} \neq y^2_{22}) \\ \{\hat{y}^1_{22}\} & ; \text{ if } (\hat{y}^1_{22} \neq y^1_{22}) \wedge (\hat{y}^2_{22} = y^2_{22}) \\ \{\hat{y}^2_{22}\} & ; \text{ if } (\hat{y}^1_{22} = y^1_{22}) \wedge (\hat{y}^2_{22} \neq y^2_{22}) \\ \emptyset & ; \text{ if } (\hat{y}^1_{22} = y^1_{22}) \wedge (\hat{y}^2_{22} = y^2_{22}) \end{cases}$
$\mathscr{I}_{23}(\alpha_2) = \mathscr{I}_{23}(\beta) = \mathscr{I}_{21}(\beta) = \emptyset$
$\mathscr{I}_{32}(\alpha_3) = \begin{cases} \{\hat{y}^1_{33}\} & ; \text{ if } (\hat{y}^1_{33} \neq y^1_{33}) \\ \emptyset & ; \text{ if } (\hat{y}^1_{33} = y^1_{33}) \end{cases}$
$\mathscr{I}_{31}(\alpha_3) = \emptyset$

a decentralized manner. A general and rigorous comparison between the two paradigms of communication is not possible at the current stage of this research. Whereas this issue deserves a separate work, a few observations might be useful to point out.

The state-based communication, examples of which are policies 1, 2, 3, and 4, relies on the binary encoding of states, where each supervisor only saves its own dynamical equation (or equations in DSDES or EFSM framework). Accordingly, a sender transmits the latest values of the bits forming (part of) the binary encoding of its current state. The size of communication would increase if the number of state changes is high and less symmetry among states exists.

The event-based communication employs a binary encoding of events, where each supervisor should store the dynamical equations of some other supervisors on top of copies of their private variables, thereby asking for more memory space. The size of communication would increase if the number of events and the changes between their encodings are high. However, in cases

that state symmetries and the number of events are few, the event-based communication may entail less communication.

Although we did not investigate the issue, but a mixture of the two paradigms is also possible to benefit from the merits of the two and avoid unnecessary communication. □

5.2.6 Synthesis of communicating supervisors using DSDESs: a summary

The whole procedure for "synthesis of communicating decentralized controllers" in the DSDES framework may be summarized in the following steps.

1. Given the plant and specification, compute the centralized supervisor, **S**, using SCT.

2. Employ an ALM to assign (a set of) integer vectors to each state of **S** to explore its information structure (Theorem 3.2). Use the symmetries in the structure of **S** to define ALMs with smaller image size. Compute updating and guard functions (Proposition 4.3).

3. Represent updating and guard functions as polynomial equations over a finite field (Algorithm 4.1). Employ "arbitrary" cases to eliminate as many external variables as possible.

4. To derive state-based communication policies (such as policies 1 to 4) implement the above-mentioned polynomials in EFSM framework (Algorithm 4.2). To derive event-based communication policies (such as policy 5) employ an event-encoding scheme.

5. Define handshaking and information policies by analyzing the polynomials ($\mathscr{I}$ events).

6. Design a routing policy ($\mathscr{R}$ events) and, if necessary, redesign the $\mathscr{I}$ events and constant messages (this step is not studied here).

5.3 A communication-based classification of DSDESs

DSDES framework enjoys an insightful divide-and-conquer approach to modeling supervisors of DESs by capturing the observation- and control- related information (of a centralized supervisor) using two distinct means, i.e. *updating* and *guard* functions, respectively. Communication naturally arises to help a decentralized supervisor to evaluate either the former or the latter, thereby inducing a characterization in terms of "communication for observation" and "communication for control," respectively.

5.3.1 Communication for observation and communication for control

Definition 5.16 Associated with PDS (4.14) let $i, j \in I, i \neq j$. A supervisor $\mathbf{S}_i$ is said to depend on $\mathbf{S}_j$ if

$$[\exists \sigma \in \Sigma_{o,i}.\ x_j \in \arg(\mathfrak{a}_i^\sigma)] \vee [\exists \alpha \in \Sigma_{c,i}.\ x_j \in \arg(\mathfrak{g}_i^\alpha)]. \qquad \square$$

The above definition may be explained in the following way. Upon observing the occurrence of an event $\sigma \in \Sigma_{o,i}$, $\mathbf{S}_i$ must update x_i with the value $x_i := \mathfrak{a}_i^\sigma(\mathbf{x})$. However, if $x_j \in \arg(\mathfrak{a}_i^\sigma)$, $\mathbf{S}_i$ needs to know the value of x_j at least once during the evolution of the system. Similarly, an event $\alpha \in \Sigma_{c,i}$ is enabled at $\mathbf{x}$ if and only if $\mathfrak{g}_i^\alpha(\mathbf{x}) = 1$. If $x_j \in \arg(\mathfrak{g}_i^\alpha)$, to determine if the equality holds, $\mathbf{S}_i$ requires the value x_j, at least once during the system's evolution. This definition includes the case where $\mathbf{S}_i$ may require to update its

copy of x_j to reevaluate both its updating and guard functions. Based on the definitions of communication policies in Section 5.2, to update its copy of x_i, $\mathbf{S}_i$ either receives the values of the variables in (a subset of) X_{jj} (see Definition 4.9) to update its X_{ij}, or the values of a subset of Y_{jj} to update its Y_{ij} (see Definition 5.13), and, in turn, reevaluates its x_j^i. Examples of these two kinds of information are seen in information policies of communication policies 1 to 4, and in the information policy of communication policy 5, respectively.

Definition 5.16 is based on the functional dependency of $\mathbf{S}_i$'s updating or guard functions on x_j. This dependency implies that, at least once during the system's evolution, $\mathbf{S}_i$ needs to have a true copy of x_j to reevaluate its guard or updating function(s). That how often $\mathbf{S}_i$ needs to update its copy of x_j, either as many times as $\mathbf{S}_j$ does so or less, would be specified by the employed information policy, which is in turn inspired by the analysis of the PDS representation of the DSDES.

Whereas the analysis of particular functional forms associated with a PDS reveals more specific points on when each supervisor "requires" to receive information from the supervisor(s) on which it depends, there is a general observation which pertains to this issue and proves useful in formalizing a supervisor's informational requirement. That is an updating function and a guard function are different from each other in one fundamental way. A typical updating function, $\mathfrak{a}_i^\sigma$, is used to reevaluate x_i only when $\mathbf{S}_i$ observes an event $\sigma \in \Sigma_{o,i}$. Thus, if $x_j \in \arg(\mathfrak{a}_i^\sigma)$, $\mathbf{S}_i$ knows when it requires a true copy of x_j, i.e. when it observes an event $\sigma \in \Sigma_{o,i}$. However, a guard function $\mathfrak{g}_i^\alpha$ is used in the algebraic equation "$\mathfrak{g}_i^\alpha(\mathbf{x}) = 1$." Thus, it should be always reevaluated, i.e. at all states $\mathbf{x} \in \mathbb{F}_p^n$, by $\mathbf{S}_i$ to determine if event $\alpha \in \Sigma_{c,i}$ should be disabled or enabled. Therefore, if $x_j \in \mathfrak{g}_i^\alpha$, $\mathbf{S}_i$ should always have a true copy of x_j, because otherwise it may be the case that $\mathbf{S}_i$ chooses the wrong control decision, which

either violates the specification or restricts the plant's behavior unnecessarily. This difference between the informational requirements of updating and guard functions justifies the following definition.

Definition 5.17 Associated with PDS (4.14) let $i, j \in I, i \neq j$. A supervisor $\mathbf{S}_i$ is said to require a *communication for observation* (CFO) from $\mathbf{S}_j$ if there exists an event $\sigma \in \Sigma_{o,i}$ such that $x_j \in \arg(\mathfrak{a}_i^\sigma)$. Similarly, a supervisor $\mathbf{S}_i$ is said to require a *communication for control* (CFC) from $\mathbf{S}_j$ if there exists an event $\alpha \in \Sigma_{c,i}$ such that $x_j \in \arg(\mathfrak{g}_i^\alpha)$. □

Once it is known that an update of the copy of x_j is required by $\mathbf{S}_i$, it is reasonable to ask how the required information can be obtained by $\mathbf{S}_i$. This issue is explained next.

Definition 5.18 Associated with PDS (4.14), assume that an information policy is given.

1. The information policy is called *automatic* if for every $i, j \in I$, $i \neq j$ such that $\mathbf{S}_i$ depends on $\mathbf{S}_j$, once the information policy determines that $\mathbf{S}_i$ needs to have the true copy of x_j, the following holds. If x_j gets updated, i.e. after the occurrence of some event $\sigma \in \Sigma_{o,j}$, $\mathbf{S}_j$ issues an event $\mathscr{I}_{ji}(\sigma)$.

2. The information policy is called *on-demand* if for every $i, j \in I$, $i \neq j$ such that $\mathbf{S}_i$ depends on $\mathbf{S}_j$, once the information policy determines that $\mathbf{S}_i$ needs to have the true copy of x_j, the following holds. First $\mathbf{S}_i$ sends a request to $\mathbf{S}_j$ in the form of an event $\mathscr{I}_{ij}(\sigma')$ (as a handshaking message) for some $\sigma' \in \Sigma_{o,i}$, and then $\mathbf{S}_j$ issues an event $\mathscr{I}_{ji}(\mathscr{I}_{ij}(\sigma'))$.

3. The information policy is called *mixed* if for every $i, j \in I$, $i \neq j$ such that $\mathbf{S}_i$ depends on $\mathbf{S}_j$, once the information policy determines that $\mathbf{S}_i$

needs to have the true copy of x_j, the following holds. Either the required information is sent as in the case of an automatic policy or as in the case of an on-demand policy. □

Let $\mathbf{S}_i$ depend on $\mathbf{S}_j$, where $i, j \in I$, and $i \neq j$. Whereas this fact implies that $\mathbf{S}_i$ requires x_j to compute its guard or updating functions, observe that Definition 5.18 relies on the fact that it is the information policy which decides when a true copy of x_j is required by $\mathbf{S}_i$. The definition classifies the information policies, which can be defined for PDS (4.14), based on if the information required by $\mathbf{S}_i$ is sent for it without its request or not. Accordingly, an automatic information policy transfers the required information through one single $\mathscr{I}_{ji}$ event, issued by the owner of the private information, $\mathbf{S}_j$. Once the information policy decides that $\mathbf{S}_i$ needs a true copy of x_j, which has been updated as a result of observing an event $\sigma \in \Sigma_{o,j}$, this $\mathscr{I}$ event will be issued. Therefore, in the absence of losses and delays, $\mathbf{S}_i$ needs not worry about anything, but updating its copy of x_j upon receiving the communication. On the other hand, in an on-demand information policy, the owner of the required information, $\mathbf{S}_j$, waits to receive a request from $\mathbf{S}_i$, which is itself an event $\mathscr{I}_{ij}(\sigma')$, issued after an event $\sigma' \in \Sigma_{o,i}$. This request is then replied by $\mathbf{S}_j$ through an event $\mathscr{I}_{ji}(\mathscr{I}_{ij}(\sigma'))$. Finally, in a mixed information policy, both types of automatic and on-demand information transfer schemes are employed for disjoint subsets of $\mathscr{I}$ events.

The difference between the informational requirements of updating and guard functions affects the choice of the information policy, as will be explained next. The fact that an automatic information policy solely relies on the updates made by the owner of the required information, makes it generally useful for cases where $\mathbf{S}_i$ requires both a CFC and a CFO from $\mathbf{S}_j$. This is in fact the type of information policy taken by all communication policies in

Definitions 5.3, 5.4, 5.6, 5.10, and 5.15. An on-demand information policy is however, more limited to use, as stated in the following.

Lemma 5.12 Associated with PDS (4.14), assume that an information policy is given and let $i, j \in I, i \neq j$. If $\mathbf{S}_i$ requires a CFC from $\mathbf{S}_j$, but does not require any CFOs from $\mathbf{S}_j$, then an on-demand information policy cannot be employed to provide $\mathbf{S}_i$ of the true copy of x_j. ■

The above lemma states that when a CFC is required by a supervisor, an on-demand policy cannot be employed singly to guarantee the correct revaluation of guards. However, if no supervisor requires any CFCs from other supervisor(s), an on-demand policy may be employed, successfully. This is shown in the next subsection.

5.3.2 PDSs with independent guard functions

In this subsection, we study the PDSs whose supervisors do not require any CFCs. Following Definition 5.17, for every such $\mathbf{S}_i$ it holds that its every guard function is solely a function of x_i. We start by introducing this property at the DSDES level.

Definition 5.19 Associated with the DSDES in Proposition 4.3, let $\mathcal{V} = \{\mathbf{v} \in \ell(r) \mid r \in R\}$, $i \in I$, and $\sigma \in \Sigma_{c,i}$. A guard function $\mathcal{G}_i(\sigma)$ is called an *independent guard function* (IGF) if

$$\forall \mathbf{v}, \mathbf{v}' \in \mathcal{V}. \quad \mathbf{v} \in \mathcal{G}_i(\sigma) \ \wedge \ v_i = v_i' \implies \mathbf{v}' \in \mathcal{G}_i(\sigma). \qquad \square$$

In simple words, if a label is in an IGF $\mathcal{G}_i(\sigma)$, every other label whose ith component is the same as that label is in the IGF, too. In a PDS representation of a DSDES, IGFs appear as functions which depend on their associated supervisor's private variable, only. The following result clarifies this fact.

Lemma 5.13 Let PDS (4.14) represent the DSDES in Proposition 4.3, $i \in I$, and $\sigma \in \Sigma_{c,i}$. For every IGF $\mathcal{G}_i(\sigma)$ it is possible to compute $\mathfrak{g}_i^\sigma$ such that $\arg(\mathfrak{g}_i^\sigma) = \{x_i\}$.

Proof: Recalling Algorithm 4.1, we have the following.

$$\forall i \in I,\ \forall \sigma \in \Sigma. \begin{cases} \mathfrak{g}_i^\sigma(\mathbf{v}) = 1; & \forall \mathbf{v} \in \mathcal{G}_i(\sigma) \\ \mathfrak{g}_i^\sigma(\mathbf{v}) = 0; & \forall \mathbf{v} \in \cup_{r \in R} \ell(r) \setminus \mathcal{G}_i(\sigma) \\ \mathfrak{g}_i^\sigma(\mathbf{v}) = \text{arbitrary}; & \forall \mathbf{v} \in \mathbb{F}_p^n \setminus \cup_{r \in R} \ell(r) \end{cases}$$

Next, for every $\mathbf{v} \in \mathbb{F}_p^n \setminus \cup_{r \in R} \ell(r)$ and each $\mathbf{v}' \in \cup_{r \in R} \ell(r)$ do the following assignment.

$$[\mathfrak{g}_i^\sigma(\mathbf{v}') = 1 \wedge v_i = v_i' \implies \mathfrak{g}_i^\sigma(\mathbf{v}) = 1] \text{ and}$$
$$[\mathfrak{g}_i^\sigma(\mathbf{v}') = 0 \wedge v_i = v_i' \implies \mathfrak{g}_i^\sigma(\mathbf{v}) = 0]$$

This, together with the fact that $\mathcal{G}_i(\sigma)$ is an IGF, and Definition 5.19, lead to the following.

$$\forall i \in I,\ \forall \sigma \in \Sigma, \forall \mathbf{v}, \mathbf{v}' \in \mathbb{F}_p^n.\quad v_i = v_i' \iff \mathfrak{g}_i^\sigma(\mathbf{v}) = \mathfrak{g}_i^\sigma(\mathbf{v}'),$$

i.e. every such $\mathfrak{g}_i^\sigma$ is a function of x_i only, and $\arg(\mathfrak{g}_i^\sigma) = \{x_i\}$. ■

For a PDS whose guard functions are all IGFs, we introduce a communication policy, whose information policy is on-demand. We start by defining the set of the copy variables which keeps, at the transmitter site, the last values of the private variables of the transmitter, which are sent to another supervisor.

Definition 5.20 Associated with PDS (4.14), for every $i, j \in I, i \neq j$, such that $\mathbf{S}_i$ depends on $\mathbf{S}_j$, define $X_{i,jj} = \{x_{i,jj}^k \mid k \in J\}$ as a set, kept by $\mathbf{S}_j$, whose element $x_{i,jj}^k$ stores the value of $x_{jj}^k \in X_{jj}$ which is sent by $\mathbf{S}_j$ to $\mathbf{S}_i$. Let every variable in $X_{i,jj}$ is initialized to 0. Furthermore, the values in $X_{i,jj}$ are updated as follows.

$$\forall i, j \in, i \neq j, \forall k \in J, \forall \sigma \in \Sigma_{\mathscr{I},j}.\quad x_{jj}^k \in \mathscr{I}_{ji}(\sigma) \implies x_{i,jj}^k = x_{jj}^k \qquad \square$$

In other words, under the instantaneous and lossless communication channels, $x^k_{i,jj}$ has the same value as x^k_{ij}. These sets of copy variables are used in communication policy 6, explained next.

Definition 5.21 Associated with PDS (4.14), for every $i \in I$ and each $\sigma \in \Sigma_{c,i}$ assume that $\mathfrak{g}^\sigma_i$ is an IGF. For PDS (4.14) and its EFSM implementation (4.15), *communication policy 6* is as follows: For every $i, j \in I, i \neq j$, let y_{ji} be a Boolean variable equal to 1, and define the following.

$H_{ji}=\{y_{ji}\}$, $\Sigma_{\mathscr{R}_{ji}}=\emptyset$, $\Sigma_{\mathscr{I},j}=\Sigma_{o,j} \cup \{\mathscr{I}_{kj}(\sigma) \mid k \in I \backslash \{j\} \wedge \sigma \in \Sigma_{o,k} \wedge x_j \in \arg(\mathfrak{a}^\sigma_k)\}$,

1) $\forall \sigma \in \Sigma_{o,j}$. $\mathscr{I}_{ji}(\sigma)=\begin{cases} \{y_{ji}\}; & \text{if } x_i \in \arg(\mathfrak{a}^\sigma_j) \\ \emptyset; & \text{otherwise} \end{cases}$, and

2) $\forall \mathscr{I}_{ij}(\sigma) \in \Sigma_{\mathscr{I},j} \setminus \Sigma_{o,j}$. $\mathscr{I}_{ji}(\mathscr{I}_{ij}(\sigma))=\{x^k_{jj} \in X_{jj} \mid x^k_{jj} \neq x^k_{i,jj} \wedge k \in J\}$.

Here, variable $x^k_{i,jj} \in X_{i,jj}$ is as introduced and updated in Definition 5.20. □

Proposition 5.5 *A solution to a special case of Problem 5.1:* Associated with Problem 5.1, if every guard function in PDS (4.14) is independent, the network of supervisors is strongly connected with lossless channels and if communication is instantaneous, communication policy 6 insures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. ■

Example 5.7 Figure 5.4 shows a plant **G** and a specification **S** for it. Having $I = \{1, 2\}$, $\Sigma_{o,1} = \Sigma_{c,1} = \{\alpha_1, \beta_1\}$, and $\Sigma_{o,2} = \Sigma_{c,2} = \{\alpha_2\}$, it can be verified that **S** is also a centralized supervisor. An ALM ℓ is defined for **S** as follows.

$$\ell(r_0) = \{(0,0)\},\ \ell(r_1) = \{(1,0)\},\ \ell(r_2) = \{(2,0)\},\ \ell(r_3) = \{(2,1)\},$$
$$\ell(r_4) = \{(1,1)\}, \ell(r_5) = \{(0,1)\},\ \ell(r_6) = \{(3,1)\}$$

As a result, we have the following guard functions and the updating functions in Table 5.10.

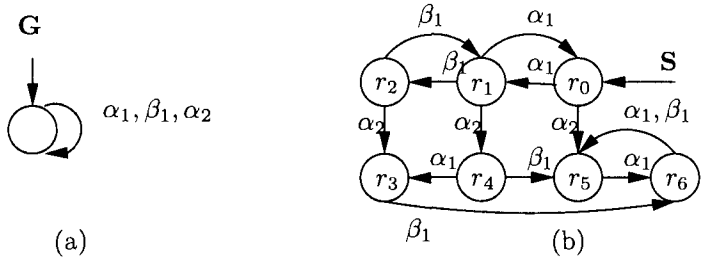


Figure 5.4: (a) A plant **G**. (b) A specification and centralized supervisor **S**.

Table 5.10: Updating functions for the DSDES in Example 5.7

$\mathbf{v}$	$0,0$	$1,0$	$1,1$	$0,1$	$3,1$
$\mathcal{A}(\alpha_1, \mathbf{v})$	$1,0$	$0,0$	$2,1$	$3,1$	$0,1$
$\mathbf{v}$	$1,0$	$2,0$	$2,1$	$1,1$	$3,1$
$\mathcal{A}(\beta_1, \mathbf{v})$	$2,0$	$1,0$	$3,1$	$0,1$	$0,1$
$\mathbf{v}$	$0,0$	$1,0$	$2,0$		
$\mathcal{A}(\alpha_2, \mathbf{v})$	$0,1$	$1,1$	$2,1$		

$$\mathcal{G}_1(\alpha_1) = \ell(r_0) \cup \ell(r_1) \cup \ell(r_4) \cup \ell(r_5) \cup \ell(r_6) = \{(0,0), (1,0), (1,1), (0,1), (3,1)\},$$

$$\mathcal{G}_1(\beta_1) = \ell(r_1) \cup \ell(r_2) \cup \ell(r_3) \cup \ell(r_4) \cup \ell(r_6) = \{(1,0), (2,0), (2,1), (1,1), (3,1)\},$$

$$\mathcal{G}_2(\alpha_2) = \ell(r_0) \cup \ell(r_1) \cup \ell(r_2) = \{(0,0), (1,0), (2,0)\}.$$

Observe that the common finite field is $\mathbb{F}_5 = \{0,1,2,3,4\}$, based on which a PDS representation of the DSDES is shown in Table 5.11. To compute the updating and guard functions, the following assumptions are made on the unreachable points in $\mathbb{F}_5^2$.

- To compute $\mathfrak{g}_1^{\alpha_1}$, we assume that α_1 is also enabled at points $(i,2), (i,3), (i,4)$, where $i \in \{0,1,3\}$, and at $(3,0)$, all unreachable.

Table 5.11: Computed polynomials for updating and guard functions of the DSDES in Example 5.7

$x_1 := \mathfrak{a}_1^{\alpha_1}(\mathbf{x}) = 2(x_1+1)(x_1+2)(x_1+3)(x_2+1)(x_2+2)(x_2+3)$ $(3x_1x_2+x_1+4x_2+3)$
$x_1 := \mathfrak{a}_1^{\beta_1}(\mathbf{x}) = x_1(x_1+1)(x_1+2)(x_2+1)(x_2+2)(x_2+3)(x_1x_2+2x_1+2x_2+2)$
$x_2 := \mathfrak{a}_2^{\alpha_2}(\mathbf{x}) = 4(x_2+1)(x_2+2)(x_2+3)(x_2+4)$
$\mathfrak{g}_1^{\alpha_1}(\mathbf{x}) = (x_1+1)(x_1+3)(2x_1^2+3x_1+2)$
$\mathfrak{g}_1^{\beta_1}(\mathbf{x}) = 4x_1(x_1+1)(3x_1^2+3x_1+1)$
$\mathfrak{g}_2^{\alpha_2}(\mathbf{x}) = 4(x_2+1)(x_2+2)(x_2+3)(x_2+4)$

- To compute $\mathfrak{g}_1^{\beta_1}$, we assume that β_1 is also enabled at points $(i,2),(i,3),(i,4)$, where $i \in \{1,2,3\}$, and at $(3,0)$, all unreachable.
- To compute $\mathfrak{g}_2^{\alpha_2}$, we assume that α_2 is also enabled at points $(3,0)$ and $(4,0)$, both unreachable.
- To compute $\mathfrak{a}_2^{\alpha_2}$, we assume that x_2 is 1 in the next states of points $(3,4)$ and $(4,0)$, reached upon the occurrence of α_2. Notice that these two points are both unreachable.
- No further assumption is made for computation of $\mathfrak{a}_1^{\alpha_1}$ and $\mathfrak{a}_1^{\beta_1}$.

Observe that all the guard functions are independent. To use Algorithm 4.2, observe that $h = 3$ and $x_i = (x_{ii}^3, x_{ii}^2, x_{ii}^1)$, for each $i \in \{1,2\}$. Guards and actions are then computed as in Table 5.12. To simplify the guards and actions, we make use of the unreachable points, as explained in the following.

- To compute $g_1(\alpha_1)$, we assume that α_1 is enabled when $(x_{11}^3, x_{11}^2, x_{11}^1)$ is (101) and (111), too. To compute $g_1(\beta_1)$, we assume that β_1 is enabled when $(x_{11}^3, x_{11}^2, x_{11}^1)$ is (101), (111), and (110), too.

- To compute $a_1(x_{11}^1, \alpha_1)$, we assume that when the Boolean encoding $(x_{11}^3 x_{11}^2 x_{11}^1, x_{22}^3 x_{22}^2 x_{22}^1)$ is equal to (i, j), where $i \in \{2, 4, 6\}$ and $j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ and when $i = 0$ and $j \in \{2, 3, 4, 5, 6, 7\}$, in the next state reached upon the occurrence of α_1, we have $x_{11}^1 = 1$. To compute $a_1(x_{11}^2, \alpha_1)$, we assume that when encoding $(x_{11}^3 x_{11}^2 x_{11}^1, x_{22}^3 x_{22}^2 x_{22}^1)$ is equal to (i, j), where $i \in \{0, 1\}$ and $j \in \{3, 5, 7\}$, in the next state reached upon the occurrence of α_1, we have $x_{11}^2 = 1$.

- To compute $a_1(x_{11}^1, \beta_1)$, we assume that when the Boolean encoding $(x_{11}^3 x_{11}^2 x_{11}^1, x_{22}^3 x_{22}^2 x_{22}^1)$ is equal to (i, j), where $i \in \{0, 4, 6\}$ and $j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ and when $i = 2$ and $j \in \{2, 3, 4, 5, 6, 7\}$, in the next state reached upon the occurrence of β_1, we have $x_{11}^1 = 1$. To compute $a_1(x_{11}^2, \beta_1)$, we assume that when encoding $(x_{11}^3 x_{11}^2 x_{11}^1, x_{22}^3 x_{22}^2 x_{22}^1)$ is equal to (i, j), in the next state reached upon the occurrence of β_1, we have $x_{11}^2 = 1$. Here, $i \in \{0, 4, 6\}$ and $j \in \{1, 3, 5, 7\}$, or $i = 2$ and $j \in \{3, 5, 7\}$, or $i \in \{0, 1\}$ and $j \in \{2, 4, 6\}$, or $i = 0$ and $j = 0$.

- No further assumption is made to compute $g_2(\alpha_2)$ or $\mathbf{S}_2$'s associated actions.

Following communication policy 6 and using Tables 5.11 and 5.12, the communication among the two supervisors is as in Table 5.13. Observe that only $\mathbf{S}_1$ depends on $\mathbf{S}_2$, i.e. only the former needs to receive information from the latter. Accordingly, since both $\mathfrak{a}_1^{\alpha_1}$ and $\mathfrak{a}_1^{\beta_1}$ depend on x_2, each time $\mathbf{S}_1$ observes α_1 or β_1, it sends a request to $\mathbf{S}_2$. Upon the receipt of this request, $\mathbf{S}_2$ sends a copy of x_{22}^1, which is the only Boolean variable which can change, to $\mathbf{S}_1$ if $\mathbf{S}_2$ finds out $\mathbf{S}_1$ does not have the latest copy of this variable, i.e. if $x_{22}^1 \neq x_{1,22}^1$. Notice that $x_{1,22}^1$ is updated each time x_{22}^1 is sent out according to

Table 5.12: Actions and guards for Example 5.7 in EFSM framework

$a_1(x_{11}^3, \alpha_1) = 0$
$a_1(x_{11}^2, \alpha_1) = \overline{x}_{11}^1$
$a_1(x_{11}^1, \alpha_1) = \overline{x}_{11}^3 \overline{x}_{11}^2 x_{12}^1$
$a_1(x_{11}^3, \beta_1) = 0$
$a_1(x_{11}^2, \beta_1) = \overline{x}_{11}^1$
$a_1(x_{11}^1, \beta_1) = \overline{x}_{11}^3 \overline{x}_{11}^2 \overline{x}_{11}^1 + \overline{x}_{11}^1 x_{12}^1$
$a_2(x_{22}^3, \alpha_2) = 0$
$a_2(x_{22}^2, \alpha_2) = 0$
$a_2(x_{22}^1, \alpha_2) = \overline{x}_{22}^1$
$g_1(\alpha_1) = \overline{x}_{11}^3 \overline{x}_{11}^2 + x_{11}^1$
$g_1(\beta_1) = x_{11}^1 + x_{11}^2$
$g_2(\alpha_2) = \overline{x}_{22}^1$

Definition 5.20. ◊

Table 5.13: Communication amongst supervisors in Example 5.7 based on policy 6

$\mathscr{I}_{12}(\alpha_1) = \mathscr{I}_{12}(\beta_1) = \{1\}$
$\mathscr{I}_{21}(\mathscr{I}_{12}(\alpha_1)) = \mathscr{I}_{21}(\mathscr{I}_{12}(\beta_1)) = \begin{cases} \{x_{22}^1\} & ; \text{ if } (x_{22}^1 \neq x_{1,22}^1) \\ \emptyset & ; \text{ if } (x_{22}^1 = x_{1,22}^1) \end{cases}$

5.3.3 PDSs with independent updating functions

An updating function also can be independent of external variables. If all updating functions associated with a supervisor $\mathbf{S}_i$ are independent of external variables, then $\mathbf{S}_i$'s observation of the system's behavior is independent of others. In the sense of Definition 5.17, this is equivalent to saying that $\mathbf{S}_i$ does not require any CFOs. In the following, we first formalize the notion of an *independent updating function* and then show its advantage.

Definition 5.22 Associated with the DSDES in Proposition 4.3, let $\mathcal{V} = \{\mathbf{v} \in \ell(r) \mid r \in R\}$, $i \in I$, and $\sigma \in \Sigma_{o,i}$. An updating function $\mathcal{A}_i(\sigma, .)$ is called an *Independent Updating Function* (IUF) if

$$\forall \mathbf{v}, \mathbf{v}' \in \mathcal{V}. \quad v_i = v_i' \implies \mathcal{A}_i(\sigma, \mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{v}'). \qquad \square$$

In simple words, the value of function $\mathcal{A}_i(\sigma, .)$ solely depends on the value of the ith component of $\mathbf{v} \in \mathcal{V}$. In a PDS representation of a DSDES, IUFs appear as functions which depend on their associated supervisor's private variable, only. The following result clarifies this point.

Lemma 5.14 Let PDS (4.14) represent the DSDES in Proposition 4.3, $i \in I$, and $\sigma \in \Sigma_{o,i}$. If $\mathcal{A}_i(\sigma, .)$ is an IUF, it is possible to design $\mathfrak{a}_i^\sigma$ such that $\arg(\mathfrak{a}_i^\sigma) = \{x_i\}$.

Proof: Recalling Algorithm 4.1, we have the following.

$$\forall i \in I, \ \forall \sigma \in \Sigma. \begin{cases} \mathfrak{a}_i^\sigma(\mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{v}); & \forall \mathbf{v} \in \mathcal{G}_i(\sigma) \\ \mathfrak{a}_i^\sigma(\mathbf{v}) = \text{arbitrary}; & \forall \mathbf{v} \in \mathbb{F}_p^n \setminus \mathcal{G}_i(\sigma) \end{cases}$$

Next, for every $\mathbf{v} \in \mathbb{F}_p^n \setminus \mathcal{G}_i(\sigma)$ and each $\mathbf{v}' \in \mathcal{G}_i(\sigma)$ do the following assignment.

$$v_i = v_i' \implies \mathfrak{a}_i^\sigma(\mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{v}')$$

This, together with the fact that $\mathcal{A}_i(\sigma)$ is an IUF and Definition 5.22, lead to the following.

$$\forall i \in I, \ \forall \sigma \in \Sigma, \forall \mathbf{v}, \mathbf{v}' \in \mathbb{F}_p^n. \quad v_i = v_i' \implies \mathfrak{a}_i^\sigma(\mathbf{v}) = \mathfrak{a}_i^\sigma(\mathbf{v}'),$$

i.e. every such $\mathfrak{a}_i^\sigma$ is a function of x_i only, and $\arg(\mathfrak{a}_i^\sigma) = \{x_i\}$. ■

Let $i, j, k \in I$, $i \neq j$, and $k \neq j$. If one of the $\mathbf{S}_i$'s associated updating functions depend on x_j, any errors in the copy of x_j, which is stored by $\mathbf{S}_i$, will affect the correctness of x_i, too. Such an error might be the result of a lossy communication channel between the two supervisors or as a result of an error in computing x_j in $\mathbf{S}_j$, which might have resulted from an erroneous copy of an

external variable x_k, on which an $\mathbf{S}_j$'s updating function depends. Motivated by Definition 5.11, the following definition specifies the set of supervisors whose private variables may affect x_i.

Definition 5.23 Recall $\mathcal{N}_{\mathfrak{a}_i} = \{j \in I \mid \exists \sigma \in \Sigma_{o,i}.\ x_j \in \arg(\mathfrak{a}_i^\sigma)\}$ for every $i \in I$, from Definition 5.11. Define set $\mathcal{O}_{\mathfrak{a}_i} \subseteq I$ recursively as follows.

$$\mathcal{N}_{\mathfrak{a}_i} \subseteq \mathcal{O}_{\mathfrak{a}_i} \quad \wedge \quad [\forall j \in I.\ j \in \mathcal{O}_{\mathfrak{a}_i} \implies \forall k \in \mathcal{N}_{\mathfrak{a}_j}.\ k \in \mathcal{O}_{\mathfrak{a}_i}] \qquad \square$$

Lemma 5.15 We have the following.

$$\forall i \in I, \forall j \in \mathcal{O}_{\mathfrak{a}_i}, \forall \sigma \in \Sigma_{o,j}.\ x_k \in \arg(\mathfrak{a}_j^\sigma) \implies k \in \mathcal{O}_{\mathfrak{a}_i}$$

Proof: Follows directly from Definition 5.23. ■

What the above definition and lemma suggest is that the correct reevaluation of x_i depends on the availability of the correct copies of x_j to $\mathbf{S}_i$, where $j \in \mathcal{N}_{\mathfrak{a}_i}$. Also correct evaluation of the set of such x_js, in turn, depends on the availability of the correct copies of x_ks to the corresponding $\mathbf{S}_j$s, where k belongs to a subset of $\mathcal{O}_{\mathfrak{a}_i}$.

The dependency of x_i on the external variables can potentially make it erroneous in the occasion of a communication error, say one which may happen in lossy channels. As a result, the whole PDS would become prone to propagation errors. Furthermore, once x_i is computed wrongly as a result of an erroneous copy, of say x_j, in general $\mathbf{S}_i$ cannot reevaluate it correctly by only receiving the correct copy of x_j. To explain this, assume that for some $\sigma \in \Sigma_{o,i}$ we have $x_i := \mathfrak{a}_i^\sigma(x_i, x_j)$. Denote by v_i, $\hat{v}_i^e$, v_j, and v_j^e the (correct) current value of x_i (before observing σ), the wrong next value of x_i (after observing σ), the correct copy of x_j, and the erroneous copy of x_j, respectively. Assume that first the wrong value of x_i is computed, i.e. $v_i^e := \mathfrak{a}_i^\sigma(v_i, v_j^e)$. As a result, $\mathbf{S}_i$ enters a "faulty state," at which wrong control decisions may be made. If now $\mathbf{S}_i$ wants to get recovered from this condition, it should not only receive v_j, but

also use v_i, rather than v_i^e. This implies that $\mathbf{S}_i$ should store the values of x_i as they change along the system's evolution. Now assume that $\mathbf{S}_i$'s updating functions does not depend on external variables and $\mathbf{S}_i$ requires no CFOs from another supervisor, say $\mathbf{S}_j$. In this case, x_i is always "trustable," and even if v_j^e is received, upon which $\mathbf{S}_i$ enters a "faulty mode" issuing wrong control decisions, it can be recovered from this condition through simply receiving v_j, with no need to store the information on the previous values of x_i.

The above argument on "fault detection and recovery" is beyond the scope of this work and is not formalized here. However, it gives some insight to distinguish how "safe" a PDS with IUFs is and why "fault recovery" is simpler for it. This informal discussion motivates the study of IUFs, their characterization, and existence conditions in more details, which is done in Section 5.4.

5.4 IUFs, DSDES behavior, and state representations

This section continues our study of PDSs with independent updating functions by first characterizing IUFs, and studying conditions of their existence, and the ALMs which can lead to IUFs.

5.4.1 Weak joint observability

It turns out that the existence of a DSDES possessing only independent updating functions, depends directly on the structure of the closed and marked languages of $\mathbf{S}$. In particular, this is related to a property of the language of supervisor $\mathbf{S}$, called *weakly joint observability.* This property, which is motivated by a definition of *joint observability* in [28], was originally defined in [47]

and is extended to include "marking" now. In simple words, joint observability requires that within the closed behavior (respectively, marked behavior) of the plant, any legal-illegal pair of strings (respectively, any marked-unmarked pair of strings) can be told apart by at least one supervisor. To simplify the notation, let us first define two equivalence relations.

Definition 5.24 Two strings $s, s' \in \Sigma^*$ are called *observationally equivalent*, denoted by $s \equiv_I s'$, if for all $i \in I$ it holds that $P_i(s) = P_i(s')$. Also let $P_{uo} : \Sigma^* \to \Sigma_{uo}^*$ be the natural projection which erases from a string $s \in \Sigma^*$ all its observable events. Define $s \equiv_U s'$ if $P_{uo}(s) = P_{uo}(s')$. □

It can be easily shown that both $\equiv_I$ and $\equiv_U$ are reflexive, symmetric, and transitive, hence proving the following result.

Lemma 5.16 Relations $\equiv_I$ and $\equiv_U$ are equivalence relations on Σ^*. ■

Denote the equivalence class of $s \in \Sigma^*$ by $[s] = \{s' \mid s' \equiv_I s\}$. Recall from [30] that two strings $s, s' \in \Sigma_*$ are *trace equivalent* if $s \equiv_I s'$ and $s \equiv_U s'$, therefore two observationally equivalent strings are not necessarily *trace equivalent* as their natural projections onto events in Σ_{uo} might be different. A *jointly observable* language is characterized as follows[5].

Lemma 5.17 (Lemma 3.1, [84]) Let L and K be two nonempty languages such that[6] $K \subseteq L \subseteq \Sigma^*$. K is called Jointly Observable (JO) with respect to L and $(\Sigma_{o,1}, \cdots, \Sigma_{o,n})$ if and only if

$$\forall s \in K,\ \forall s' \in L \setminus K,\ \exists i \in I.\ P_i(s) \neq P_i(s'). \qquad \blacksquare$$

[5]Lemma 5.17 was originally introduced as the definition of a jointly observable language in [28], but was shown later to be equivalent to a new definition for—what is then called—observable languages in [84]. Here, for notational convenience we choose the original definition.

[6]While the original definition is for any two languages K and L, we assume that $K \subseteq L$. This is plausible since we use the definition for the case where L and K are respectively the plant's and the centralized supervisor's languages (see Assumption 5.4).

For control purposes one should differentiate between closed and marked languages. Having this in mind, Lemma 5.17 can be equivalently stated for a language and its prefix-closure as follows.

Lemma 5.18 Let L and K be two nonempty languages such that $K \subseteq L \subseteq \Sigma^*$. K is called Jointly Observable (JO) with respect to L and $(\Sigma_{o,1}, \cdots, \Sigma_{o,n})$ if and only if

$$\forall s, s' \in \overline{L}.\ \ s \in \overline{K} \ \wedge\ s \equiv_I s' \implies s' \in \overline{K}. \tag{5.6}$$

$$\forall s, s' \in \overline{K}.\ \ s \in K \ \wedge\ s \equiv_I s' \implies s' \in K. \tag{5.7}$$

■

Weak joint observability weakens the requirements of joint observability by asking for the distinguishing property between legal strings and the *minimal-length* illegal strings as defined next. The reason is that, from a control perspective, one cares about the first illegal move regardless of any of its feasible future behavior in the plant.

Definition 5.25 Let L and K be two nonempty languages such that $K \subseteq L \subseteq \Sigma^*$. K is called Weakly Jointly Observable (WJO) with respect to L and $(\Sigma_{o,1}, \cdots, \Sigma_{o,n})$ if the following conditions hold.

$$\forall s, s' \in \overline{K},\ \forall \sigma \in \Sigma.\quad s\sigma \in \overline{K} \ \wedge\ s'\sigma \in \overline{L} \ \wedge\ s \equiv_I s' \Longrightarrow s'\sigma \in \overline{K} \tag{5.8}$$

$$\forall s, s' \in \overline{K}.\quad s \in K \ \wedge\ s \equiv_I s' \Longrightarrow s' \in K \tag{5.9}$$

□

When it is clear from the context, $(\Sigma_{o,1}, \cdots, \Sigma_{o,n})$ will not be mentioned.

Lemma 5.19 Let $\emptyset \neq K \subseteq L \subseteq \Sigma^*$. If K is JO with respect to L, K is WJO with respect to L. ■

Assumption 5.4 As can be seen, weak joint observability assumes that $K \subseteq L$. In general, when K and L are respectively taken to be $L_m(\mathbf{S})$ and $L_m(\mathbf{G})$, where $\mathbf{S}$ is the supervisor, designed using supervisory control theory, for plant and $\mathbf{G}$, this does not hold. However, such a supervisor $\mathbf{S}$ implements the supervisory control map in the following sense:

$$L_m(\mathbf{S}) \cap L_m(\mathbf{G}) = K \ \wedge \ L(\mathbf{S}) \cap L(\mathbf{G}) = \overline{K}. \tag{5.10}$$

Therefore, in the subsequent discussion, we may safely assume that $L_m(\mathbf{S}) \subseteq L_m(\mathbf{G})$. □

We finish this part by proving a result which is used later in subsection.

Definition 5.26 A language $A \subseteq \Sigma^*$ is called *trace-closed* if the following holds.

$$\forall s, s' \in \Sigma^*. \ \ s \equiv_I s' \ \wedge \ s \equiv_U s' \ \wedge s \in A \implies s' \in A \qquad \square$$

Trace-closedness is inherited by K when it is jointly observable with respect to L.

Lemma 5.20 Let $\emptyset \neq K \subseteq L \subseteq \Sigma^*$, L be trace-closed, and K be JO with respect to L. Then K is trace-closed.

5.4.2 Undecidability of verification of weak joint observability

It is proved in [31] that checking joint observability of K is decidable if L is trace-closed, otherwise it is undecidable in general. Inspired by the proof,

in the following it is shown that verification of weak joint observability is undecidable. We begin by recalling the "Post correspondence problem."

Problem 5.2 (*Post Correspondence Problem (PCP)* [85]) Fix a finite alphabet Γ and let $\mathcal{M}$ be an infinite set of dominos of $m \in \mathbb{Z}^+$ types, where a domino of type i, denoted by d_i $(1 \leq i \leq m)$, is characterized by a top string $u_i \in \Gamma^+$ and a bottom string $w_i \in \Gamma^+$.
A match is a sequence of k dominos $d_{i_1} d_{i_2} \ldots d_{i_k}$, where $\forall 1 \leq l \leq k.\ i_l \in \{1, 2, \ldots, m\}$ such that the concatenated top string is identical to the concatenated bottom string, i.e. $u_{i_1}.u_{i_2}.\ldots.u_{i_k} = w_{i_1}.w_{i_2}.\ldots.w_{i_k}$.
Given an instance of Post correspondence problem, does there exist a match? ■

To avoid trivial solutions we assume that for all $1 \leq i \leq m$ we have $u_i \neq w_i$ (otherwise d_i would be a match of length 1). It is known that PCP is undecidable [85].

Example 5.8 Let $\Gamma = \{a, b\}$, $m = 2$, $d_1 = \frac{aab}{b}$ and $d_2 = \frac{a}{aa}$. There is a match of length 3 with $i_1 = i_2 = 2$ and $i_3 = 1$.

a	a	aab
aa	aa	b

◊

For two *regular* languages K and L over Σ with $K \subseteq L$, we show that the problem of deciding whether K is WJO with respect to L is undecidable by reducing PCP to WJO^c, namely, an instance of PCP has a match if in the corresponding instance of WJO K is not WJO[7] with respect to L.

Theorem 5.6 Weak joint observability is undecidable.

[7]With abuse of notation, WJO is used both as an abbreviation, and as the name of the corresponding decision problem.

5.4.3 Weak joint observability as a necessary condition for the existence of IUFs

This part establishes one side of the relationship between the language of a centralized supervisor, i.e. a behavior, and its state realization. The result was first shown in [79] in EFSM framework and, for the sake of completeness, is proved here in DSDES framework.

Lemma 5.21 Associated with the DSDES in Proposition 4.3, let $i \in I$ and for each $\sigma \in \Sigma_{o,i}$ assume that the updating function $\mathcal{A}_i(\sigma, .)$ is independent. Then the following holds.

$$\forall s, s' \in L(\mathbf{S}),\ \forall \mathbf{v} \in \mathcal{V}.\ \ P_i(s) = P_i(s') \implies \mathcal{A}_i(s, \mathbf{v}) = \mathcal{A}_i(s', \mathbf{v})$$

Corollary 5.1 Associated with the DSDES in Proposition 4.3, let $i \in I$ and for each $\sigma \in \Sigma_{o,i}$ assume that the updating function $\mathcal{A}_i(\sigma, .)$ is independent. Then the following holds.

$$\forall s \in L(\mathbf{S}),\ \forall \mathbf{v} \in \mathcal{V}.\ \ \mathcal{A}_i(s, \mathbf{v}) = \mathcal{A}_i(P_i(s), \mathbf{v})$$

Proof: Lemma 5.21 holds for every $s, s' \in L(\mathbf{S})$, including the case where $s' = P_i(s)$. ■

The above results imply that every two observationally equivalent strings lead to the same state.

Corollary 5.2 Associated with the DSDES in Proposition 4.3, for every $i \in I$ and for each $\sigma \in \Sigma_{o,i}$ assume that the updating function $\mathcal{A}_i(\sigma, .)$ is independent. Then the following holds.

$$\forall i \in I, \forall s, s' \in L(\mathbf{S}), \forall \mathbf{v} \in \mathcal{V}.\ s \equiv_I s' \implies \mathcal{A}_i(s, \mathbf{v}) = \mathcal{A}_i(s', \mathbf{v}) = \mathcal{A}_i(P_i(s), \mathbf{v})$$

Proof: Corollary 5.1 holds for every $\mathbf{v} \in \mathcal{V}$ and specifically for $\mathbf{v} = \mathbf{0}$. ■

Proposition 5.7 Associated with the DSDES in Proposition 4.3, let $\ell(.)$ be an ALM which labels the states of $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ such that for every $i \in I$ and each $\sigma \in \Sigma_{o,i}$, $\mathcal{A}_i(\sigma, .)$ is an IUF. Then $L_m(\mathbf{S})$ is WJO with respect to $L_m(G)$.

5.4.4 Some state representations which lead to IUFs

By Proposition 5.7, if for every supervisor each updating function associated with every event is independent of other supervisors's private information, the behavior of the centralized supervisor enjoys WJO property. The other side of this relationship, i.e. if for a WJO behavior we can compute IUFs for every supervisor and associated with all its events, seems much more involved. First of all, there are different possible state representations for a given behavior and there is not a sufficiently rich theory which determines which ones are amenable to IUFs. Second, given such a state representation, it can be shown, through examples, that the existence of IUFs depends on the choice of the associated ALM (see Chapter 3 or [50]), too. Moreover, a general constructive procedure for finding a suitable ALM for a specific representation of the behavior that yields IUFs is not yet known to exist (even worse, the problem may be undecidable). Therefore, the sufficient condition(s) for the existence of IUFs apparently are not restricted to WJO property and include state representations as well as ALM assignment of state labels.

This motivates the study of specific constructions for which IUFs can be computed. To this end, two types of results are derived here. The first type studies specific state representations which are amenable to IUFs and the second class of results introduce specific constructive procedures for ALMs which yield IUFs.

Table 5.14: Updating functions for $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ of Example 5.9.

σ_1	$\mathbf{v}$	$\mathcal{A}_1(\sigma_1, \mathbf{v})$	σ_2	$\mathbf{v}'$	$\mathcal{A}_2(\sigma_2, \mathbf{v}')$
α_1	0	1	α_2	0	1
α_1	2	1	α	1	0
β_1	1	0	β_2	0	0
α	0	2	—	—	—
α	2	0	—	—	—

Lemma 5.22 For $i \in I$ let $A_i \subseteq \Sigma_i^*$ be a language recognized by $\mathbf{A}_i = (Q_i, \Sigma_i, \eta_i, q_{0,i}, Q_{m,i})$ such that for all $i \in I$ we have

$$\forall q, q' \in Q_i,\ \forall \sigma \in \Sigma_i \setminus \Sigma_{o,i}.\quad q' = \eta_i(q, \sigma) \implies q' = q, \tag{5.11}$$

i.e. all unobservable transitions to agent i are selfloops. Let $A = A_1 \| \cdots \| A_n \subseteq \Sigma^*$ be the synchronous product of A_i's recognized by $\mathbf{A} = \mathbf{A}_1 \| \cdots \| \mathbf{A}_n$. There exists a finite ALM yielding IUF's for (almost) any such A. ■

Example 5.9 Figure 5.5-a shows two recognizers $\mathbf{A}_1$ and $\mathbf{A}_2$ where $\Sigma_1 = \Sigma_{o,1} = \{\alpha, \alpha_1, \beta_1\}$, $\Sigma_2 = \{\alpha, \alpha_2, \beta_2\}$, and $\Sigma_{o,2} = \{\alpha, \alpha_2\}$ and all states are assumed to be marked. Clearly (5.11) is satisfied. To arrive at recognizers $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ in part (b) of the same figure, we notice that while the common event α makes a selfloop transition at state 0 of $\mathbf{A}_1$, it also makes a state change at state 0 of $\mathbf{A}_2$. Therefore, by the modification (✠) in the proof, state 0 is unfolded yielding state 2 of $\tilde{\mathbf{A}}_1$. Labeling the states of recognizers $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$, we would have the corresponding updating functions $\mathcal{A}_1$ and $\mathcal{A}_2$ as in Table 5.14.

Recognizer $\mathbf{A}$ is then computed as the meet of $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ as can be

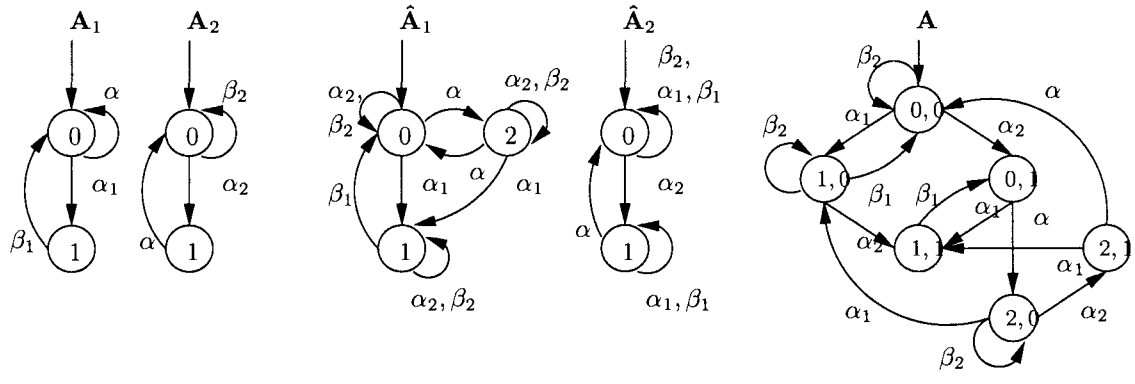


Figure 5.5: (a) Recognizers $\mathbf{A}_1$ and $\mathbf{A}_2$. (b) The modified recognizers $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ whose states are labeled. (c) The meet of the modified recognizers whose states are labeled by the joint labeling of the ALMs for $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$.

seen in part (c) where the tuple formed by the two ALMs for $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ is used as the ALM for $\mathbf{A}$ for which the updating functions in Table 5.14 are independent. ◊

Next we show that Lemma 5.22 can be applied to the case where a language is trace-closed.

Lemma 5.23 For a trace-closed language $A \in \Sigma^*$ there exists a state representation for which there exists an ALM yielding IUFs.

Proof: Following the fact that A is trace-closed, for each string in A all members of the equivalence class of that string modulo $\equiv_I$ belong to A, i.e. $A = \bigcup_{i \in I} P_i^{-1} P_i(A)$ and $A = A_1 \| \cdots \| A_n$. Then each $P_i(A)$ can be recognized by $\mathbf{A}_i = (Q_i, \Sigma_i, \eta_i, q_{0,i}, Q_{m,i})$ satisfying the condition of Lemma 5.22, by which the required ALM is computed. ■

An intuitive application of the above result is where a (supervisor's) language, $L_m(\mathbf{S})$, is derived as a the synchronous product of a number of component languages, $L_m(\mathbf{S}_i)$, where events in $\Sigma_{uo,i}$ appear as selfloops in $\mathbf{S}_i$.

Another application is the case where the plant's behavior $L(\mathbf{G})$ is trace-closed and $\mathbf{S}$ is JO with respect to $L_m(\mathbf{G})$.

Corollary 5.3 If $L_m(\mathbf{G})$ is trace-closed and $L_m(\mathbf{S})$ is JO with respect to $L_m(\mathbf{G})$, there exists an ALM for $\mathbf{S}$ yielding IUFs.

Proof: By Lemma 5.20, $\mathbf{S}$ is itself trace-closed and then Lemma 5.23 guarantees the existence of the required ALM. ■

In light of Lemma 5.22, every substructure derived from such a recognizer $\mathbf{A}$, as described in Lemma 5.22, may be assigned the same ALM. This point is clarified next.

Definition 5.27 Let $\mathbf{B} = (Q, \Sigma, \xi, q_0, Q_m)$ be a recognizer. A recognizer $\hat{\mathbf{B}} = (\hat{Q}, \Sigma, \hat{\xi}, q_0, \hat{Q}_m)$ is called a *subrecognizer* of $\mathbf{B}$ if $\hat{Q} \subseteq Q$, $\hat{Q}_m = \hat{Q} \cap Q_m$, and

$$\forall \sigma \in \Sigma^*,\ \forall q, q' \in \hat{Q}.\ \ q' = \hat{\xi}(q, \sigma) \implies q' = \xi(q, \sigma). \qquad \square$$

Corollary 5.4 Let $\mathbf{A} = (Q, \Sigma, \xi, q_0, Q_m)$ be a recognizer for which there exists an ALM ℓ yielding IUFs and $\hat{\mathbf{A}} = (\hat{Q}, \Sigma, \hat{\xi}, q_0, \hat{Q}_m)$ be a subrecognizer of $\mathbf{A}$. Then ℓ, restricted to the states of $\hat{\mathbf{A}}$, is an ALM for $\hat{\mathbf{A}}$ yielding IUFs, too.

Proof: For every two states $q, q' \in Q$ and each string $s \in \Sigma^*$, with $q' = \xi(q, s) \in Q$, and every label $\mathbf{v} \in \ell(q)$ assigned by the ALM, and the updating function $\mathcal{A}$ it holds that

$$\begin{aligned} q' = \hat{\xi}(q, s) &\implies q' = \xi(q, s) && \text{[Defn. 5.27]} \\ &\iff \mathcal{A}(s, \mathbf{v}) \in \ell(q). && \text{[Lemma 4.3]} \end{aligned}$$

Thus the same updating function applies to $\hat{\mathbf{A}}$. In particular, this holds for the case of the independent updating functions, too. ■

5.4.5 Construction of IUF-yielding ALMs

The applicability of the results such as Lemma 5.22 is limited to cases where a centralized supervisor can be obtained as the synchronous product of n automata, each defined on an observable subalphabet $\Sigma_{o,i}$. Apart from such cases, in general finding IUFs for every supervisor and all its events may be too much to expect, both because this may not be possible for a given state representation and since no constructive procedure for obtaining IUFs is known.

Accordingly, one approach to tackle the problem of computing IUFs is to narrow the study to find out sufficient conditions and constructive ALM-building procedures under which the updating function, associated with an observable event of a supervisor, becomes independent of other supervisors' private information. On top of being a reasonable step to begin with, making even a single updating function independent of external variables is important in its own, say when the supervisor's communication or computation load asks for such reduction. In the following, we follow this idea by first reducing the problem of finding ALMs to a problem in terms of Latin squares, and then employing this characterization to propose constructive procedures to obtain IUFs for some particular observable events of a supervisor.

Latin squares and ALMs

By Definition 5.22, to arrive at an IUF for an event $\alpha \in \Sigma_{o,i}$, the ALM-assigned labels should be such that no v_i is mapped, via transitions labeled by α, to more than one v_i', where $v_i \neq v_i'$. In Theorem 3.2 of Chapter 3 we used the structure of a Latin square to prove the existence of an ALM for a given centralized supervisor. It turns out that this structure might be useful in finding ALMs yielding IUFs, too. In what follows, we employ Latin squares for this purpose.

Definition 5.28 [78] Fix a set Γ of m symbols. A *Latin square of side m* is an $m \times m$ array in which each cell contains a single element from Γ, such that each element occurs exactly once in each row and exactly once in each column. □

Figure 5.6 shows two Latin squares of side 5, with horizontal and vertical coordinates numbered 0 to 4, and with symbols taken from $\Gamma = \{r_0, r_1, r_2, r_3, r_4\}$ (for now ignore the arrows). We refer to the bottom row and the leftmost column as the first row and first column, respectively. Observe that a Latin square is robust with respect to changing the order its rows and columns.

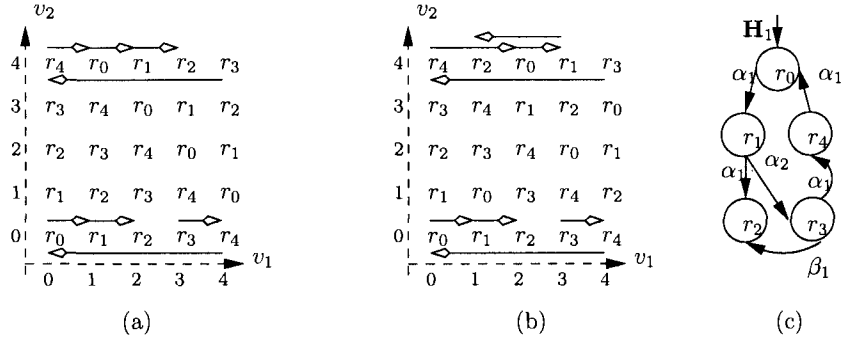


Figure 5.6: (a), (b) Two Latin squares of side $m = 5$ with $\Gamma = \{r_0, r_1, r_2, r_3, r_4\}$. v_1 and v_2 denote the horizontal and vertical components of a vector v assigned by an ALM to a state r_i, $(i = 0, 1, \cdots, m-1)$, respectively. (c) An example of an automaton whose states can be labeled by the Latin squares in parts (a) and (b) ($\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$).

The fact that there is a copy of every element of Γ in each row and in each column makes this structure appealing for building ALMs. To explain the idea, let[8] $|I| = 2$. If $\Gamma = R$, i.e. symbols are essentially the states of the centralized supervisor, each state has "exactly" one target state at which it can arrive using a transition labeled with an event from $\Sigma_{o,1}$ (respectively,

[8]Latin hypercubes are used in cases that $|I| > 2$ [50].

$\Sigma_{o,2}$) along the horizontal (respectively, vertical) axis. In other words, for every transition from a source state r_i to a target state r_j, $(j \neq i)$[9] labeled by an event $\alpha \in \Sigma_{o,1}$ (respectively, $\beta \in \Sigma_{o,2}$), there exist one copy of r_i and one copy of r_j in each row (respectively, column) of the Latin square. Following the definition of an ALM in Definition 4.6 and its associated updating functions (see Lemma 4.2), state r_i is assigned the set of integer vector labels whose positions in the Latin square are denoted by r_i.

Part (c) of Fig. 5.6 shows an automaton $\mathbf{H}_1$ with state set $Q = \{r_0, r_1, r_2, r_3, r_4\}$. Any of the Latin squares in parts (a) and (b) (and in fact any Latin square of side 5) may be used to define an ALM for $\mathbf{H}_1$.

Latin squares and IUFs

Continuing our observation on Fig. 5.6, notice that out of the two squares in parts (a) and (b), only the former yields an IUF for α_1. To explain this point, let us denote a transition from r_j to r_k, $(j \neq k)$, by an arrow from r_j to r_k along $\mathbf{S}_i$'s associated axis; for example, a state change from r_0 to r_1 using $\alpha_1 \in \Sigma_{o,1}$ in part (c) is represented by a horizontal arrow from r_0 to r_1 in the first row of the Latin square in part (a). Arrows are shown for rows 1 and 5 of both squares. Observe that in part (a) arrows map all states in one column consistently to states in another column; e.g. for any row (say, 5), α_1 triggers a transition from a state in column 1 (r_4) to another state in column 2 (r_0) whenever such transition is defined. However, this is not the case in part (b); for example, whereas in row 1 the first column is mapped to the second column (corresponding to r_0-r_1 transition), in the fifth row the first column is mapped to the third column (corresponding to the r_4-r_0 transition). In other words, in part (a), regardless of which row they are in, the arrows induce a map

[9]To see how selfloops are treated see [50].

which assigns a column to another column, whereas in part (b) this assignment depends on the row number, too. As a result, the first Latin square defines an ALM yielding an IUF for α_1, whereas in part (b) the updating functions associated with α_1 depend on row numbers, i.e. $\mathbf{S}_2$'s labels.

Remark 5.4 According to the suggested arrangement for obtaining IUFs, it is quite possible that a state with no outgoing arrow is mapped to another state by the requirement that the mapping be done regardless of the row number (e.g. in Fig. 5.6-a, row 1, r_2 is mapped to r_3 following the fact that in other rows, the cell in column 3 is always mapped to the cell in column 4). However, adding such transitions does not contradict the transition function of the supervisor since their corresponding guard formulas would evaluate to false at the source states. □

The above observation may be formalized as follows.

Definition 5.29 Let $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ be a centralized supervisor and $i \in I$. For an event $\alpha \in \Sigma_{o,i}$, denote by $\eta_\alpha^{i,d}$ a z-tuple of states $(q_0, q_1, \cdots, q_{z-1})$, where $z \in \mathbb{N}$, and

1) $\hat{Q} = \{q_0, q_1, \cdots, q_{z-1}\} \subseteq R$,

2) $\forall j, k \in \{0, \cdots, z-1\}, (j \neq k).\ q_j \neq q_k$,

3) $\forall j \in \{0, \cdots, z-2\},\ q_{j+1} = \xi(q_j, \alpha)$, and

4) $\forall q \in R.\ \ q_0 = \xi(q, \alpha) \implies q = q_{z-1}$.

Index $d \in \mathbb{N}$ is to number a tuple when there is more than one such tuple associated with α. Also let $\Delta_\alpha^i = \{\eta_\alpha^{i,d} \mid \exists m \in \mathbb{N}.\ d \in \{1, \cdots, m\}\}$. □

Observe that by items 3 and 4 of the above definition and the fact that $\mathbf{S}$ is a deterministic recognizer, an α-labeled transition from a state $q_j \in \hat{Q}$

would arrive at a state in the same set. In other words, there is no outgoing α-labeled transition from the states in $\eta_\alpha^{i,d}$ and in this sense it is a *maximal* tuple. However, there may exist α-labeled transitions leaving some states in $R \setminus \hat{Q}$ and entering a state in $\{q_1, \cdots, q_{z-1}\}$.

Definition 5.30 Let $\mathbf{S} = (R, \Sigma, \xi, r_0, R_m)$ be a centralized supervisor, $\hat{Q} = \{q_0, q_1, \cdots, q_{z-1}\}$, and $\eta_\alpha^{i,d} = (q_0, q_1, \cdots, q_{z-1})$ be defined as in Definition 5.29. Call such an $\eta_\alpha^{i,d}$ (or its corresponding tuple) *isolated* if the following holds.

$$\forall q \in R,\ q' \in \hat{Q}.\ \ q' = \xi(q, \alpha) \implies q \in \hat{Q}$$ □

As an example, for automaton $\mathbf{H}_1$ in Fig. 5.6-c we have $\eta_{\alpha_1}^{1,1} = (r_3, r_4, r_0, r_1, r_2)$, $\eta_{\beta_1}^{1,1} = (r_3, r_2)$, and $\eta_{\alpha_2}^{2,1} = (r_1, r_3)$. Notice that all these tuples are isolated.

The following result provides a constructive procedure to obtain an IUF for $\mathcal{A}_i(\alpha, .)$ when $|\Delta_\alpha^i| = 1$.

Proposition 5.8 Let $i \in I, \alpha \in \Sigma_{o,i}$. If $\Delta_\alpha^i = \{\eta_\alpha^{i,1}\}$ and $\eta_\alpha^{i,1}$ is isolated, there exists an ALM which yields an IUF for $\mathcal{A}_i(\alpha, .)$. ■

Remark 5.5 Regarding the above result, the following two points are in order. First, in Fig. B.1, left-shifting of the first row to construct the square, can be replaced with right shifting, too. Second, to meet the requirement of $\mathbf{0} \in \ell(r_0)$ in Definition 4.6, two rows or two columns in the final rectangle may be required to change their positions with each other. □

A closer observation of the proof of Proposition 5.8 reveals that as long as we are dealing with disjoint subsets of R, the result still holds.

Corollary 5.5 Let $i \in I, \alpha, \beta \in \Sigma_{o,i}$, $|\Delta_\alpha^i| = J_\alpha$, and $|\Delta_\beta^i| = J_\beta$. For every $d \in \{1, \cdots, J_\alpha\}$ and each $d' \in \{1, \cdots, J_\beta\}$, let

1) $\eta_\alpha^{i,d} \in \Delta_\alpha^i$ be isolated, and

2) $\eta_\beta^{i,d'} \in \Delta_\beta^i$ be isolated, and

3) $\eta_\alpha^{i,d}$ and $\eta_\beta^{i,d'}$ be mutually disjoint.

There exists an ALM which yields IUFs for $\mathcal{A}_i(\alpha, .)$ and $\mathcal{A}_i(\beta, .)$. ■

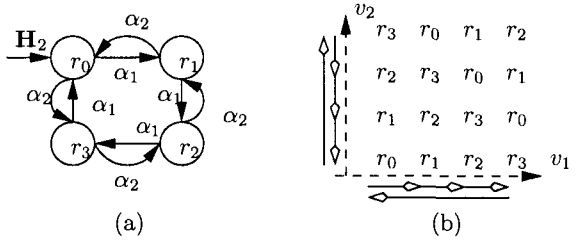


Figure 5.7: (a) An automaton with $\Sigma_{o,1} = \{\alpha_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) A Latin square which defines an ALM yielding IUFs for α_1 and α_2.

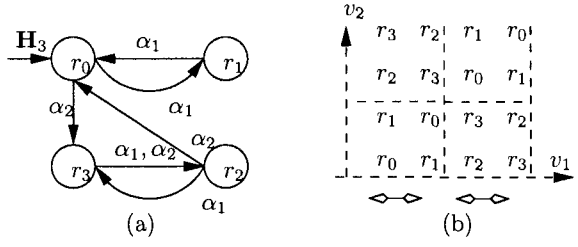


Figure 5.8: (a) An automaton with $\Sigma_{o,1} = \{\alpha_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) A Latin square which defines an ALM yielding IUF for α_1, but not for α_2.

Example 5.10 As a first simple application of Proposition 5.8, Figure 5.7-a illustrates an automaton $\mathbf{H}_2$. For this automaton we have $\eta_{\alpha_1}^{1,1} = \eta_{\alpha_2}^{2,1} = (r_0, r_1, r_2, r_3)$. These two tuples are the only ones for their corresponding events and both are isolated. Therefore, we may employ a Latin square of side 4 whose rows and columns are arranged as stated in the proof of the lemma and shown in part (b).

Figure 5.8-a shows another example for which we have $\eta_{\alpha_1}^{1,1} = (r_0, r_1)$, $\eta_{\alpha_1}^{1,2} = (r_2, r_3)$, and $\eta_{\alpha_2}^{2,1} = (r_0, r_2, r_3)$, where all tuples are isolated. If we are after an IUF for just α_1, a simple solution is to put side by side two Latin squares of side 2 with symbols from $\{r_0, r_1\}$ and $\{r_2, r_3\}$, respectively and then copy this structure to build a side 4 Latin square as in part (b). Notice that such an arrangement would not result in IUF for α_2. $\Diamond$

Finding IUFs for non-isolated tuples

Proposition 5.8 and Corollary 5.5 consider finding IUFs for the case where the subsets of R which correspond to $\eta_{\cdot}^{i,\cdot}$ are disjoint. If this is not the case, then the Latin rectangle which is desired should "embed" in it Latin squares which have some symbols in common, as formally defined next.

Problem 5.3 Given a Latin square of side m_1 with symbols taken from the set Γ_1 and another Latin square of side m_2 with symbols taken from the set Γ_2, find a Latin square (if it exists at all) which "embeds" copies of both squares such that all appearances of any $\gamma \in \Gamma_1 \cap \Gamma_2$ falls in a copy of each square? $\square$

Unfortunately, there are no existence conditions, nor a general algorithm available for finding solutions to the above problem [86]. However, there are cases for which a solution exists as demonstrated by the following example.

Example 5.11 For the automaton in Fig. 5.9-a we have $\eta_{\alpha_1}^{1,1} = (r_0, r_1)$, $\eta_{\beta_1}^{1,1} = (r_0, r_2, r_4)$, and $\eta_{\alpha_2}^{2,1} = (r_0, r_1, r_2, r_4)$ where all tuples are isolated. Let us assume that we are interested in finding IUFs for α_1 and β_1. The two Latin squares in parts (b) and (c) define ALMs which yield IUFs for events α_1 and α_2. Notice that state r_0 is a common symbol in both squares. The Latin square in part (d) has side 6 and embeds both squares in parts (b) and (c), where copies of

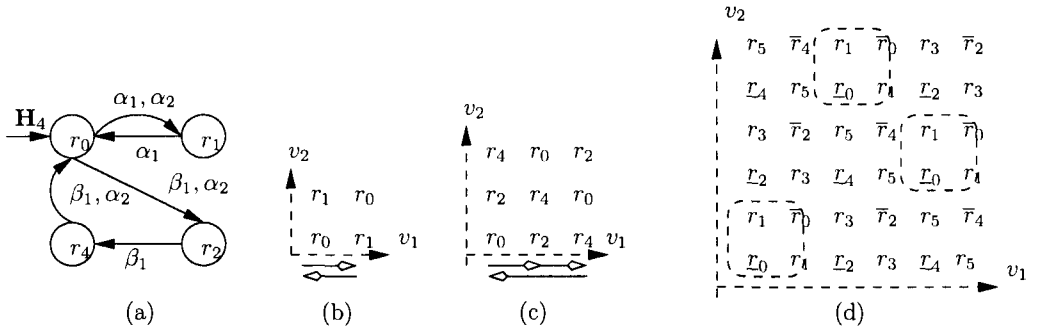


Figure 5.9: (a) An automaton with $\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b), (c) Two Latin squares which defines ALMs yielding IUFs for α_1, and β_1, but not for α_2. (d) A Latin square which embeds the Latin squares in parts (b) and (c) and satisfies the conditions in Problem 5.3. copies of the square corresponding to $\eta_{\alpha_1}^{1,1}$ are enclosed with dashed lines, and symbols of the two copies corresponding to $\eta_{\beta_1}^{1,1}$ are marked with underlines and overlines.

the square corresponding to $\eta_{\alpha_1}^{1,1}$ are enclosed with dashed lines, and symbols of the two copies corresponding to $\eta_{\beta_1}^{1,1}$ are marked with underlines and overlines. Moreover, it can be observed that whenever r_0 appears in some place, it participates simultaneously in building two copies of Latin squares corresponding to $\eta_{\alpha_1}^{1,1}$ and $\eta_{\beta_1}^{1,1}$. As can be seen, there are other fictitious states, namely, r_3 and r_5, which do not exist in $\mathbf{H}_4$ and merely participate in building the larger square. Such states may be employed later to simplify guard formulas. $\Diamond$

The last example illustrates a case where IUFs for all $\mathbf{S}_1$'s events can be found.

Example 5.12 The automaton $\mathbf{H}_5$ has three states with $\eta_{\alpha_1}^{1,1} = (r_0, r_2)$, $\eta_{\beta_1}^{1,1} = (r_0, r_1, r_2)$, and $\eta_{\alpha_2}^{2,1} = (r_0, r_1)$ where all tuples are isolated. We show that an ALM can be found for events $\alpha_1, \beta_1 \in \Sigma_{o,1}$. We start by building a Latin square which yields IUF for β_1. This square can be seen in the left side of part (b). To represent α_1-labeled transitions, we associate to each column of this

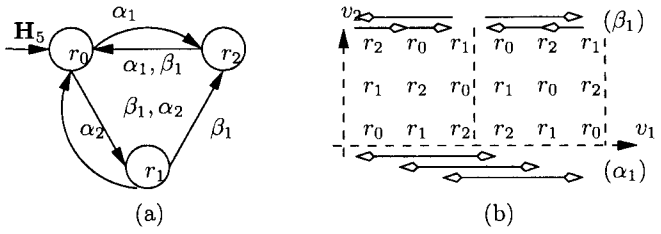


Figure 5.10: (a) An automaton with $\Sigma_{o,1} = \{\alpha_1, \beta_1\}$, $\Sigma_{o,2} = \{\alpha_2\}$. (b) The columns of the left Latin square, which defines an ALM yielding IUF for β_1, are suitably mapped to columns of the second square, also yielding IUF for β_1, to define the part of the ALM which yields IUF for α_1. The top and bottom arrows represent β_1- and α_1-related transitions, respectively.

square a column in a new square, which is put beside it. For example, since α_1 moves states r_0 and r_2 to r_2 and r_0, respectively, each column in the first square is associated with a column in the second square in which r_0 and r_2 are interchanged. In other words, we choose a Latin square of the same side and symbols but with a different arrangement such that α_1-related transitions can be represented by mappings between their columns. $\Diamond$

5.5 Conclusion

This chapter studies the communication among supervisors in DSDES framework. The study begins by defining "information" and "routing" events, as the two types of communication-related events, and also introducing the notions of information and routing policies. This is followed by deriving six information policies for transferring state- or event-related information among supervisors, with some prescribing less communication. Next, a communication-based partitioning of DSDESs is introduced and some properties of two such special classes are derived. The chapter ends with some results on communication-based state representation of centralized supervisors and introducing some

classes of ALMs which enjoy communication-oriented properties.

Chapter 6

Case Study

6.1 Introduction

"A *communication protocol* consists of a set of rules which govern the orderly exchange of messages among the system components in order to provide a specified set of services to service users located at different access points" [67]. The service users of the protocol are usually geographically separated from each other. From the protocol point of view, each service user is connected to the network through a *Protocol Entity* (PE), which has partial observation of the behavior of the communication network in which it participates. A protocol specification determines the functions which should be done by the whole network and their details, such as order, time, etc, in a cooperative and distributed manner [67].

A *reliable* protocol is one which meets all its associated specifications. Reliability, which is inevitably required for almost all commercial protocols, can be obtained through mathematical modeling of the network of PEs and systematic synthesis and analysis of protocols using standard mathematical tools. The discrete nature of a communication protocol allows us to model

it, its associated network of PEs, and related specifications as DESs. If such DESs can be behaviorally modeled as regular languages, or equivalently as finite automata, RW supervisory control theory can be employed to synthesize for them a *centralized* supervisor which guarantees the satisfaction of the specifications by the network of PEs. This centralized-supervisor observes every event which is observable to at least one PE.

In the next step, DSDES framework can be employed to explore the dynamical structure of the centralized supervisor in a distributed way and represent it as a PDS. This distributed representation, which assumes limited observational window for each PE, serves to systematically derive informational dependencies of each PE and design communication events which provides it with the required information, owned by other PEs. The set of communication events thus computed, forms the logical part of a communication protocol, i.e. the part which results from logical modeling of the network and its specification. This part together with the physical part, which addresses continuous dynamics of the network as represented by differential equations, form the communication protocol. This classification of a protocol into logical and physical parts is based on a general view of protocols as "hybrid systems," having discrete-event and continuous dynamics. Whereas, it is beyond the scope of this work to formalize this classification and verify the correctness of that, we take this as a conservative vantage point into the structure of a protocol and limit this work to the study to the logical part, only. Furthermore, we notice that the logical part should be itself expressible by a number of finite automata, otherwise RW theory cannot be used as stated above.

A communication protocol may provide service to one or more network layers in the sense of *Open Systems Interconnection* (OSI) reference model [87]. In Chapter 3 we discussed *Alternating Bit Protocol* (ABP), which is used in

"physical" and "data link" layers. There, using the proposed machinery in EFSM framework, the protocol was synthesized by modeling the network of a sender and a receiver, and providing the required information for each entity using communication events, which transfer one extra bit attached to each data frame. Following that successful experience, this chapter studies another protocol, which is known as Reliable Multicast Transport Protocol (RMTP).

This chapter continues with an overview of RMTP in Section 6.2. For a simple network, the model of the component plants and the specifications associated with RMTP are introduced in Section 6.3. The synthesis procedure of two RMTP-like communication policies for the network is elaborated in Section 6.4.

6.2 An overview of RMTP

The concept of "multicast communication" means simultaneous delivery of messages to a group of destinations such that a message passes over a link only once [88]. Whereas multicasting has applications in data link layer, it is mainly considered as "IP multicast" and is used for "streaming media" and "internet television." In this sense, multicasting is done at the routing level [88]. RMTP is a transport protocol for IP multicast which provides reliable transport by employing the tree topology of message acknowledgement and local recovery from losses [2]. Its revised form, RMTP-II, adds more real-time features to increase its efficiency and address time issues. To review RMTP, we assume a tree topology for the network and describe the terminology and explain the functions of RMTP.

6.2.1 Terminology

We start by introducing the terminology used to describe RMTP within the scope of this work. The following should be considered as informal definitions.

- Control signal: Any message other than "data," which is exchanged in the network is called a control signal. It serves control objectives such as reliability and guaranteed functionality of the network.

- Control channel: It is a communication channel which is used for transfer of control signals.

- Unicast channel: It is a data or control channel which is defined between two specific PEs.

- Multicast channel: It is a collection of unicast channels associated with the children of a single parent.

- Sender: It is a PE which wants to send data to other PEs in a network.

- Receiver: It is a PE which is the end target of the sender's data.

- Acknowledgement (ACK): It is a control signal which is generated by "receivers" of the sender's data or a parent node to inform their upper nodes of the receipt and non-receipt of previously sent data by a sender.

- Tree-based acknowledgement (TRACK): It is a type of acknowledgement whose issuance follows the bottom-up network topology of a tree.

- Data channel: It is a communication channel which is used for data transfer.

- Interior node: Every node between a TN and a receiver in the tree structure is called an interior node.

- Aggregator node: It is an interior node which collects acknowledgements from its children and sends an appropriate acknowledgement to its parent.

- Designated Receiver: It is an interior node which has the functionality of an aggregator node. Furthermore, it receives the data sent by the sender so that if its children require the data it sends it to them.

- Control node: This is another name for every PE, i.e. each node, in the RMTP's associated network.

- Top node (TN): It is a control node which is the highest-located node in the tree-hierarchy of control nodes.

- Stable packet: A data packet which has been received by all receivers is called by the sender a stable packet.

- Transmission window: It is a finite-length frame consisting of ordered pieces of data to be sent sequentially.

6.2.2 Basic functions of RMTP

This section describes the basic functionality of RMTP and RMTP-II protocols, following [89] and [2].

A simple RMTP's associated network consists of one sender node (SD), many receiver nodes (RNs), a Top Node (TN), and zero or more Designated Receivers (DRs) and Aggregator Nodes (ANs). These nodes are organized into a tree with TN (and SD) nodes being its root (or top), RNs nodes being its leaves, and all other nodes being in between. Figure 6.1 illustrates an RMTP tree with multiple control nodes. The sender and the TN are on one host and receivers are on multiple other hosts connected to the network.

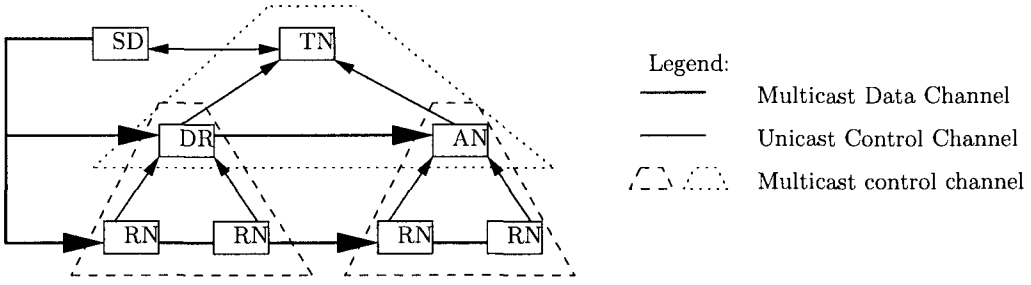


Figure 6.1: An RMTP tree (reprinted from [2]). Arrows specify the direction of data and control signals flow.

There are mechanisms for joining, staying with, or leaving this network, which should be performed by all above-mentioned nodes[1] [89]. These mechanisms are beyond the scope of this work and are not considered here. Also, in an implementation different types of nodes may run in a single process, say a host can be a sender and a TN, a receiver and a DR, etc. These cases are not studied here, either. Rather, we focus on the basic operation of the network which is sending a data packet from sender to all receivers in a reliable manner. Accordingly, one such above-mentioned network should be considered to be fixed during the transfer of the data packet. Once this is done, the network can change itself to send other data packets. In the following, we limit our discussion to one fixed network.

RMTP performs its communication using *multicast data channels* and *unicast* and *multicast control channels*. Initially data is sent on a *multicast data channel* by the *sender*. Data may be *retransmitted* by the *sender* on the *multicast data channel*, or "*multicast* or *unicast* on a *local control channel* by a DR" [2]. The tree topology has a one to one relationship with the way the control channels are organized. If the network is symmetrical, i.e. control

[1]These include Heartbeat, HearbeatResponse, NullData, Negative ACK (NACK), Forward Error Correction (FEC), etc.

tree topology is relatively congruent to the multicast routing tree topology, the data and control channels may share the same path. If the network is asymmetrical, "such as a one-way satellite network with a terrestrial return path" [2], data will be multicast to RNs and DRs directly, but TNs and ANs will only receive control traffic.

The basic operation of RMTP over the network may be explained as follows. "A receiver *periodically* informs its parent about the packets it has or has not received by unicasting a Tree-based ACK (TRACK) packet to the parent. Each parent node aggregates the TRACKs from its child nodes and unicasts a single aggregated TRACK to its parent. The TN aggregates the TRACKs from its child nodes and unicasts a single TRACK to SN. Each DR or TN has a local control channel, which is a multicast group that connects it to all of its children. This is used for *local multicast retransmission* of lost packets to just the parent's children. A tree forms a loop from the sender to the receivers, through the control tree, and back to the sender. Data regularly exercise the data channel. TRACKs regularly exercise the control channel in the upward direction. This combination constantly checks that all of the nodes in the tree are still functioning correctly" [2]. The main objective of the tree topology is to reduce the control traffic, resulting from receiver's acknowledgements of the data receipt, by processing these acknowledgements locally in parent nodes.

6.3 Modeling of RMTP's basic functions for a simple network

Following the overview in Section 6.2, this section considers an asymmetrical network and a reliability specification for it, and derives an RMTP-like set

of communication rules which guarantees reliable data transmission. In this study, we assume a fixed network and neglect node mechanisms for joining, staying, and leaving the network as well as time issues. As a result, the model is limited to these assumptions and the resulting communication rules should be considered a guideline to the basic operation of the RMTP. It should be noticed that other than expressibility by regular languages, which is currently a fundamental limitation of RW supervisory control theory, other assumptions are for the sake of this primary study and computational purposes. Therefore, once this first study is done, the model can be extended, within the limits of expressibility by automata theory, by releasing assumptions and deriving parametric solutions to the general case of the problem. Nevertheless, this primary study illustrates the applicability of the proposed approach of this work and is insightful toward a thorough investigation of RMTP, too.

6.3.1 Network topology and control nodes

Let **N** be an asymmetrical network consisting of sender **SD**, top node **TN**, designated receivers $\mathbf{DR}_1$ and $\mathbf{DR}_2$, and receivers $\mathbf{RN}_1$, $\mathbf{RN}_2$, and $\mathbf{RN}_3$, all organized in a tree topology shown in Figure 6.2. Observe that **TN** has two children, $\mathbf{DR}_1$ and $\mathbf{DR}_2$, which are designated receivers. Also $\mathbf{DR}_1$ and $\mathbf{DR}_2$ have two and one children, respectively.

The network is supposed to function as follows.

1. **SD** fetches a data packet and multicasts it over the multicast data channel. It is intended that $\mathbf{RN}_1$, $\mathbf{RN}_2$, and $\mathbf{RN}_3$ receive the data packet. $\mathbf{DR}_1$ and $\mathbf{DR}_2$ may receive it, too though this is not required unless the packet is required to retransmit to their children.

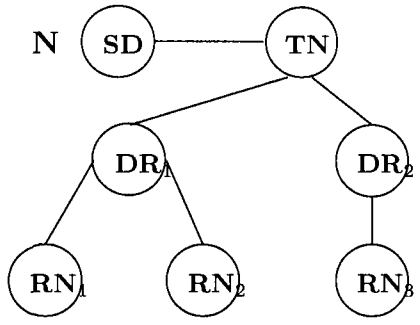


Figure 6.2: An asymmetrical network consisting of 7 control nodes.

2. Upon the successful receipt of the data packet or after a certain fixed amount of time, during which receivers wait to receive the data packet, each receiver issues an acknowledgement to inform its parent of either successful receipt of the data packet or receipt failure.

3. Once a designated receiver receives a receipt-indicating acknowledgement from all its children, it issues a successful-indicating acknowledgement to its parent, TN. On the other hand, if a designated receiver receives a failure-indicating acknowledgement from one of its children or if a time-out indicates that no acknowledgement has arrived within a specific time interval, one of the two cases happens. If the designated receiver has the data packet itself, it retransmits the data packet to the failure-indicating children and waits for its acknowledgement within a specific time limit. If the designated receiver itself does not have the data packet, it issues a failure-indicating acknowledgment to its parent, **TN**.

4. Once **TN** receives success-indicating acknowledgements from $\mathbf{DR}_1$ and $\mathbf{DR}_2$, it issues a success-indicating acknowledgement to **SD**. Otherwise, if it receives a failure-indicating acknowledgement from either child, it issues a failure-indicating acknowledgement to **SD**.

5. If **SD** receives a success-indicating acknowledgement, it proceeds to fetch a new data packet and sends it, otherwise, it retransmits the previous data packet on multicast data channel.

Assumption 6.1 It is assumed, as a fairness condition, that eventually the sender will receive a success-indicating acknowledgement either after it sends the data packet, or after some retransmissions by designated receivers or the sender itself. □

Based on this description, in the following subsections, first the events which are used to model the network are introduced and then each control node is modeled accordingly.

Assumption 6.2 Since the communication rules which are studied here pertain to the transport layer of OSI model [87], it is reasonable to assume that the protocols used in lower layers, such as network or data link layers, equip each control node with means to distinguish if the piece of information it receives (i.e. data or acknowledgements) is erroneous or not. □

6.3.2 Definition of events

A major step in modeling a DES is to define events in such a way that they unambiguously describe the part of the behavior of the DES in question, while keeping the model simple enough for analysis and synthesis purposes. Here, we intend to model that part of the network's behavior which is relevant to RMTP modeling and synthesis. Accordingly, only those events of each control node are defined which serve this modeling. In other words, we avoid defining events which "internally" implement the functions of the control node unless they also serve to implement the RMTP-related behavior. For every event, first its definition, based on the description in Subsection 6.3.1, is given and

its symbol is introduced. Then the control nodes which can exercise control over it or observe it are identified.

1. "Data Fetch": This event is to represent **SD**'s fetching a data packet and is denoted by f_{SD}. It is controllable and observable by **SD**.

2. "Send Data Packet": A data packet may be sent for the first time or retransmitted by **SD**, or retransmitted by a designated receiver, say $\mathbf{DR}_j$ ($j \in \{1,2\}$). In the first case, the event is denoted by S_{SD} and in the second case, the event is denoted by S_{DRj}. These events are controllable and observable by their issuers, only.

3. "Data Packet Received Error-free": This type of event corresponds to the successful receipt of a data packet[2] by a receiver or a designated receiver. Assumption 6.2 implies that this type of event is well-defined for the control node which generates it in the sense that every such event follows a successful data receipt within its specified time limit. Since data packets are supposed to be received by receivers and designated receivers, this event, denoted by R, is owned by every such node, and a subscript would determine to which control node it belongs. Therefore, R_{RNi} and R_{DRj} belong to $\mathbf{RN}_i$ ($i \in \{1,2,3\}$) and $\mathbf{DR}_j$ ($j \in \{1,2\}$), respectively. Each event is observable by its owner, but it is uncontrollable because its issuance follows the error-free receipt of the data packet, automatically.

4. "Data Packet Received Erroneous": This event corresponds to the receipt of erroneous data, which is by Assumption 6.2 a well-defined event. This type of event is denoted as R^e with a subscript which determines to which control node it belongs. Similar to event R, R^e is used by

[2]Note that the data could be sent by **SD** or the designated receiver who is the parent of the control node in question.

receivers and designated receivers. Therefore, R^e_{RNi} ($i \in \{1,2,3\}$) and R^e_{DRj} ($j \in \{1,2\}$) belong to $\mathbf{RN}_i$ and $\mathbf{DR}_j$, respectively. Each event is observable by its owner and uncontrollable .

5. "Success-indicating Acknowledgement": This type of acknowledgement is denoted by A. Three types of control nodes generate this acknowledgement; the receivers, the designated receivers, and the top node. Every receiver issues an A event upon the successful receipt of a data packet within a specified time limit. This data packet could be sent by **SD** or the designated receiver who is the receiver's parent. For receiver $\mathbf{RN}_i$ ($i \in \{1,2,3\}$), this acknowledgement is denoted by A_{RNi}. Also, every designated receiver $\mathbf{DR}_j$ ($j \in \{1,2\}$), issues an A event, denoted by A_{DRj}, upon receiving event A from all its children. Top node **TN** generates an A event, denoted by A_{TN}, when it receives success-indicating acknowledgements from both its children. Every A event is controllable solely by its issuer and observable by both its issuer and its parent (considering **SD** as **TN**'s parent for now).

6. "Failure-indicating Acknowledgement": This type of acknowledgement is denoted by A^e and is generated by the receivers, the designated receivers, and the top node. Every receiver issues an A^e upon the receipt of an erroneous data packet. This data packet could be sent by **SD** or the designated receiver, who is the receiver's parent. For receiver $\mathbf{RN}_i$ ($i \in \{1,2,3\}$), this acknowledgement is denoted by A^e_{RNi}. Also, every designated receiver $\mathbf{DR}_j$ ($j \in \{1,2\}$), issues an A^e event, denoted by A^e_{DRj}, a) upon receiving either an A^e event or an erroneous A or A^e from (at least) one of its children and b) it does not have the data packet itself (to retransmit it to its children). Top node **TN** issues an A^e event, denoted by A^e_{TN}, upon receiving an A^e or an erroneous A or A^e from

(at least) one of its children. Every A^e event is controllable solely by its issuer and observable by both its issuer and its parent (considering **SD** as **TN**'s parent for now).

7. "Acknowledgement Received": When an acknowledgement, of type A or A^e, is received by a control node, which is of course the parent of the issuer of the acknowledgement, it is indicated by RA or RA^e respectively. Furthermore, RA^e also represents the *erroneous* receipt of A and A^e acknowledgements, because an erroneous acknowledgement has the same effect as a failure-indicating acknowledgement , i.e. the receiver of the acknowledgement has received no success-indicating acknowledgement. Each such "acknowledgement received" event borrows the subscript of the acknowledgement it receives; for example, RA_{TN} indicates that **SD**, which is **TN**'s parent, has received A_{TN}. RA and RA^e are observable by their issuers. Since there are two types of acknowledgements, the control node which receives an acknowledgement can choose to generate either an RA or an RA^e. As a result, we assume that RA and RA^e are controllable by their issuers. Notice that this is different from the case of R and R^e events, which represent the receiving of data packets. Unlike an acknowledgement, a data packet is sent in one form, i.e. error-free, but may receive in two forms, i.e. error-free or erroneous, giving rise to R and R^e, respectively.

8. "Time-out": Each time a piece of information, i.e. a data packet or an acknowledgement, is to be received by a control node, the control node waits for this receipt for a limited time interval. This limited time is a parameter of the control node and results from physical constraints and other real-time considerations, which are beyond the scope of this work. Once the information does not arrive within the time limit, a

time-out signal is generated at the control node, which is observable, but uncontrollable by it. We assume that the occurrence of a time-out resets the timer generating it to 0, though this timer does not appear in the model. Event "time-out" may happen in the following situations.

(a) For $i \in \{1, 2, 3\}$, if receiver $\mathbf{RN}_i$ does not receive, within a specified time, the data message from the sender nor form its parent, i.e. the designated receiver above it, a time-out[3] is generated at $\mathbf{RN}_i$. This time-out, which is observable but uncontrollable by $\mathbf{RN}_i$, is denoted by T_{RNi}.

(b) For $j \in \{1, 2\}$, if designated receiver $\mathbf{DR}_j$ does not receive the acknowledgements sent from its children within a specified time, a time-out is generated at $\mathbf{DR}_j$, which is uncontrollable and observable by it and is denoted by T_{DRj}.

(c) If **TN** does not receive acknowledgements from its children within a specified time, a time-out is generated at **TN**, which is denoted by T_{TN} and is uncontrollable and observable by **TN**.

(d) If **SD** does not receive **TN**'s acknowledgement in a specified time, a time-out is generated at **SD**, which is denoted by T_{SD} and is uncontrollable and observable by it.

The above events are summarized in Table 6.1, where we use "DR" for "designated receiver" and "ack." for "acknowledgement." To employ the notation define in Chapter 4, we set $\mathbf{I} = \{\mathbf{SD}, \mathbf{TN}, \mathbf{DR}_1, \mathbf{DR}_2, \mathbf{RN}_1, \mathbf{RN}_2, \mathbf{RN}_3\}$ and $I = \{1, 2, 3, 4, 5, 6, 7\}$, and define a mapping $\mathbf{f} : \mathbf{I} \to I$ which assigns an index to each control node such that $\mathbf{f}(\mathbf{SD}) = 1$, $\mathbf{f}(\mathbf{TN}) = 2$, $\mathbf{f}(\mathbf{DR}_1) = 3$, $\mathbf{f}(\mathbf{DR}_2) = 4$, $\mathbf{f}(\mathbf{RN}_1) = 5$, $\mathbf{f}(\mathbf{RN}_2) = 6$, and $\mathbf{f}(\mathbf{RN}_3) = 7$. This mapping lets

[3]We assume that both cases have the same time limit, otherwise two time-out events should be defined.

us refer to the subalphabets associated with each control node by its index. Following Notation 4.2, for every $i \in I$, let $\Sigma_{c,i}$ and $\Sigma_{o,i}$ denote controllable and observable subalphabets of control node i and define $\Sigma_i = \Sigma_{c,i} \cup \Sigma_{o,i}$, $\Sigma_c = \bigcup_{i \in I} \Sigma_{c,i}$, $\Sigma_o = \bigcup_{i \in I} \Sigma_{o,i}$, and $\Sigma = \bigcup_{i \in I} \Sigma_i$. Correspondingly, the subalphabets of each control node are listed in Table 6.2.

6.3.3 Modeling the control nodes

The model of each control node $\mathbf{n} \in \mathbf{I}$ is obtained by identifying a number of states and state transitions labeled with the events in the node's subalphabet $\Sigma_{\mathbf{n}}$. Whereas the detailed behavioral description of the control node may require more events, states, and state transitions to employ, to keep the model simple for synthesis, it is preferred to avoid introducing events and states which are irrelevant to RMTP's description (see the explanations in Subsection 6.3.2). Accordingly, the introduction of subalphabets in Table 6.2 is to formalize RMTP *specifications* rather than deriving a detailed model of each control node. In the following, the models of the control nodes are explained.

Assumption 6.3 It is assumed that each control node has *local control means* to prohibit the occurrence of some events at some states. Whereas these "means" are not studied here, it is worth justifying this assumption. To this end, notice that if such an event is generated by the node as a reply to the receipt of some acknowledgement, this local control simply means that the node ignores the acknowledgement. Otherwise, i.e. if the event happens as a result of some "autonomous internal decision," the node simply avoids generating the event. Example of this case is event "fetch," generated by **SD**. □

Table 6.1: Summary of the events used to model the RMTP network. Here $i \in \{1,2,3\}$ and $j \in \{1,2\}$.

Event's name (σ)	$I_c(\sigma)$	$I_o(\sigma)$	Description
f	$\{\mathbf{SD}\}$	$\{\mathbf{SD}\}$	Fetch a data packet
S_{SD}	$\{\mathbf{SD}\}$	$\{\mathbf{SD}\}$	Send or retransmit a data packet (multicast)
T_{SD}	$\emptyset$	$\{\mathbf{SD}\}$	TN's ack. has not arrived on time
RA_{TN}	$\{\mathbf{SD}\}$	$\{\mathbf{SD}\}$	**SD** received A_{TN} from **TN** on time
RA^e_{TN}	$\{\mathbf{SD}\}$	$\{\mathbf{SD}\}$	**SD** received A^e_{TN} from **TN** on time
R_{RNi}	$\emptyset$	$\{\mathbf{RN}_i\}$	Error-free data received on time
R^e_{RNi}	$\emptyset$	$\{\mathbf{RN}_i\}$	Received erroneous data
A_{RNi}	$\{\mathbf{RN}_i\}$	$\{\mathbf{RN}_i\}$	Error-free data received on time
A^e_{RNi}	$\{\mathbf{RN}_i\}$	$\{\mathbf{RN}_i\}$	Data erroneous or not received on time
T_{RNi}	$\emptyset$	$\{\mathbf{RN}_i\}$	No data received on time
S_{DRj}	$\{\mathbf{DR}_j\}$	$\{\mathbf{DR}_j\}$	Retransmit a data packet (local multicast)
R_{DRj}	$\emptyset$	$\{\mathbf{DR}_j\}$	Error-free data received on time
R^e_{DRj}	$\emptyset$	$\{\mathbf{DR}_j\}$	Received erroneous data
A_{DRj}	$\{\mathbf{DR}_j\}$	$\{\mathbf{DR}_j\}$	A from all children received on time
A^e_{DRj}	$\{\mathbf{DR}_j\}$	$\{\mathbf{DR}_j\}$	An A^e or an erroneous ack. received from a child
T_{DRj}	$\emptyset$	$\{\mathbf{DR}_j\}$	An ack. from a child has not arrived on time
RA_{RNi}	$\{\mathbf{DR}_j\}$	$\{\mathbf{DR}_j\}$	$\mathbf{DR}_j$ received A_{RNi} from $\mathbf{RN}_i$ on time
RA^e_{RNi}	$\{\mathbf{DR}_j\}$	$\{\mathbf{DR}_j\}$	$\mathbf{DR}_j$ received A^e_{RNi} from $\mathbf{RN}_i$ on time
A_{TN}	$\{\mathbf{TN}\}$	$\{\mathbf{TN}\}$	A from all DRs received on time
A^e_{TN}	$\{\mathbf{TN}\}$	$\{\mathbf{TN}\}$	An A^e or an erroneous ack. received from a DR
T_{TN}	$\emptyset$	$\{\mathbf{TN}\}$	An ack. from a DR has not arrived on time
RA_{DRj}	$\{\mathbf{TN}\}$	$\{\mathbf{TN}\}$	**TN** received A_{DRj} from $\mathbf{DR}_j$ on time
RA^e_{DRj}	$\{\mathbf{TN}\}$	$\{\mathbf{TN}\}$	**TN** received A^e_{DRj} from $\mathbf{DR}_j$ on time

Table 6.2: Event subalphabets for control nodes ($i \in \{1, 2, 3\}$).

$\mathbf{n} \in \mathbf{I}$	$i \in I$	$\Sigma_{c,i}$	$\Sigma_{o,i}$
SD	1	$\{f, S_{SD}, RA_{TN}, RA^e_{TN}\}$	$\{f, S_{SD}, T_{SD}, RA_{TN}, RA^e_{TN}\}$
TN	2	$\{A_{TN}, A^e_{TN}, RA_{DR1}, RA^e_{DR1}, RA_{DR2}, RA^e_{DR2}\}$	$\{A_{TN}, A^e_{TN}, T_{TN}, RA_{DR1}, RA^e_{DR1}, RA_{DR2}, RA^e_{DR2}\}$
$\mathbf{DR}_1$	3	$\{S_{DR1}, A_{DR1}, A^e_{DR1}, RA_{RN1}, RA^e_{RN1}, RA_{RN2}, RA^e_{RN2}\}$	$\{S_{DR1}, R_{DR1}, R^e_{DR1}, A_{DR1}, A^e_{DR1}, T_{DR1}, RA_{RN1}, RA^e_{RN1}, RA_{RN2}, RA^e_{RN2}\}$
$\mathbf{DR}_2$	4	$\{S_{DR2}, A_{DR2}, A^e_{DR2}, RA_{RN3}, RA^e_{RN3}\}$	$\{S_{DR2}, R_{DR2}, R^e_{DR2}, A_{DR2}, A^e_{DR2}, T_{DR2}, RA_{RN3}, RA^e_{RN3}\}$
$\mathbf{RN}_1$	5	$\{A_{RN1}, A^e_{RN1}\}$	$\{R_{RN1}, R^e_{RN1}, A_{RN1}, A^e_{RN1}, T_{RN1}\}$
$\mathbf{RN}_2$	6	$\{A_{RN2}, A^e_{RN2}\}$	$\{R_{RN2}, R^e_{RN2}, A_{RN2}, A^e_{RN2}, T_{RN2}\}$
$\mathbf{RN}_3$	7	$\{A_{RN3}, A^e_{RN3}\}$	$\{R_{RN3}, R^e_{RN3}, A_{RN3}, A^e_{RN3}, T_{RN3}\}$

Assumption 6.4 It is assumed that the initial state of each model is also its only marked state. This is justified by the fact that in each model the functions of the control node would be considered finished once the node returns to its initial condition, where it is ready to begin its next round of functions. □

Modeling the sender

Figure 6.3 shows the automaton model of **SD**. Accordingly, initially at state r_0 a data packet is fetched (event f) and then sent (event S_{SD}) on multicast channel at state r_1. The sender then waits to receive one of TN's acknowledgements (of the receipt of the data) at state r_2. If it receives A_{TN} (event RA_{TN}), indicating the successful receipt of the sent data packet by all three receivers, it moves to r_0 to fetch a new data packet. Otherwise, if the sender receives A^e_{TN} (event RA^e_{TN}), indicating that the sent data packet has not been received by at least one receiver and cannot be retransmitted by designated

receivers, or if no **TN**'s acknowledgement arrives within a specified time limit (event T_{SD}), it retransmits the data packet (event S_{SD} at r_1) and then waits for its associated acknowledgement at r_2.

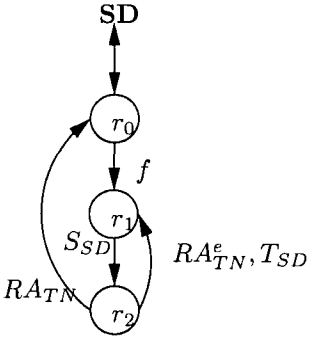


Figure 6.3: The automaton model of sender **SD**.

Following Assumption 6.3, observe that the model prohibits the occurrence of some events at some states; for example f and RA_{TN} cannot happen at r_1 not at r_2, and at r_0 nor at r_1, respectively. For RA_{TN}, this means that the "local control," not studied here, ignores the receipt of A_{TN} from **TN** when not at r_0 and r_1.

Modeling the receivers

Figure 6.4 shows the model of receiver $\mathbf{RN}_i$, where $i \in \{1, 2, 3\}$. Initially the receiver waits (for a specified amount of time) to receive a data packet from either **SD** or the receiver's parent. Upon the on-time and error-free receipt of the data packet (event R_{RNi}), the receiver moves to state r_1 and issues A_{RNi}. On the other hand, if the data does not receive on time (event TN_{RNi}) or it is erroneous (event R^e_{RNi}), the receiver goes from r_0 to r_2, where it issues A^e_{RNi} and returns to the initial state.

Following Assumption 6.3, we assume that the receiver takes action on

the first occurrence of R_{RNi} and R^e_{RNi}, only, and ignores the later occurrences of them which happen between the first occurrence and issuing the corresponding acknowledgement by the receiver. As a result, the later occurrences of these two events do not appear at states r_1 and r_2.

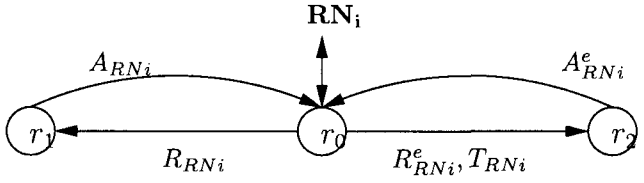


Figure 6.4: The automaton model of receiver $\mathbf{RN}_i$, where $i \in \{1, 2, 3\}$.

Modeling the designated receivers

The automaton models of $\mathbf{DR}_2$ and $\mathbf{DR}_1$ are shown in Fig. 6.5 and Fig. 6.6, respectively. We start by explaining the model for $\mathbf{DR}_2$, which is simpler.

The model of $\mathbf{DR}_2$ has two halves, whose difference is in receiving the data packet from **SD** (event R_{DR2}). In fact, if $\mathbf{DR}_2$ resides in a state in the left half, the occurrence of uncontrollable event R_{DR2} moves it to a counterpart state in the right half. The explanation for the left half is as follows. Initially at state r_0, $\mathbf{DR}_2$ waits to receive the acknowledgement from its child, $\mathbf{RN}_3$. Upon the on-time receiving of a success-indicating acknowledgement A_{RN3} from $\mathbf{RN}_3$ (event RA_{RN3}), $\mathbf{DR}_2$ moves to state r_1. Following this, $\mathbf{DR}_2$ issues an A_{DR2} and moves to r_0. However, if either a) no acknowledgement arrives during the specified time limit (event T_{DR2}), or b) the received acknowledgement is A^e_{RN3}, indicating $\mathbf{RN}_3$'s failure in receiving the data packet, or is an erroneous copy of A_{RN3}, both modeled as event RA^e_{RN3}, then $\mathbf{DR}_2$ is led to r_2 where it issues an A^e_{DR2} and moves to r_0. Notice that it is the first acknowledgement

which is considered by the designated receiver and further acknowledgements are actually neglected by it, as modeled by selfloops at states r_1 and r_2. If $\mathbf{DR}_2$ receives the data packet sent by **SD** error-free and on-time (event R_{DR2}), depending on where R_{DR2} occurs, $\mathbf{DR}_2$ moves to one of the states r_3, r_4, or r_5 in the right half of the automaton, which are counterparts of r_0, r_1, and r_2, respectively. When in the states of the right half, $\mathbf{DR}_2$ has a copy of the data packet so that if RA^e_{RN3} or T_{DR2} occur, it retransmits the data packet to its child, $\mathbf{RN}_3$. However, if the data packet is received erroneous (event R^e_{DR2}), $\mathbf{DR}_2$ does not change its state. This is modeled by adding this event as selfloop at each state, though these selfloops are not shown for the sake of clarity.

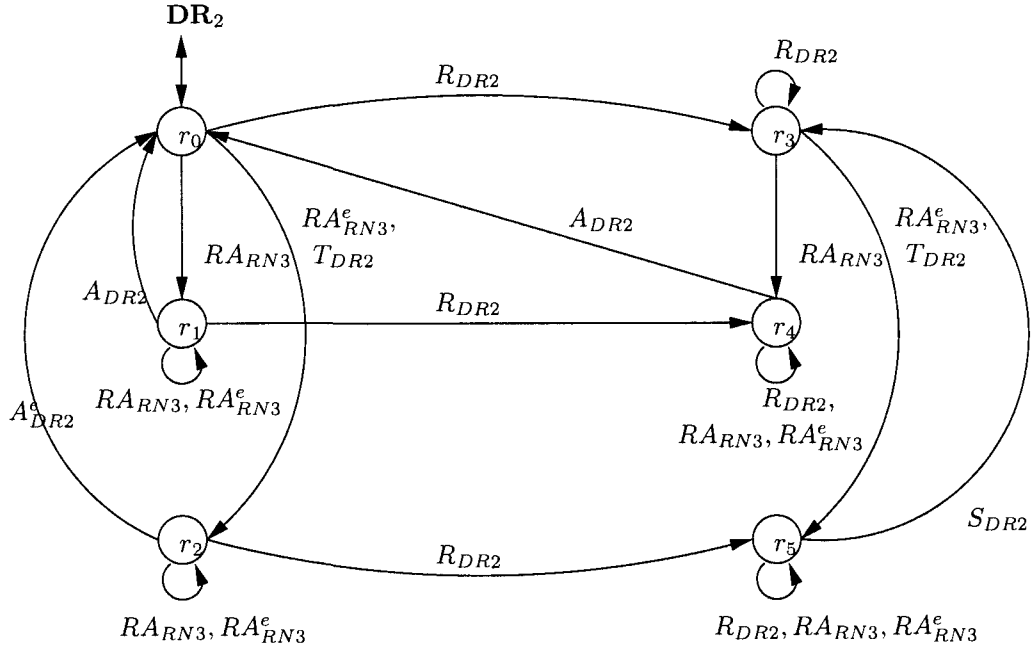


Figure 6.5: Automaton model of $\mathbf{DR}_2$. A selfloop labeled with R^e_{DR2} should be added to every state of the automaton.

Similarly, the model of $\mathbf{DR}_1$ has two halves, which correspond to processing acknowledgements and time-outs without and with having the copy of the data packet. Comparing to the model of $\mathbf{DR}_2$, the only difference is that here $\mathbf{DR}_1$ has two children and should process two acknowledgements, which may arrive in either order, from receivers $\mathbf{RN}_1$ and $\mathbf{RN}_2$.

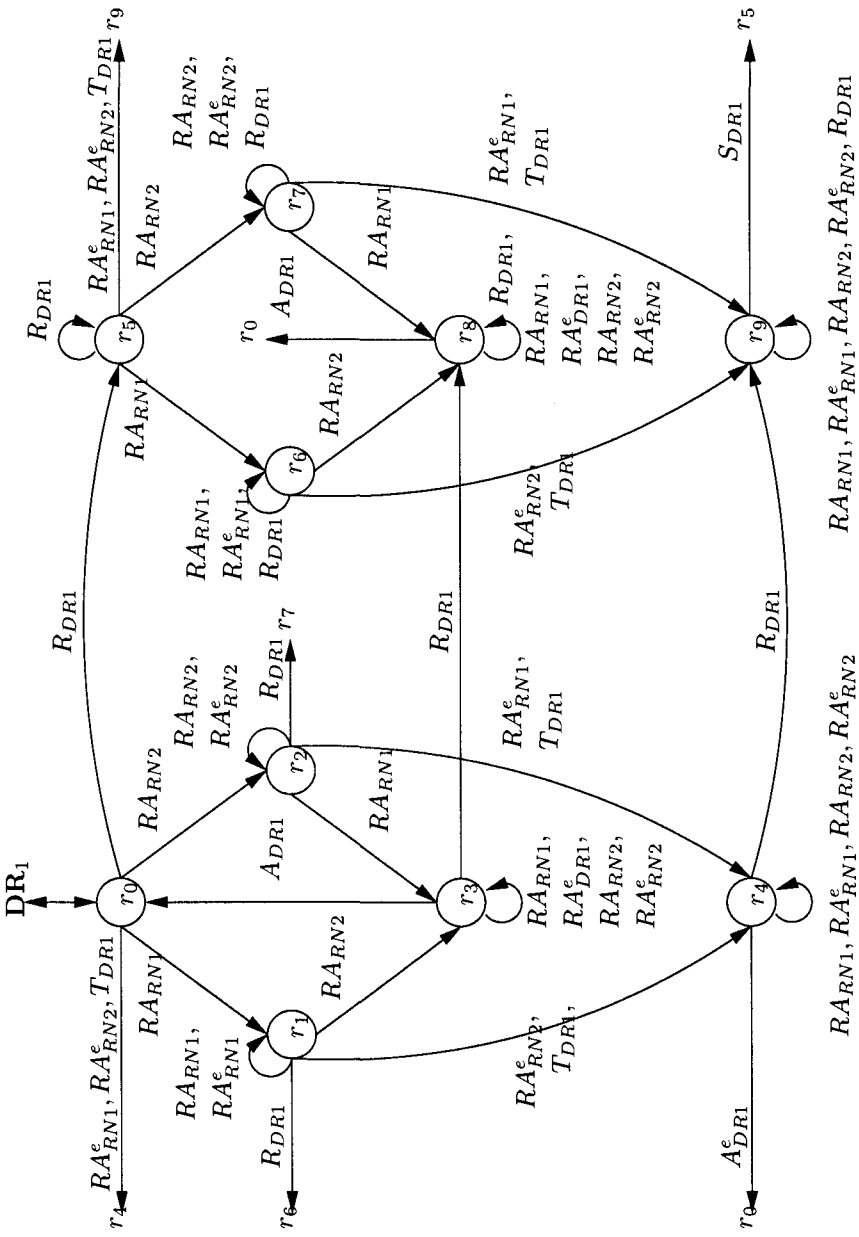


Figure 6.6: Automaton model of $\mathbf{DR}_1$. A selfloop labeled with R^e_{DR1} should be added to every state of the automaton.

Modeling the top node

The automaton model of top node **TN** is shown in Fig. 6.7. Initially, **TN** waits to receive the acknowledgements from both its children, $\mathbf{DR}_1$ and $\mathbf{DR}_2$, within a specified time limit. If both acknowledgements are success-indicating and receive in time, i.e. if events RA_{DR1} and RA_{DR2} arrive in either order, they lead **TN** to r_3, following which **TN** moves to r_0 by issuing A_{TN}. Otherwise, if either acknowledgement is failure-indicating or is an erroneous acknowledgement (event RA^e_{DR1} or RA^e_{DR2}), or either acknowledgement does not arrive within the specified time limit (event T_{TN}), **TN** is led to r_4, following which **TN** moves to r_0 by issuing A^e_{TN}. Similar to the designated receivers, **TN** takes action on the first acknowledgement received from its every child and ignores second and next acknowledgements.

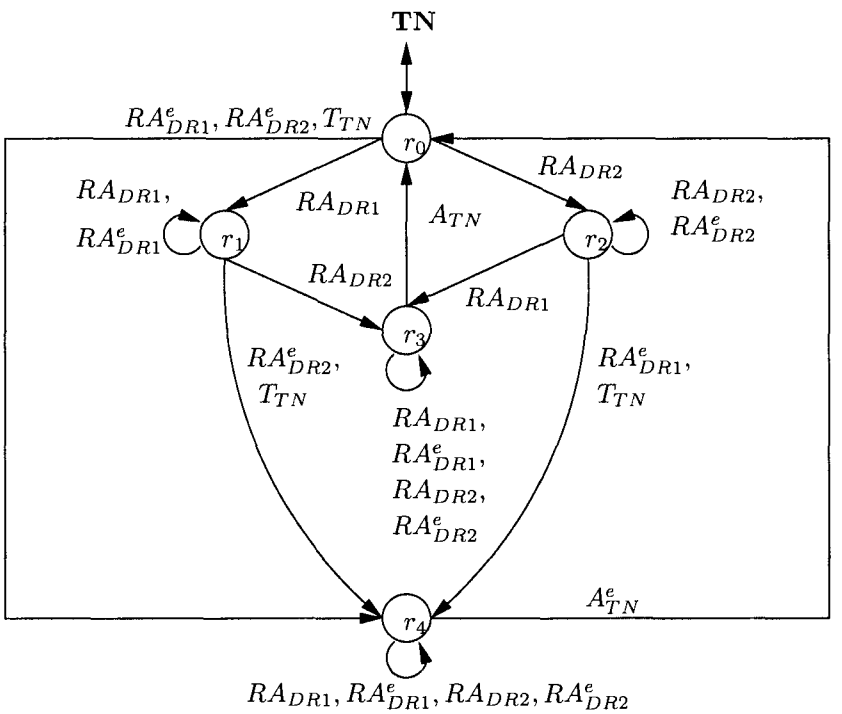


Figure 6.7: The automaton model of top node **TN**.

Model of the network

The model of network, $\mathbf{N}$, is the synchronous product of all control node models, i.e.

$$\mathbf{N} = \mathbf{SD}||\mathbf{TN}||\mathbf{DR}_1||\mathbf{DR}_2||\mathbf{RN}_1||\mathbf{RN}_2||\mathbf{RN}_3,$$

where $||$ denotes the "synchronous product" operation.

6.3.4 RMTP control specifications

The model of every control node $\mathbf{n} \in \mathbf{I}$ in the RMTP network of Fig. 6.2 is defined based on its observable events in $\Sigma_{\mathbf{n}}$. However, certain events of a parent node should follow certain events of its children in order for the network to transfer acknowledgements from bottom to top, yielding a reliable multicast transfer of data packets. The network hierarchical topology helps breaking the "global" reliability constraint into control specifications for each parent and its children. There are four such parents and their children, resulting in a number of desired parent-child behaviors, which are formalized as control specifications in the following.

Assumption 6.5 For each specification, the initial state is considered as the marked state, too. The justification is the same as what mentioned in Assumption 6.4. □

Control specification for $\mathbf{DR}_1$ and its children

Parts (a) and (b) of Fig. 6.8 show the desired joint behavior of $\mathbf{DR}_1$ and its children, $\mathbf{RN}_1$ and $\mathbf{RN}_2$, as two specifications $\mathbf{S}_1$ and $\mathbf{S}_2$. Each specification requires that every child's acknowledgement be followed by its corresponding acknowledgement-received event of $\mathbf{DR}_1$. For example, $\mathbf{S}_1$ requires that A_{RN1}

be followed by RA_{RN1}. This requirement can be modeled simply by alternating an acknowledgement and its corresponding acknowledgement-received event, too, as shown in part (c) for the case of $\mathbf{DR}_1$ and $\mathbf{RN}_1$. This simple model inhibits multiple occurrence of the same acknowledgement, which in turn reduces the network traffic. However, observe that in this case, the synchronous application of the two alternations, one for error-free- and one for erroneous-data acknowledgements, which is shown in part (d), allows the serial issuance of both kinds of acknowledgements before processing each first. This condition would make the processing of the acknowledgements prone to event-order errors and increases the network traffic. Therefore, on top of the alternation of the events of a parent and those of its child, it is required that no acknowledgement (of the same or different type) be issued unless the previously sent acknowledgement be processed first. For example, according to $\mathbf{S}_1$ once A_{RN1} is issued by $\mathbf{RN}_1$, neither it nor A^e_{RN1} can be issued until RA_{RN1} is issued by $\mathbf{DR}_1$. This argument justifies why $\mathbf{S}_1$ and $\mathbf{S}_2$ are modeled as in parts (a) and (b), rather than part (c). Notice that in each part of the figure, all the events in Σ which do not appear in that part, are selflooped at all states of the automaton of that part, but not shown.

Control specification for $\mathbf{DR}_2$ and its child

Figure 6.9 shows the desired joint behavior of $\mathbf{DR}_2$ and its child, $\mathbf{RN}_3$, as specification $\mathbf{S}_3$. The idea behind this specification is similar to those of specifications $\mathbf{S}_1$ and $\mathbf{S}_2$. Notice that all the events in Σ which do not appear in the figure, are selflooped at all states of the automaton, but not shown.

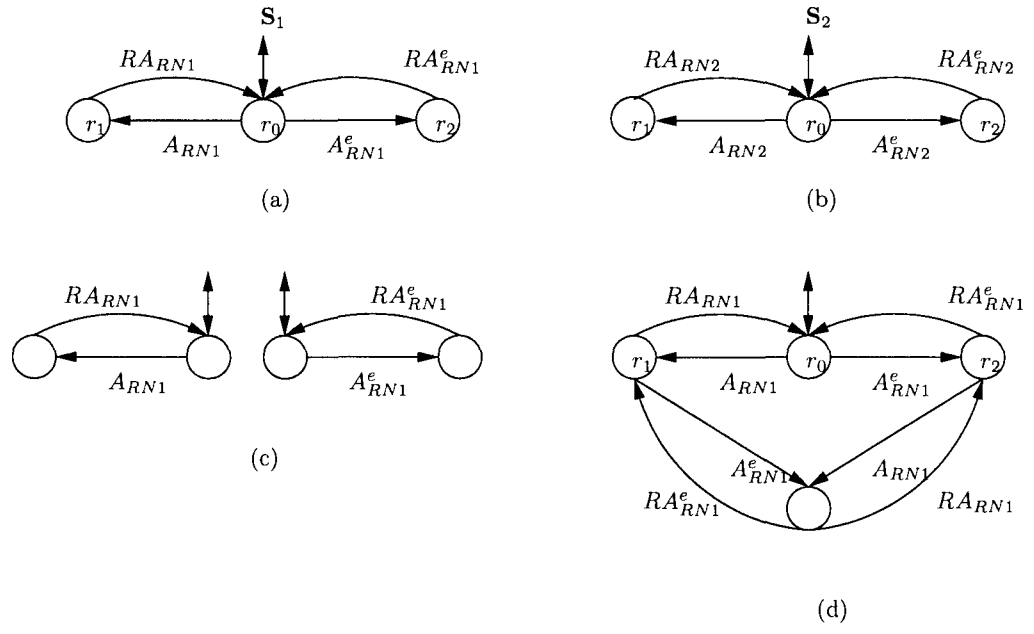


Figure 6.8: (a), (b) The control specifications for the joint behavior of $\mathbf{DR}_1$ and its children, $\mathbf{RN}_1$ and $\mathbf{RN}_2$, respectively. All of the events in $\Sigma \setminus \{A_{RN1}, A^e_{RN1}, RA_{RN1}, RA^e_{RN1}\}$ and every event in $\Sigma \setminus \{A_{RN2}, A^e_{RN2}, RA_{RN2}, RA^e_{RN2}\}$ are selflooped at all states of $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. These selfloops are not shown to keep the figures clear. (c) Two simple requirements for the alternation of $\mathbf{RN}_1$'s two types of acknowledgements and $\mathbf{DR}_1$'s corresponding acknowledgement-received events. (d) Synchronous application of the simple requirements in part (c) allows the serial issuance of multiple acknowledgements by $\mathbf{RN}_1$ without processing them by $\mathbf{DR}_1$, hence justifying the introduction of specification $\mathbf{S}_1$ in part (a).

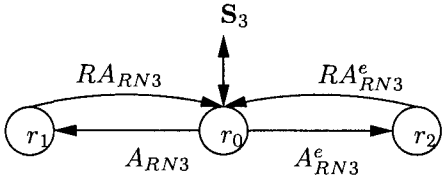


Figure 6.9: The control specification for the joint behavior of $\mathbf{DR}_2$ and its child. All of the events in $\Sigma \setminus \{A_{RN3}, A^e_{RN3}, RA_{RN3}, RA^e_{RN3}\}$ are selflooped at all states, but not shown.

Control specification for TN and its children

Parts (a) and (b) of Fig. 6.10 show the desired joint behavior of **TN** and its children, $\mathbf{DR}_1$ and $\mathbf{DR}_2$, as specifications $\mathbf{S}_4$ and $\mathbf{S}_5$, respectively. The idea behind these specifications is similar to the previous specifications. Notice that all the events in Σ which do not appear in each part of the figure, are selflooped at all states of the automaton of that part, but not shown.

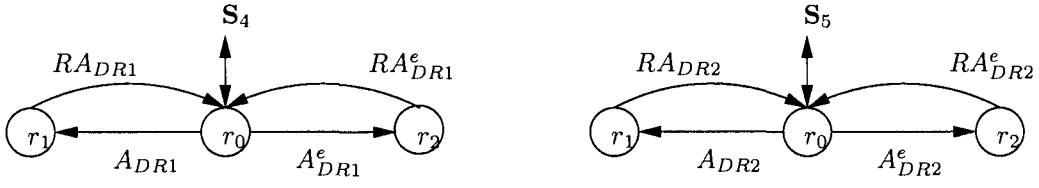


Figure 6.10: The control specifications for the joint behavior of **TN** and its children $\mathbf{DR}_1$ and $\mathbf{DR}_2$, respectively. All of the events in $\Sigma \setminus \{A_{DR1}, A^e_{DR1}, RA_{DR1}, RA^e_{DR1}\}$ and $\Sigma \setminus \{A_{DR2}, A^e_{DR2}, RA_{DR2}, RA^e_{DR2}\}$ are selflooped at all states of the automata in parts (a) and (b), respectively. These selfloops are not shown to keep the figures clear.

Control specification for SD and its child

Figure 6.9 shows the desired joint behavior of **SD** and its child, **TN** as specification $\mathbf{S}_6$. The idea behind this specification is similar to the previous specifications. Notice that all the events in Σ which do not appear in the figure, are selflooped at all states of the automaton, but not shown.

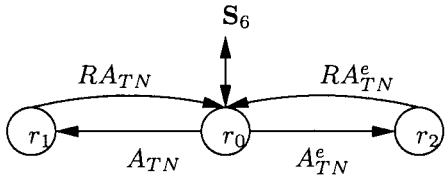


Figure 6.11: The control specification for the joint behavior of **SD** and **TN**. All of the events in $\Sigma \setminus \{A_{TN}, A^e_{TN}, RA_{TN}, RA^e_{TN}\}$ are selflooped at all states, but not shown.

6.4 Synthesis of the RMTP-like information policies for the simple network

This section provides solutions, in the form of information policies, to the communication rules of the RMTP network of Section 6.3. The synthesis procedure follows the general procedure for the synthesis of communicating supervisors, outlined in Chapter 5, but in a modular fashion. Accordingly, for each specification, a centralized supervisor is computed using supervisory control theory, which insures the satisfaction of that specification by the plant (here RMTP network). This results in six centralized supervisor which are shown to be non-conflicting, i.e. their joint supervision does not lead to blocking (or deadlock). Then for each of the six centralized supervisors, an ALM is introduced, using which a DSDES formulation in the form of a polynomial dynamical system (PDS) representation of the supervisor is obtained. This representation is then employed to compute a solution in the form of an information policy. These steps are explained in the following subsections. The following result is used in the subsequent computations.

Theorem 6.1 ([3] Thm. 3.6.2) Let **S** be any nonblocking DES over Σ such that $S := L_m(\mathbf{S})$ satisfies the following two conditions.

1. S is controllable with respect to plant **G**,

2. $\overline{S \cap L_m(\mathbf{G})} = \overline{S} \cap L(\mathbf{G})$.

Let $\emptyset \neq K := S \cap L_m(\mathbf{G})$ and let W be a marking nonblocking supervisory control such that $L_m(W/\mathbf{G}) = K$. Then $\mathbf{S}$ implements W. In particular the following holds.

$$L_m(W/\mathbf{G}) = L_m(\mathbf{G}) \cap L_m(\mathbf{S}), \quad L(W/\mathbf{G}) = L(\mathbf{G}) \cap L(\mathbf{S}) \tag{6.1}$$

■

6.4.1 Centralized supervisors for RMTP control specifications

For each $k \in \{1, 2, 3, 4, 5, 6\}$, specification $\mathbf{S}_k$ enjoys the following properties.

1. $\mathbf{S}_k$ is nonblocking over Σ. This is verified in Appendix C.

2. $L_m(\mathbf{S}_k)$ is controllable with respect to $\mathbf{N}$ (and Σ_c). This is verified in Appendix C.

3. $\overline{S_k \cap L_m(\mathbf{N})} = \overline{S}_k \cap L(\mathbf{N})$. This is verified in Appendix C.

4. $\emptyset \neq E_k := S_k \cap L_m(\mathbf{N})$ is controllable with respect to $\mathbf{N}$ (and Σ_c). This is verified in Appendix C.

5. $L_m(\mathbf{S}_k)$ is observable with respect to $\mathbf{N}$, and natural projection P (induced by Σ_o). This is because all unobservable events, i.e. events in Σ_{uo}, appear as slefloop transitions at all states of $\mathbf{S}_k$.

By Theorem 4.1, the last three properties imply that there exists a nonblocking feasible supervisory control W_k for $\mathbf{N}$ such that $L_m(W_k/\mathbf{N}) = L_m(\mathbf{S}_k)$. Then, by Theorem 6.1 and the first three properties it turns out that $\mathbf{S}_k$ implements

W_k in the sense of (6.1). In other words, each $\mathbf{S}_k$ is a *modular* supervisor for $\mathbf{N}$ which enforces $L_m(\mathbf{S}_k)$. Furthermore, as shown in Appendix C, $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, $\mathbf{S}_4$, $\mathbf{S}_5$, and $\mathbf{S}_6$ are mutually nonconflicting, so their synchronous supervision is nonblocking. This implies that they can be used as modular supervisors to supervise plant $\mathbf{N}$ with no risk of blocking.

As a result, rather than dealing with one big centralized supervisor, which is the synchronous product of all six supervisors, and implementing it in a decentralized way, one can implement six smaller modular supervisors in a decentralized way. Towards this end, for each modular supervisor an ALMs is introduced, based on which a PDS representation is derived which is then used for computation of information policies. In the rest of this chapter, $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, $\mathbf{S}_4$, $\mathbf{S}_5$, and $\mathbf{S}_6$ are referred to as modular supervisors.

Before proceeding to compute PDS representations, it is worth comparing the centralized and modular solutions from the point of view of ALMs and integer variables.

1. First notice that there are 7 control nodes and six modular supervisors, each having 3 states. A centralized solution would have (of the order of) $3^6 = 729$ states. Therefore, a finite field which could encompass (about) 729 distinct elements would be required. However, in the modular case, each required ALM should have 3 distinct elements. This reduction in the size of the underlying finite field(s), shows the advantage of working with the modular solution.

2. On the other hand, the state changes of every modular supervisor are labeled by the events of exactly 2 control nodes and the events of all other control nodes appear as selfloops. For example, all the state changes in $\mathbf{S}_1$ are labeled with events in Σ_3 and Σ_5, and all events in Σ_j, where $j \in \{1, 2, 4, 7\}$, are selflooped, i.e. make no state change anywhere. As

a result, the actual "dimension" of the ALM, used for each modular supervisor, will be 3. Therefore, the order of the underlying finite field is reduced from 7 to 2 and 2, being a second advantage for modular approach.

3. Following the previous observation, the events of a control node result in state change in more than one modular supervisor. For example, $\mathbf{DR}_1$'s events result in state changes in $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_4$. This means that each such supervisor would employ, in its PDS representation, one separate variable of its own to represent the labels assigned to $\mathbf{DR}_1$ by the supervisor's ALM. As a result, in the final PDS representation of the whole network of the modular supervisors, there are control nodes whose related observational information is captured in more than one integer variable.

   The following definition is useful in the next subsection.

Definition 6.1 For $k \in \{1, 2, 3, 4, 5, 6\}$, let $\mathbf{S}_k = (R_k, \Sigma_k, \xi_k, r_{0,k}, R_{m,k})$ be one of the modular supervisors and $\mathbf{n} \in \mathbf{I}$ be one of the control nodes whose index is $i \in I$, assigned by function $\mathbf{F}$ in Subsection 6.3.2. Control node $\mathbf{n}$ is said to *participate* in $\mathbf{S}_k$ if there exists an event $\sigma \in \Sigma_i$ and distinct states $r, r' \in R_k$ such that $r' = \xi(r, \sigma)$. □

For example, control nodes $\mathbf{DR}_1$ and $\mathbf{RN}_1$ participate in $\mathbf{S}_1$, since (at least) they result in state changes from r_0 to r_1 and from r_1 to r_0, respectively. Events of other control nodes appear as selfloop transitions, only, so these nodes do not participate in $\mathbf{S}_1$.

6.4.2 DSDES model of the modular supervisors

DSDES modeling starts by defining ALMs for each modular supervisor. To define an ALM for each modular supervisor, the method of Latin hypercubes, introduced in Chapter 3, is employed. To this end, one needs to know the side and order of the six hypercubes. These two correspond to, respectively, the number of the states of each modular supervisor, and the number of control nodes which participate in the supervisor's automaton. Based on these two measures, the following observations can be made for every supervisor $\mathbf{S}_k$, where $k \in I$.

1. A general model for $\mathbf{S}_k$ can be shown in Fig. 6.12-a, where $i, j \in I$, $i \neq j$, $\alpha_i, \beta_i \in \Sigma_i$, and $\alpha_j, \beta_j \in \Sigma_j$. Selfloop transitions are labeled by $*$, which represents the events in $\Sigma \setminus \{\alpha_i, \beta_i, \alpha_j, \beta_j\}$.

2. $\mathbf{S}_k$ has 3 states and 2 participating control nodes.

3. Moreover, if an event appears as a selfloop transition at some state, it appears as selfloop transition at every other state. In other words, there is no need to unfold any selfloop transitions or add states to $\mathbf{S}_k$ (see Remark 3.1).

The last two observations call for a Latin square of side 3 and dimension 2. Out of different arrangements of such Latin squares, one is shown in Fig. 6.12-b.

Let $\mathbb{F}_3 = \{0, 1, 2\}$ and addition be done modulo 3. Then the ALM, which is built using the Latin square in Fig. 6.12 and is denoted by ℓ, assigns labels to states r_0, r_1, and r_2 as follows.

$$\forall q \in \mathbb{F}_3.\ \ \ell(r_q) = \{(m, n) \mid m, n \in \mathbb{F}_3\ \wedge\ m + n = q\} \tag{6.2}$$

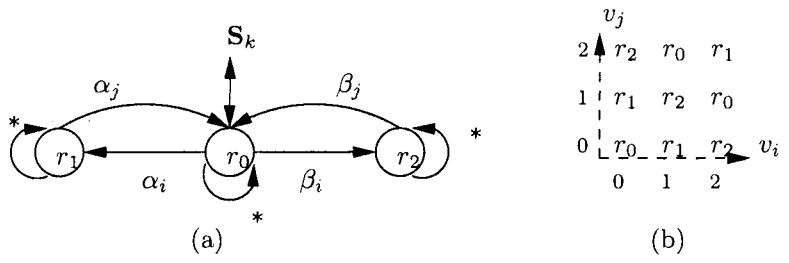


Figure 6.12: (a) General model of the modular supervisors, where $*$ denotes the events in the set $\Sigma \setminus \{\alpha_i, \beta_i, \alpha_j, \beta_j\}$. (b) One arrangement of the Latin squares of side 3 and dimension 2, which is suitable for defining an ALM for each modular supervisor.

This results in the following state labels.

$$\ell(r_0)=\{(0,0),(1,2),(2,1)\},$$
$$\ell(r_1)=\{(0,1),(1,0),(2,2)\},$$
$$\ell(r_2)=\{(0,2),(1,1),(2,0)\}$$

Associated with $\mathbf{S}_k$ and ℓ, the guard functions will be as follows.

$$\mathcal{G}_i(\alpha_i) = \mathcal{G}_i(\beta_i) = \{r_0\},\ \ \mathcal{G}_j(\alpha_j) = \{r_1\},\ \ \mathcal{G}_j(\alpha_j) = \{r_2\}, \tag{6.3}$$

$$\forall \sigma \in (\Sigma_i \setminus \{\alpha_i, \beta_i\}).\ \ \mathcal{G}_i(\sigma) = \mathbb{F}_3^2, \quad \forall \sigma \in (\Sigma_j \setminus \{\alpha_j, \beta_j\}).\ \ \mathcal{G}_j(\sigma) = \mathbb{F}_3^2, \tag{6.4}$$

$$\forall m \in I \setminus \{i,j\}, \forall \sigma \in \Sigma_m.\ \ \mathcal{G}_m(\sigma) = \mathbb{F}_3^2 \tag{6.5}$$

The corresponding nonidentity updating functions are as shown in Table 6.3, where a point $(a,b) \in \mathbb{F}_3^2$ is denoted by ab. The identity updating functions are as follows.

$$\forall \sigma \in (\Sigma_i \cup \Sigma_j \setminus \{\alpha_i, \beta_i, \alpha_j, \beta_j\}), \forall \mathbf{v} \in \mathbb{F}_3.\ \ \mathcal{A}(\sigma, \mathbf{v}) = \mathbf{v}, \tag{6.6}$$

$$\forall m \in I \setminus \{i,j\}, \forall \sigma \in \Sigma_m, \forall \mathbf{v} \in \mathbb{F}_3.\ \ \mathcal{A}(\sigma, \mathbf{v}) = \mathbf{v} \tag{6.7}$$

Table 6.3: Updating functions associated with $\mathbf{S}_k$ and ℓ

$\mathbf{v}$	00	21	12	else
$\mathcal{A}(\alpha_i, \mathbf{v})$	10	01	22	–
$\mathbf{v}$	00	21	12	else
$\mathcal{A}(\beta_i, \mathbf{v})$	20	11	02	–
$\mathbf{v}$	10	01	22	else
$\mathcal{A}(\alpha_j, \mathbf{v})$	12	00	21	–
$\mathbf{v}$	20	11	02	else
$\mathcal{A}(\beta_j, \mathbf{v})$	21	12	00	–

The DSDES model of the network is the collection of the six DSDES models which are obtained for all six modular supervisors and their corresponding ALMs.

6.4.3 PDS representation of the RMTP network

PDS representation of the typical DSDES: To arrive at the PDS representation of the network, we start by computing the PDS representation of the typical DSDES associated with Fig. 6.12. The PDS representation of the network is he collection of six PDS representations, each corresponding to one DSDES model. To this end, first define the following polynomials (see (4.7), (4.8), and (4.9)).

$$L_0(x) = \frac{(x-1)(x-2)}{(-1)(-2)} = 2(x+1)(x+2) \tag{6.8}$$

$$L_1(x) = \frac{x(x-2)}{(1)(-1)} = 2x(x+1) \tag{6.9}$$

$$L_2(x) = \frac{x(x-1)}{(2)(1)} = 2x(x+2) \tag{6.10}$$

In computing the guard polynomials using Algorithm 4.1, notice that

there is no unreachable point in $\mathbb{F}_3$, which can be used to simplify these polynomials (see Remark 4.2). Therefore the following results are obtained.

$$\begin{aligned}\mathfrak{g}_i(\alpha_i) = \mathfrak{g}_i(\beta_i) &= L_{00}(x_i, x_j) + L_{21}(x_i, x_j) + L_{12}(x_i, x_j) \\ &= (x_i^2 + 2)(x_j^2 + 2) + x_i x_j(2x_i x_j + 1) \quad (6.11)\end{aligned}$$

$$\begin{aligned}\mathfrak{g}_j(\alpha_j) &= L_{10}(x_i, x_j) + L_{01}(x_i, x_j) + L_{22}(x_i, x_j) \\ &= 2(x_i + 1)(x_j + 1)(x_i x_j + x_i + x_j) + x_i x_j(x_i + 2)(x_j + 2) \quad (6.12)\end{aligned}$$

$$\begin{aligned}\mathfrak{g}_j(\beta_j) &= L_{20}(x_i, x_j) + L_{11}(x_i, x_j) + L_{02}(x_i, x_j) \\ &= (x_i + 2)(x_j + 2)(2x_i x_j + x_i + x_j) + x_i x_j(x_i + 1)(x_j + 1) \quad (6.13)\end{aligned}$$

Obviously, the rest of guard functions are always equal to 1 and are not mentioned here. Nonidentity updating polynomials are computed as follows.

1. Originally we have $\mathfrak{a}_i^{\alpha_i}(x_i, x_j) = L_{00}(x_i, x_j) + 2L_{12}(x_i, x_j)$. However, α_i is disabled at states r_1 and r_2 and the labels associated with these two states may be used to simplify $\mathfrak{a}_i$. To this end, $L_{01}(x_i, x_j)$, $2L_{10}(x_i, x_j)$, $L_{02}(x_i, x_j)$, and $2L_{11}(x_i, x_j)$ are added to $\mathfrak{a}_i$ leading to the following computations.

$$\begin{aligned}\mathfrak{a}_i^{\alpha_i}(x_i, x_j) &= [L_{00}(x_i, x_j) + L_{01}(x_i, x_j) + L_{02}(x_i, x_j)] \\ &+ 2[L_{12}(x_i, x_j) + L_{10}(x_i, x_j) + L_{11}(x_i, x_j)] = L_0(x_i) + 2L_1(x_i) = x_i + 1 \quad (6.14)\end{aligned}$$

2. Originally we have $\mathfrak{a}_i^{\beta_i}(x_i, x_j) = 2L_{00}(x_i, x_j) + L_{21}(x_i, x_j)$. By a similar

reasoning to part 1, we may add $2L_{10}(x_i, x_j)$, $L_{22}(x_i, x_j)$, $L_{20}(x_i, x_j)$, and $2L_{02}(x_i, x_j)$ to this expression to compute the following.

$$\mathfrak{a}_i^{\beta_i}(x_i, x_j) = x_i + 2 \tag{6.15}$$

3. Originally we have $\mathfrak{a}_j^{\alpha_j}(x_i, x_j) = 2L_{10}(x_i, x_j) + L_{22}(x_i, x_j)$. By a similar reasoning to part 1, we may add $2L_{00}(x_i, x_j)$, $L_{12}(x_i, x_j)$, $2L_{20}(x_i, x_j)$, and $L_{02}(x_i, x_j)$ to this expression to compute the following.

$$\mathfrak{a}_i^{\beta_i}(x_i, x_j) = x_j + 2 \tag{6.16}$$

4. Originally we have $\mathfrak{a}_j^{\beta_j}(x_i, x_j) = L_{20}(x_i, x_j) + 2L_{11}(x_i, x_j)$. By a similar reasoning to part 1, we may add $L_{00}(x_i, x_j)$, $2L_{21}(x_i, x_j)$, $L_{10}(x_i, x_j)$, and $2L_{01}(x_i, x_j)$ to this expression to compute the following.

$$\mathfrak{a}_i^{\beta_i}(x_i, x_j) = x_j + 1 \tag{6.17}$$

Clearly the above polynomials are all independent of their corresponding external variables, i.e. they are independent updating functions (IUFs) (see Section 5.3).

Introduction of private variables: The next step is to introduce integer variables, as required by the PDS representations of different modular supervisors to come up with a concrete PDS representation of the network. To this end, first the number of private variables, which each node requires to represent its information, should be specified. This number is determined by distinguishing the number of distinct modular supervisors in which the control node participates. Accordingly, a separate variable is assigned to the control node for each distinct modular supervisor. Table 6.4 determines in which

Table 6.4: Participation of control nodes in the six modular supervisors

$\mathbf{n} \in \mathbf{I}$	**SD**	**TN**	$\mathbf{DR}_1$	$\mathbf{DR}_2$	$\mathbf{RN}_1$	$\mathbf{RN}_2$	$\mathbf{RN}_3$
$\mathbf{S}_k$ in which **n** participates	$\mathbf{S}_6$	$\mathbf{S}_4, \mathbf{S}_5, \mathbf{S}_6$	$\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_4$	$\mathbf{S}_3, \mathbf{S}_5$	$\mathbf{S}_1$	$\mathbf{S}_2$	$\mathbf{S}_3$

Table 6.5: Private variables owned by control nodes when participating in different modular supervisors

$\mathbf{n} \in \mathbf{I}$	**SD**	**TN**	$\mathbf{DR}_1$	$\mathbf{DR}_2$	$\mathbf{RN}_1$	$\mathbf{RN}_2$	$\mathbf{RN}_3$
n's variable, Used in $\mathbf{S}_k$	x_1, $\mathbf{S}_6$	x_2, $\mathbf{S}_4$	x_3, $\mathbf{S}_1$	x_4, $\mathbf{S}_3$	x_5, $\mathbf{S}_1$	x_6, $\mathbf{S}_2$	x_7,$\mathbf{S}_3$
n's variable, Used in $\mathbf{S}_k$	–	y_2, $\mathbf{S}_5$	y_3, $\mathbf{S}_2$	y_4,$\mathbf{S}_5$	–	–	–
n's variable, Used in $\mathbf{S}_k$	–	z_2, $\mathbf{S}_6$	z_3, $\mathbf{S}_4$	–	–	–	–

modular supervisor(s) each control node participates.

Using Table 6.4, private variables are assigned to control nodes as specified in Table 6.5. Observe that the index of each variable is the index of its associated control node in I. Depending on in how many modular supervisors a control node participates, it employs letters x, y, and z for the lowest-index to the highest-index supervisors. For example, **SD**, whose index is 1 in I, participates only in $\mathbf{S}_6$ and its private variable is x_1. Similarly, the private variables of **TN**, whose index is 2 in I, are x_2 (used with $\mathbf{S}_4$), y_2 (used with $\mathbf{S}_5$), and z_2 (used with $\mathbf{S}_6$).

PDS representation of the whole network: Table 6.6 shows the PDS representation of the RMTP network, which is the collection of the PDS representations of its six aggregate DSDESs, where we define $\mathbf{x} = (x_1, x_2, y_2, z_2, x_3, y_3,$

$z_3, x_4, y_4, x_5, x_6, x_7$). In the following two subsections, two information policies are derived for this PDS representation.

6.4.4 RMTP-like state-transferring information policy for the RMTP network

We start by deriving state-transferring information policy 2 for the network and justify this policy. To this end, first a set of Boolean variables should be introduced for each control node to encode its variables. To this end, the notation in Definition 4.9 is employed, where we notice that the integers in $\mathbb{F}_3$ can be encoded with two Boolean variables. This justifies the introduction of the Boolean variables in Table 6.7. Also recall from Definition 4.9 that the copy of a typical Boolean variable x^k_{jj} which is kept by the i'th node is denoted by x^k_{ij} (same holds for y and z variables).

The next step is to compute the actions and guards associated with the typical supervisor. To this effect, notice that two Boolean variables can encode four different integers and here there are only three integers in $\mathbb{F}_3$. Hence the fourth one, i.e. 3, encoded by 11, can be used arbitrarily to simplify the expressions. Computation of the actions and guards can be summarized as follows.

1. There two kinds of actions associated with the two updating polynomials $x_i := \mathfrak{a}^\alpha_i(\mathbf{x}) = x_i + 1$ and $x_i := \mathfrak{a}^\alpha_i(\mathbf{x}) = x_i + 2$, where x_i and α are an integer variable owned by the i'th control node and an event in $\Sigma_{o,i}$, respectively.

2. To compute the actions corresponding to $x_i := \mathfrak{a}^\alpha_i(\mathbf{x}) = x_i + 1$, it is

Table 6.6: The PDS representation of the RMTP network

$x_1 := \mathfrak{a}_1^{RA_{TN}}(\mathbf{x}) = x_1 + 2,$	$x_1 := \mathfrak{a}_1^{RA^e_{TN}}(\mathbf{x}) = x_1 + 1$
$x_2 := \mathfrak{a}_2^{RA_{DR1}}(\mathbf{x}) = x_2 + 2,$	$x_2 := \mathfrak{a}_2^{RA^e_{DR1}}(\mathbf{x}) = x_2 + 1$
$y_2 := \mathfrak{a}_2^{RA_{DR2}}(\mathbf{x}) = y_2 + 2,$	$y_2 := \mathfrak{a}_2^{RA^e_{DR2}}(\mathbf{x}) = y_2 + 1$
$z_2 := \mathfrak{a}_2^{A_{TN}}(\mathbf{x}) = z_2 + 1,$	$z_2 := \mathfrak{a}_2^{A^e_{TN}}(\mathbf{x}) = z_2 + 2$
$x_3 := \mathfrak{a}_3^{RA_{RN1}}(\mathbf{x}) = x_3 + 2,$	$x_3 := \mathfrak{a}_3^{RA^e_{RN1}}(\mathbf{x}) = x_3 + 1$
$y_3 := \mathfrak{a}_3^{RA_{RN2}}(\mathbf{x}) = y_3 + 2,$	$y_3 := \mathfrak{a}_3^{RA^e_{RN2}}(\mathbf{x}) = y_3 + 1$
$z_3 := \mathfrak{a}_3^{A_{DR1}}(\mathbf{x}) = z_3 + 1,$	$z_3 := \mathfrak{a}_3^{A^e_{DR1}}(\mathbf{x}) = z_3 + 2$
$x_4 := \mathfrak{a}_4^{RA_{RN3}}(\mathbf{x}) = x_4 + 2,$	$x_4 := \mathfrak{a}_4^{RA^e_{RN3}}(\mathbf{x}) = x_4 + 1$
$y_4 := \mathfrak{a}_4^{A_{DR2}}(\mathbf{x}) = y_4 + 1,$	$y_4 := \mathfrak{a}_4^{A^e_{DR2}}(\mathbf{x}) = y_4 + 2$
$x_5 := \mathfrak{a}_5^{A_{RN1}}(\mathbf{x}) = x_5 + 1,$	$x_5 := \mathfrak{a}_5^{A^e_{RN1}}(\mathbf{x}) = x_5 + 2$
$x_6 := \mathfrak{a}_6^{A_{RN2}}(\mathbf{x}) = x_6 + 1,$	$x_6 := \mathfrak{a}_6^{A^e_{RN2}}(\mathbf{x}) = x_6 + 2$
$x_7 := \mathfrak{a}_7^{A_{RN3}}(\mathbf{x}) = x_7 + 1,$	$x_7 := \mathfrak{a}_7^{A^e_{RN3}}(\mathbf{x}) = x_7 + 2$
$\mathfrak{g}_1^{RA_{TN}}(\mathbf{x}) = 2(z_2+1)(x_1+1)(z_2x_1+z_2+x_1) + z_2x_1(z_2+2)(x_1+2)$	
$\mathfrak{g}_1^{RA^e_{TN}}(\mathbf{x}) = (z_2+2)(x_1+2)(2z_2x_1+z_2+x_1) + z_2x_1(z_2+1)(x_1+1)$	
$\mathfrak{g}_2^{RA_{DR1}}(\mathbf{x}) = 2(z_3+1)(x_2+1)(z_3x_2+z_3+x_2) + z_3x_2(z_3+2)(x_2+2)$	
$\mathfrak{g}_2^{RA^e_{DR1}}(\mathbf{x}) = (z_3+2)(x_2+2)(2z_3x_2+z_3+x_2) + z_3x_2(z_3+1)(x_2+1)$	
$\mathfrak{g}_2^{RA_{DR2}}(\mathbf{x}) = 2(y_4+1)(y_2+1)(y_4y_2+y_4+y_2) + y_4y_2(y_4+2)(y_2+2)$	
$\mathfrak{g}_2^{RA^e_{DR2}}(\mathbf{x}) = (y_4+2)(y_2+2)(2y_4y_2+y_4+y_2) + y_4y_2(y_4+1)(y_2+1)$	
$\mathfrak{g}_2^{A_{TN}}(\mathbf{x}) = (z_2^2+2)(x_1^2+2) + z_2x_1(2z_2x_1+1)$	
$\mathfrak{g}_2^{A^e_{TN}}(\mathbf{x}) = (z_2^2+2)(x_1^2+2) + z_2x_1(2z_2x_1+1)$	
$\mathfrak{g}_3^{RA_{RN1}}(\mathbf{x}) = 2(x_5+1)(x_3+1)(x_5x_3+x_5+x_3) + x_5x_3(x_5+2)(x_3+2)$	
$\mathfrak{g}_3^{RA^e_{RN1}}(\mathbf{x}) = (x_5+2)(x_3+2)(2x_5x_3+x_5+x_3) + x_5x_3(x_5+1)(x_3+1)$	
$\mathfrak{g}_3^{RA_{RN2}}(\mathbf{x}) = 2(x_6+1)(y_3+1)(x_6y_3+x_6+y_3) + x_6y_3(x_6+2)(y_3+2)$	
$\mathfrak{g}_3^{RA^e_{RN2}}(\mathbf{x}) = (x_6+2)(y_3+2)(2x_6y_3+x_6+y_3) + x_6y_3(x_6+1)(y_3+1)$	
$\mathfrak{g}_3^{A_{DR1}}(\mathbf{x}) = (z_3^2+2)(x_2^2+2) + z_3x_2(2z_3x_2+1)$	
$\mathfrak{g}_3^{A^e_{DR1}}(\mathbf{x}) = (z_3^2+2)(x_2^2+2) + z_3x_2(2z_3x_2+1)$	
$\mathfrak{g}_4^{RA_{RN3}}(\mathbf{x}) = 2(x_7+1)(x_4+1)(x_7x_4+x_7+x_4) + x_7x_4(x_7+2)(x_4+2)$	
$\mathfrak{g}_4^{RA^e_{RN3}}(\mathbf{x}) = (x_7+2)(x_4+2)(2x_7x_4+x_7+x_4) + x_7x_4(x_7+1)(x_4+1)$	
$\mathfrak{g}_4^{A_{DR2}}(\mathbf{x}) = (y_4^2+2)(y_2^2+2) + y_4y_2(2y_4y_2+1)$	
$\mathfrak{g}_4^{A^e_{DR2}}(\mathbf{x}) = (y_4^2+2)(y_2^2+2) + y_4y_2(2y_4y_2+1)$	
$\mathfrak{g}_5^{A_{RN1}}(\mathbf{x}) = (x_5^2+2)(x_3^2+2) + x_5x_3(2x_5x_3+1)$	
$\mathfrak{g}_5^{A^e_{RN1}}(\mathbf{x}) = (x_5^2+2)(x_3^2+2) + x_5x_3(2x_5x_3+1)$	
$\mathfrak{g}_6^{A_{RN2}}(\mathbf{x}) = (x_6^2+2)(y_3^2+2) + x_6y_3(2x_6y_3+1)$	
$\mathfrak{g}_6^{A^e_{RN2}}(\mathbf{x}) = (x_6^2+2)(y_3^2+2) + x_6y_3(2x_6y_3+1)$	
$\mathfrak{g}_7^{A_{RN3}}(\mathbf{x}) = (x_7^2+2)(x_4^2+2) + x_7x_4(2x_7x_4+1)$	
$\mathfrak{g}_7^{A^e_{RN3}}(\mathbf{x}) = (x_7^2+2)(x_4^2+2) + x_7x_4(2x_7x_4+1)$	

Table 6.7: Private Boolean variables owned by control nodes for state-transferring information policy

Integer variable	Its Boolean-variable encoding
x_1	$(x_{11}^2 x_{11}^1)$
x_2	$(x_{22}^2 x_{22}^1)$
y_2	$(y_{22}^2 y_{22}^1)$
z_2	$(z_{22}^2 z_{22}^1)$
x_3	$(x_{33}^2 x_{33}^1)$
y_3	$(y_{33}^2 y_{33}^1)$
z_3	$(z_{33}^2 z_{33}^1)$
x_4	$(x_{44}^2 x_{44}^1)$
y_4	$(y_{44}^2 y_{44}^1)$
x_5	$(x_{55}^2 x_{55}^1)$
x_6	$(x_{66}^2 x_{66}^1)$
x_7	$(x_{77}^2 x_{77}^1)$

assumed that 3 is mapped to 2. This yields the following actions.

$$x_{ii}^2 := a_i(x_{ii}^2, \alpha) = x_{ii}^1, \quad x_{ii}^1 := a_i(x_{ii}^1, \alpha) = \overline{x}_{ii}^2 \overline{x}_{ii}^1 \tag{6.18}$$

3. To compute the actions corresponding to $x_i := \mathfrak{a}_i^\alpha(\mathbf{x}) = x_i + 2$, it is assumed that 3 is mapped to 1. This yields the following actions.

$$x_{ii}^2 := a_i(x_{ii}^2, \alpha) = \overline{x}_{ii}^2 \overline{x}_{ii}^1, \quad x_{ii}^1 := a_i(x_{ii}^1, \alpha) = x_{ii}^2 \tag{6.19}$$

4. There are three types of guard polynomials: $\mathfrak{g}_i(\alpha_i) == (x_i^2+2)(x_j^2+2)+x_ix_j(2x_ix_j+1)$, $\mathfrak{g}_j(\alpha_j) = 2(x_i+1)(x_j+1)(x_ix_j+x_i+x_j)+x_ix_j(x_i+2)(x_j+2)$ and $\mathfrak{g}_j(\beta_j) = (x_i+2)(x_j+2)(2x_ix_j+x_i+x_j)+x_ix_j(x_i+1)(x_j+1)$, where $\alpha_i \in \Sigma_{o,i}$ and $\alpha_j, \beta_j \in \Sigma_{o,j}$.

5. To compute the guard corresponding to $\mathfrak{g}_i(\alpha_i) = (x_i^2+2)(x_j^2+2)+x_ix_j(2x_ix_j+1)$, it is assumed that α_i is also enabled at the following

points in $\mathbb{F}_3^2$: $(1,3),(2,3),(3,1),(3,2)$, and $(3,3)$. This results in the following expression for the guard.

$$g_i(\alpha_i) = \overline{x}_{ii}^2 \overline{x}_{ii}^1 \overline{x}_{ij}^2 \overline{x}_{ij}^1 + x_{ii}^2 x_{ij}^1 + x_{ii}^1 x_{ij}^2 \tag{6.20}$$

6. To compute the guard corresponding to $\mathfrak{g}_j(\alpha_j) = 2(x_i+1)(x_j+1)(x_i x_j + x_i + x_j) + x_i x_j (x_i+2)(x_j+2)$, it is assumed that α_j is also enabled at the following points in $\mathbb{F}_3^2$: $(0,3),(2,3),(3,0),(3,2)$, and $(3,3)$. This results in the following expression for the guard.

$$g_j(\alpha_j) = x_{ji}^1 \overline{x}_{jj}^2 \overline{x}_{jj}^1 + \overline{x}_{ji}^2 \overline{x}_{ji}^1 x_{jj}^1 + x_{ji}^2 x_{jj}^2 \tag{6.21}$$

7. To compute the guard corresponding to $\mathfrak{g}_j(\beta_j) = (x_i+2)(x_j+2)(2x_i x_j + x_i + x_j) + x_i x_j (x_i+1)(x_j+1)$, it is assumed that α_j is also enabled at the following points in $\mathbb{F}_3^2$: $(0,3),(1,3),(3,0),(3,1),(3,2)$, and $(3,3)$. This results in the following expression for the guard.

$$g_j(\beta_j) = x_{ji}^2 \overline{x}_{jj}^2 \overline{x}_{jj}^1 + \overline{x}_{ji}^2 \overline{x}_{ji}^1 x_{jj}^2 + x_{ji}^1 x_{jj}^1 \tag{6.22}$$

The above results lead to the actions and guards for the whole RMTP network in Table 6.8 and Table 6.9, respectively.

Communication policy 2 (see Definition 5.4) requires that the value of every external variable, on which a control node depends functionally (i.e. as an argument of one of its guards or actions), be sent right after it gets changed. Updating functions of the nodes all depend on private variables, i.e. only guards may depend on external variables. Accordingly, Table 6.10,

Table 6.8: Actions of the control nodes of the RMTP network

$x_{11}^2 := a_1(x_{11}^2, RA_{TN}) = \overline{x}_{11}^2\overline{x}_{11}^1,$ $x_{11}^1 := a_1(x_{11}^1, RA_{TN}) = x_{11}^2$	$x_{11}^2 := a_1(x_{11}^2, RA_{TN}^e) = x_{11}^1$ $x_{11}^1 := a_1(x_{11}^1, RA_{TN}^e) = \overline{x}_{11}^2\overline{x}_{11}^1$
$x_{22}^2 := a_2(x_{22}^2, RA_{DR1}) = \overline{x}_{22}^2\overline{x}_{22}^1,$ $x_{22}^1 := a_2(x_{22}^1, RA_{DR1}) = x_{22}^2$	$x_{22}^2 := a_2(x_{22}^2, RA_{DR1}^e) = x_{22}^1$ $x_{22}^1 := a_2(x_{22}^1, RA_{DR1}^e) = \overline{x}_{22}^2\overline{x}_{22}^1$
$y_{22}^2 := a_2(y_{22}^2, RA_{DR2}) = \overline{y}_{22}^2\overline{y}_{22}^1,$ $y_{22}^1 := a_2(y_{22}^1, RA_{DR2}) = y_{22}^2$	$y_{22}^2 := a_2(y_{22}^2, RA_{DR2}^e) = y_{22}^1$ $y_{22}^1 := a_2(y_{22}^1, RA_{DR2}^e) = \overline{y}_{22}^2\overline{y}_{22}^1$
$z_{22}^2 := a_2(z_{22}^2, A_{TN}) = z_{22}^1,$ $z_{22}^1 := a_2(z_{22}^1, A_{TN}) = \overline{z}_{22}^2\overline{z}_{22}^1$	$z_{22}^2 := a_2(z_{22}^2, A_{TN}^e) = \overline{z}_{22}^2\overline{z}_{22}^1$ $z_{22}^1 := a_2(z_{22}^1, A_{TN}^e) = z_{22}^2$
$x_{33}^2 := a_3(x_{33}^2, RA_{RN1}) = \overline{x}_{33}^2\overline{x}_{33}^1,$ $x_{33}^1 := a_3(x_{ii}^1, RA_{RN1}) = x_{33}^2$	$x_{33}^2 := a_i(x_{33}^2, RA_{RN1}^e) = x_{33}^1$ $x_{33}^1 := a_i(x_{33}^1, RA_{RN1}^e) = \overline{x}_{33}^2\overline{x}_{33}^1$
$y_{33}^2 := a_3(y_{33}^2, RA_{RN2}) = \overline{y}_{33}^2\overline{y}_{33}^1,$ $y_{33}^1 := a_3(y_{ii}^1, RA_{RN2}) = y_{33}^2$	$y_{33}^2 := a_i(y_{33}^2, RA_{RN2}^e) = y_{33}^1$ $y_{33}^1 := a_i(y_{33}^1, RA_{RN2}^e) = \overline{y}_{33}^2\overline{y}_{33}^1$
$z_{33}^2 := a_3(z_{33}^2, A_{DR1}) = z_{33}^1,$ $z_{33}^1 := a_3(z_{33}^1, A_{DR1}) = \overline{z}_{33}^2\overline{z}_{33}^1$	$z_{33}^2 := a_3(z_{33}^2, A_{DR1}^e) = \overline{z}_{33}^2\overline{z}_{33}^1$ $z_{33}^1 := a_3(z_{ii}^1, A_{DR1}^e) = z_{33}^2$
$x_{44}^2 := a_4(x_{44}^2, RA_{RN3}) = \overline{x}_{44}^2\overline{x}_{44}^1,$ $x_{44}^1 := a_4(x_{44}^1, RA_{RN3}) = x_{44}^2$	$x_{44}^2 := a_4(x_{44}^2, RA_{RN3}^e) = x_{44}^1$ $x_{44}^1 := a_4(x_{44}^1, RA_{RN3}^e) = \overline{x}_{44}^2\overline{x}_{44}^1$
$y_{44}^2 := a_i(y_{44}^2, A_{DR2}) = y_{44}^1,$ $y_{44}^1 := a_4(y_{44}^1, A_{DR2}) = \overline{y}_{44}^2\overline{y}_{44}^1$	$y_{44}^2 := a_4(y_{44}^2, A_{DR2}^e) = \overline{y}_{44}^2\overline{y}_{44}^1$ $y_{44}^1 := a_4(y_{44}^1, A_{DR2}^e) = y_{44}^2$
$x_{55}^2 := a_5(x_{55}^2, A_{RN1}) = x_{55}^1,$ $x_{55}^1 := a_5(x_{55}^1, A_{RN1}) = \overline{x}_{55}^2\overline{x}_{55}^1$	$x_{55}^2 := a_5(x_{55}^2, A_{RN1}^e) = \overline{x}_{55}^2\overline{x}_{55}^1$ $x_{55}^1 := a_5(x_{55}^1, A_{RN1}^e) = x_{55}^2$
$x_{66}^2 := a_6(x_{66}^2, A_{RN2}) = x_{66}^1,$ $x_{66}^1 := a_6(x_{66}^1, A_{RN2}) = \overline{x}_{66}^2\overline{x}_{66}^1$	$x_{66}^2 := a_6(x_{66}^2, A_{RN2}^e) = \overline{x}_{66}^2\overline{x}_{66}^1$ $x_{66}^1 := a_6(x_{66}^1, A_{RN2}^e) = x_{66}^2$
$x_{77}^2 := a_7(x_{77}^2, A_{RN3}) = x_{77}^1,$ $x_{77}^1 := a_7(x_{77}^1, A_{RN3}) = \overline{x}_{77}^2\overline{x}_{77}^1$	$x_{77}^2 := a_7(x_{77}^2, A_{RN3}^e) = \overline{x}_{77}^2\overline{x}_{77}^1$ $x_{77}^1 := a_7(x_{77}^1, A_{RN3}^e) = x_{77}^2$

Table 6.9: Guards of the control nodes of the RMTP network

$g_1(RA_{TN}) = z_{12}^1 \overline{x}_{11}^2 \overline{x}_{11}^1 + \overline{z}_{12}^2 \overline{z}_{12}^1 x_{11}^1 + z_{12}^2 x_{11}^2$
$g_1(RA_{TN}^e) = z_{12}^2 \overline{x}_{11}^2 \overline{x}_{11}^1 + \overline{z}_{12}^2 \overline{z}_{12}^1 x_{11}^2 + z_{12}^1 x_{11}^1$
$g_2(RA_{DR1}) = z_{23}^1 \overline{x}_{22}^2 \overline{x}_{22}^1 + \overline{z}_{23}^2 \overline{z}_{23}^1 x_{22}^1 + z_{23}^2 x_{22}^2$
$g_2(RA_{DR1}^e) = z_{23}^2 \overline{x}_{22}^2 \overline{x}_{22}^1 + \overline{z}_{23}^2 \overline{z}_{23}^1 x_{22}^2 + z_{23}^1 x_{22}^1$
$g_2(RA_{DR2}) = y_{24}^1 \overline{y}_{22}^2 \overline{y}_{22}^1 + \overline{y}_{24}^2 \overline{y}_{24}^1 y_{22}^1 + y_{24}^2 y_{22}^2$
$g_2(RA_{DR2}^e) = y_{24}^2 \overline{y}_{22}^2 \overline{y}_{22}^1 + \overline{y}_{24}^2 \overline{y}_{24}^1 y_{22}^2 + y_{24}^1 y_{22}^1$
$g_2(A_{TN}) = \overline{z}_{22}^2 \overline{z}_{22}^1 \overline{x}_{21}^2 \overline{x}_{21}^1 + z_{22}^2 x_{21}^1 + z_{22}^1 x_{21}^2$
$g_2(A_{TN}^e) = \overline{z}_{22}^2 \overline{z}_{22}^1 \overline{x}_{21}^2 \overline{x}_{21}^1 + z_{22}^2 x_{21}^1 + z_{22}^1 x_{21}^2$
$g_3(RA_{RN1}) = x_{35}^1 \overline{x}_{33}^2 \overline{x}_{33}^1 + \overline{x}_{35}^2 \overline{x}_{35}^1 x_{33}^1 + x_{35}^2 x_{33}^2$
$g_3(RA_{RN1}^e) = x_{35}^2 \overline{x}_{33}^2 \overline{x}_{33}^1 + \overline{x}_{35}^2 \overline{x}_{35}^1 x_{33}^2 + x_{35}^1 x_{33}^1$
$g_3(RA_{RN2}) = x_{36}^1 \overline{y}_{33}^2 \overline{y}_{33}^1 + \overline{x}_{36}^2 \overline{x}_{36}^1 y_{33}^1 + x_{36}^2 y_{33}^2$
$g_3(RA_{RN2}^e) = x_{36}^1 \overline{y}_{33}^2 \overline{y}_{33}^1 + \overline{x}_{36}^2 \overline{x}_{36}^1 y_{33}^1 + x_{36}^2 y_{33}^2$
$g_3(A_{DR1}) = \overline{z}_{33}^2 \overline{z}_{33}^1 \overline{x}_{32}^2 \overline{x}_{32}^1 + z_{33}^2 x_{32}^1 + z_{33}^1 x_{32}^2$
$g_3(A_{DR1}^e) = \overline{z}_{33}^2 \overline{z}_{33}^1 \overline{x}_{32}^2 \overline{x}_{32}^1 + z_{33}^2 x_{32}^1 + z_{33}^1 x_{32}^2$
$g_4(RA_{RN3}) = x_{47}^1 \overline{x}_{44}^2 \overline{x}_{44}^1 + \overline{x}_{47}^2 \overline{x}_{47}^1 x_{44}^1 + x_{47}^2 x_{44}^2$
$g_4(RA_{RN3}^e) = x_{47}^2 \overline{x}_{44}^2 \overline{x}_{44}^1 + \overline{x}_{47}^2 \overline{x}_{47}^1 x_{44}^2 + x_{47}^1 x_{44}^1$
$g_4(A_{DR2}) = \overline{y}_{44}^2 \overline{y}_{44}^1 \overline{y}_{42}^2 \overline{y}_{42}^1 + y_{44}^2 y_{42}^1 + y_{44}^1 y_{42}^2$
$g_4(A_{DR2}^e) = \overline{y}_{44}^2 \overline{y}_{44}^1 \overline{y}_{42}^2 \overline{y}_{42}^1 + y_{44}^2 y_{42}^1 + y_{44}^1 y_{42}^2$
$g_5(A_{RN1}) = \overline{x}_{55}^2 \overline{x}_{55}^1 \overline{x}_{53}^2 \overline{x}_{53}^1 + x_{55}^2 x_{53}^1 + x_{55}^1 x_{53}^2$
$g_5(A_{RN1}^e) = \overline{x}_{55}^2 \overline{x}_{55}^1 \overline{x}_{53}^2 \overline{x}_{53}^1 + x_{55}^2 x_{53}^1 + x_{55}^1 x_{53}^2$
$g_6(A_{RN2}) = \overline{x}_{66}^2 \overline{x}_{66}^1 \overline{y}_{63}^2 \overline{y}_{63}^1 + x_{66}^2 y_{63}^1 + x_{66}^1 y_{63}^2$
$g_6(A_{RN2}^e) = \overline{x}_{66}^2 \overline{x}_{66}^1 \overline{y}_{63}^2 \overline{y}_{63}^1 + x_{66}^2 y_{63}^1 + x_{66}^1 y_{63}^2$
$g_7(A_{RN3}) = \overline{x}_{77}^2 \overline{x}_{77}^1 \overline{x}_{74}^2 \overline{x}_{74}^1 + x_{77}^2 x_{74}^1 + x_{77}^1 x_{74}^2$
$g_7(A_{RN3}^e) = \overline{x}_{77}^2 \overline{x}_{77}^1 \overline{x}_{74}^2 \overline{x}_{74}^1 + x_{77}^2 x_{74}^1 + x_{77}^1 x_{74}^2$

Table 6.11, and Table 6.12 illustrate the implementation of the information policy 2 for the network.

6.4.5 RMTP-like event-transferring information policy for the RMTP network

This subsection presents an implementation of information policy 5, introduced in Definition 5.15, for the RMTP network. Recall that policy 5 prescribes communication of the (parts of) the event-encoding Boolean variables, under Assumption 5.1.

Computation of $\mathcal{O}_i$ for each control node $i \in I$: Using Algorithm 5.1 for the PDS in Table 6.6, and noticing the fact that the updating functions of the network are all independent, the following results are obtained.

$$\mathcal{O}_1 = \{2\},\ \mathcal{O}_2 = \{1,3,4\},\ \mathcal{O}_3 = \{2,5,6\},\ \mathcal{O}_4 = \{2,7\},$$
$$\mathcal{O}_5 = \{3\},\ \mathcal{O}_6 = \{3\},\ \mathcal{O}_7 = \{4\} \quad (6.23)$$

Introduction of event-encoding Boolean variables: Following Definition 5.13, the observable events of each control node, which result in a state-change in one of the six modular supervisors, i.e. have nonidentity updating functions, are encoded. The list of these events, the supervisor(s) in which they make a state change, and their Boolean encodings are shown in Table 6.13. For the sake of later referencing, these events are defined to form a set themselves

Table 6.10: Communication amongst supervisors based on policy 2

$$\mathscr{I}_{12}(RA_{TN}) = \begin{cases} \{x_{11}^1, x_{11}^2\} & ; \text{ if } (x_{11}^1 \oplus [x_{11}^2]) \wedge (x_{11}^2 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \\ \{x_{11}^1\} & ; \text{ if } (x_{11}^1 \oplus [x_{11}^2]) \wedge \neg(x_{11}^2 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \\ \{x_{11}^2\} & ; \text{ if } \neg(x_{11}^1 \oplus [x_{11}^2]) \wedge (x_{11}^2 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \\ \emptyset & ; \text{ if } \neg(x_{11}^1 \oplus [x_{11}^2]) \wedge \neg(x_{11}^2 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \end{cases}$$

$$\mathscr{I}_{12}(RA_{TN}^e) = \begin{cases} \{x_{11}^1, x_{11}^2\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge (x_{11}^2 \oplus x_{11}^1) \\ \{x_{11}^1\} & ; \text{ if } (x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge \neg(x_{11}^2 \oplus x_{11}^1) \\ \{x_{11}^2\} & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge (x_{11}^2 \oplus x_{11}^1) \\ \emptyset & ; \text{ if } \neg(x_{11}^1 \oplus [\overline{x}_{11}^2 \overline{x}_{11}^1]) \wedge \neg(x_{11}^2 \oplus x_{11}^1) \end{cases}$$

$$\forall j \in I \setminus \{2\}, \forall \alpha \in \Sigma_{o,1}. \quad \mathscr{I}_{1j}(\alpha) = \emptyset$$

$$\mathscr{I}_{21}(A_{TN}) = \begin{cases} \{z_{22}^1, z_{22}^2\} & ; \text{ if } (z_{22}^1 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \wedge (z_{22}^2 \oplus [z_{22}^1]) \\ \{z_{22}^1\} & ; \text{ if } (z_{22}^1 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \wedge \neg(z_{22}^2 \oplus [z_{22}^1]) \\ \{z_{22}^2\} & ; \text{ if } \neg(z_{22}^1 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \wedge (z_{22}^2 \oplus [z_{22}^1]) \\ \emptyset & ; \text{ if } \neg(z_{22}^1 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \wedge \neg(z_{22}^2 \oplus [z_{22}^1]) \end{cases}$$

$$\mathscr{I}_{21}(A_{TN}^e) = \begin{cases} \{z_{22}^1, z_{22}^2\} & ; \text{ if } (z_{22}^1 \oplus [z_{22}^2]) \wedge (z_{22}^2 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \\ \{z_{22}^1\} & ; \text{ if } (z_{22}^1 \oplus [z_{22}^2]) \wedge \neg(z_{22}^2 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \\ \{z_{22}^2\} & ; \text{ if } \neg(z_{22}^1 \oplus [z_{22}^2]) \wedge (z_{22}^2 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \\ \emptyset & ; \text{ if } \neg(x_{22}^1 \oplus [z_{22}^2]) \wedge \neg(x_{22}^2 \oplus [\overline{z}_{22}^2 \overline{z}_{22}^1]) \end{cases}$$

$$\mathscr{I}_{23}(RA_{DR1}) = \begin{cases} \{x_{22}^1, x_{22}^2\} & ; \text{ if } (x_{22}^1 \oplus [x_{22}^2]) \wedge (x_{22}^2 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \\ \{x_{22}^1\} & ; \text{ if } (x_{22}^1 \oplus [x_{22}^2]) \wedge \neg(x_{22}^2 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \\ \{x_{22}^2\} & ; \text{ if } \neg(x_{22}^1 \oplus [x_{22}^2]) \wedge (x_{22}^2 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \\ \emptyset & ; \text{ if } \neg(x_{22}^1 \oplus [x_{22}^2]) \wedge \neg(x_{22}^2 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \end{cases}$$

$$\mathscr{I}_{23}(RA_{DR1}^e) = \begin{cases} \{x_{22}^1, x_{22}^2\} & ; \text{ if } (x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge (x_{22}^2 \oplus x_{22}^1) \\ \{x_{22}^1\} & ; \text{ if } (x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge \neg(x_{22}^2 \oplus x_{22}^1) \\ \{x_{22}^2\} & ; \text{ if } \neg(x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge (x_{22}^2 \oplus x_{22}^1) \\ \emptyset & ; \text{ if } \neg(x_{22}^1 \oplus [\overline{x}_{22}^2 \overline{x}_{22}^1]) \wedge \neg(x_{22}^2 \oplus x_{22}^1) \end{cases}$$

$$\mathscr{I}_{24}(RA_{DR2}) = \begin{cases} \{y_{22}^1, y_{22}^2\} & ; \text{ if } (y_{22}^1 \oplus [y_{22}^2]) \wedge (y_{22}^2 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \\ \{y_{22}^1\} & ; \text{ if } (y_{22}^1 \oplus [y_{22}^2]) \wedge \neg(y_{22}^2 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \\ \{y_{22}^2\} & ; \text{ if } \neg(y_{22}^1 \oplus [y_{22}^2]) \wedge (y_{22}^2 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \\ \emptyset & ; \text{ if } \neg(y_{22}^1 \oplus [y_{22}^2]) \wedge \neg(y_{22}^2 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \end{cases}$$

$$\mathscr{I}_{24}(RA_{DR2}^e) = \begin{cases} \{y_{22}^1, y_{22}^2\} & ; \text{ if } (y_{22}^1 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \wedge (y_{22}^2 \oplus y_{22}^1) \\ \{y_{22}^1\} & ; \text{ if } (y_{22}^1 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \wedge \neg(y_{22}^2 \oplus y_{22}^1) \\ \{y_{22}^2\} & ; \text{ if } \neg(y_{22}^1 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \wedge (y_{22}^2 \oplus y_{22}^1) \\ \emptyset & ; \text{ if } \neg(y_{22}^1 \oplus [\overline{y}_{22}^2 \overline{y}_{22}^1]) \wedge \neg(y_{22}^2 \oplus y_{22}^1) \end{cases}$$

$$\forall j \in I \setminus \{1,3,4\}, \forall \alpha \in \Sigma_{o,2}. \quad \mathscr{I}_{2j}(\alpha) = \emptyset$$

Table 6.11: Communication amongst supervisors based on policy 2 (Part 2)

$$\mathscr{I}_{32}(A_{DR1}) = \begin{cases} \{z^1_{33}, z^2_{33}\} & ; \text{ if } (z^1_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \wedge (z^2_{33} \oplus [z^1_{33}]) \\ \{z^1_{33}\} & ; \text{ if } (z^1_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \wedge \neg(z^2_{33} \oplus [z^1_{33}]) \\ \{z^2_{33}\} & ; \text{ if } \neg(z^1_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \wedge (z^2_{33} \oplus [z^1_{33}]) \\ \emptyset & ; \text{ if } \neg(z^1_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \wedge \neg(z^2_{33} \oplus [z^1_{33}]) \end{cases}$$

$$\mathscr{I}_{32}(A^e_{DR1}) = \begin{cases} \{z^1_{33}, z^2_{33}\} & ; \text{ if } (z^1_{33} \oplus [z^2_{33}]) \wedge (z^2_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \\ \{z^1_{33}\} & ; \text{ if } (z^1_{33} \oplus [z^2_{33}]) \wedge \neg(z^2_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \\ \{z^2_{33}\} & ; \text{ if } \neg(z^1_{33} \oplus [z^2_{33}]) \wedge (z^2_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \\ \emptyset & ; \text{ if } \neg(x^1_{33} \oplus [z^2_{33}]) \wedge \neg(x^2_{33} \oplus [\overline{z}^2_{33}\overline{z}^1_{33}]) \end{cases}$$

$$\mathscr{I}_{35}(RA_{RN1}) = \begin{cases} \{x^1_{33}, x^2_{33}\} & ; \text{ if } (x^1_{33} \oplus [x^2_{33}]) \wedge (x^2_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \\ \{x^1_{33}\} & ; \text{ if } (x^1_{33} \oplus [x^2_{33}]) \wedge \neg(x^2_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \\ \{x^2_{33}\} & ; \text{ if } \neg(x^1_{33} \oplus [x^2_{33}]) \wedge (x^2_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \\ \emptyset & ; \text{ if } \neg(x^1_{33} \oplus [x^2_{33}]) \wedge \neg(x^2_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \end{cases}$$

$$\mathscr{I}_{35}(RA^e_{RN1}) = \begin{cases} \{x^1_{33}, x^2_{33}\} & ; \text{ if } (x^1_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \wedge (x^2_{33} \oplus x^1_{33}) \\ \{x^1_{33}\} & ; \text{ if } (x^1_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \wedge \neg(x^2_{33} \oplus x^1_{33}) \\ \{x^2_{33}\} & ; \text{ if } \neg(x^1_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \wedge (x^2_{33} \oplus x^1_{33}) \\ \emptyset & ; \text{ if } \neg(x^1_{33} \oplus [\overline{x}^2_{33}\overline{x}^1_{33}]) \wedge \neg(x^2_{33} \oplus x^1_{33}) \end{cases}$$

$$\mathscr{I}_{36}(RA_{RN2}) = \begin{cases} \{y^1_{33}, y^2_{33}\} & ; \text{ if } (y^1_{33} \oplus [y^2_{33}]) \wedge (y^2_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \\ \{y^1_{33}\} & ; \text{ if } (y^1_{33} \oplus [y^2_{33}]) \wedge \neg(y^2_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \\ \{y^2_{33}\} & ; \text{ if } \neg(y^1_{33} \oplus [y^2_{33}]) \wedge (y^2_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \\ \emptyset & ; \text{ if } \neg(y^1_{33} \oplus [y^2_{33}]) \wedge \neg(y^2_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \end{cases}$$

$$\mathscr{I}_{36}(RA^e_{RN2}) = \begin{cases} \{y^1_{33}, y^2_{33}\} & ; \text{ if } (y^1_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \wedge (y^2_{33} \oplus y^1_{33}) \\ \{y^1_{33}\} & ; \text{ if } (y^1_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \wedge \neg(y^2_{33} \oplus y^1_{33}) \\ \{y^2_{33}\} & ; \text{ if } \neg(y^1_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \wedge (y^2_{33} \oplus y^1_{33}) \\ \emptyset & ; \text{ if } \neg(y^1_{33} \oplus [\overline{y}^2_{33}\overline{y}^1_{33}]) \wedge \neg(y^2_{33} \oplus y^1_{33}) \end{cases}$$

$$\forall j \in I \setminus \{2,5,6\}, \forall \alpha \in \Sigma_{o,3}. \quad \mathscr{I}_{3j}(\alpha) = \emptyset$$

$$\mathscr{I}_{42}(A_{DR2}) = \begin{cases} \{y^1_{44}, y^2_{44}\} & ; \text{ if } (y^1_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \wedge (y^2_{44} \oplus [y^1_{44}]) \\ \{y^1_{44}\} & ; \text{ if } (y^1_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \wedge \neg(y^2_{44} \oplus [y^1_{44}]) \\ \{y^2_{44}\} & ; \text{ if } \neg(y^1_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \wedge (y^2_{44} \oplus [y^1_{44}]) \\ \emptyset & ; \text{ if } \neg(y^1_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \wedge \neg(y^2_{44} \oplus [y^1_{44}]) \end{cases}$$

$$\mathscr{I}_{42}(A^e_{DR2}) = \begin{cases} \{y^1_{44}, y^2_{44}\} & ; \text{ if } (y^1_{44} \oplus [y^2_{44}]) \wedge (y^2_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \\ \{y^1_{44}\} & ; \text{ if } (y^1_{44} \oplus [y^2_{44}]) \wedge \neg(y^2_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \\ \{y^2_{44}\} & ; \text{ if } \neg(y^1_{44} \oplus [y^2_{44}]) \wedge (y^2_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \\ \emptyset & ; \text{ if } \neg(y^1_{44} \oplus [y^2_{44}]) \wedge \neg(y^2_{44} \oplus [\overline{y}^2_{44}\overline{y}^1_{44}]) \end{cases}$$

$$\mathscr{I}_{47}(RA_{RN3}) = \begin{cases} \{x^1_{44}, x^2_{44}\} & ; \text{ if } (x^1_{44} \oplus [x^2_{44}]) \wedge (x^2_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \\ \{x^1_{44}\} & ; \text{ if } (x^1_{44} \oplus [x^2_{44}]) \wedge \neg(x^2_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \\ \{x^2_{44}\} & ; \text{ if } \neg(x^1_{44} \oplus [x^2_{44}]) \wedge (x^2_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \\ \emptyset & ; \text{ if } \neg(x^1_{44} \oplus [x^2_{44}]) \wedge \neg(x^2_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \end{cases}$$

$$\mathscr{I}_{47}(RA^e_{RN3}) = \begin{cases} \{x^1_{44}, x^2_{44}\} & ; \text{ if } (x^1_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \wedge (x^2_{44} \oplus x^1_{44}) \\ \{x^1_{44}\} & ; \text{ if } (x^1_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \wedge \neg(x^2_{44} \oplus x^1_{44}) \\ \{x^2_{44}\} & ; \text{ if } \neg(x^1_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \wedge (x^2_{44} \oplus x^1_{44}) \\ \emptyset & ; \text{ if } \neg(x^1_{44} \oplus [\overline{x}^2_{44}\overline{x}^1_{44}]) \wedge \neg(x^2_{44} \oplus x^1_{44}) \end{cases}$$

$$\forall j \in I \setminus \{2,7\}, \forall \alpha \in \Sigma_{o,4}. \quad \mathscr{I}_{4j}(\alpha) = \emptyset$$

Table 6.12: Communication amongst supervisors based on policy 2 (Part 3)

$$\mathscr{I}_{53}(A_{RN1}) = \begin{cases} \{x^1_{55}, x^2_{55}\} & ; \text{ if } (x^1_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \wedge (x^2_{55} \oplus [x^1_{55}]) \\ \{x^1_{55}\} & ; \text{ if } (x^1_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \wedge \neg(x^2_{55} \oplus [x^1_{55}]) \\ \{x^2_{55}\} & ; \text{ if } \neg(x^1_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \wedge (x^2_{55} \oplus [x^1_{55}]) \\ \emptyset & ; \text{ if } \neg(x^1_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \wedge \neg(x^2_{55} \oplus [x^1_{55}]) \end{cases}$$

$$\mathscr{I}_{53}(A^e_{RN1}) = \begin{cases} \{x^1_{55}, x^2_{55}\} & ; \text{ if } (x^1_{55} \oplus [x^2_{55}]) \wedge (x^2_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \\ \{x^1_{55}\} & ; \text{ if } (x^1_{55} \oplus [x^2_{55}]) \wedge \neg(x^2_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \\ \{x^2_{55}\} & ; \text{ if } \neg(x^1_{55} \oplus [x^2_{55}]) \wedge (x^2_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \\ \emptyset & ; \text{ if } \neg(x^1_{55} \oplus [x^2_{55}]) \wedge \neg(x^2_{55} \oplus [\overline{x}^2_{55}\overline{x}^1_{55}]) \end{cases}$$

$$\forall j \in I \setminus \{3\}, \forall \alpha \in \Sigma_{o,5}.\ \ \mathscr{I}_{5j}(\alpha) = \emptyset$$

$$\mathscr{I}_{63}(A_{RN2}) = \begin{cases} \{x^1_{66}, x^2_{66}\} & ; \text{ if } (x^1_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \wedge (x^2_{66} \oplus [x^1_{66}]) \\ \{x^1_{66}\} & ; \text{ if } (x^1_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \wedge \neg(x^2_{66} \oplus [x^1_{66}]) \\ \{x^2_{66}\} & ; \text{ if } \neg(x^1_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \wedge (x^2_{66} \oplus [x^1_{66}]) \\ \emptyset & ; \text{ if } \neg(x^1_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \wedge \neg(x^2_{66} \oplus [x^1_{66}]) \end{cases}$$

$$\mathscr{I}_{63}(A^e_{RN2}) = \begin{cases} \{x^1_{66}, x^2_{66}\} & ; \text{ if } (x^1_{66} \oplus [x^2_{66}]) \wedge (x^2_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \\ \{x^1_{66}\} & ; \text{ if } (x^1_{66} \oplus [x^2_{66}]) \wedge \neg(x^2_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \\ \{x^2_{66}\} & ; \text{ if } \neg(x^1_{66} \oplus [x^2_{66}]) \wedge (x^2_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \\ \emptyset & ; \text{ if } \neg(x^1_{66} \oplus [x^2_{66}]) \wedge \neg(x^2_{66} \oplus [\overline{x}^2_{66}\overline{x}^1_{66}]) \end{cases}$$

$$\forall j \in I \setminus \{3\}, \forall \alpha \in \Sigma_{o,6}.\ \ \mathscr{I}_{6j}(\alpha) = \emptyset$$

$$\mathscr{I}_{74}(A_{RN3}) = \begin{cases} \{x^1_{77}, x^2_{77}\} & ; \text{ if } (x^1_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \wedge (x^2_{77} \oplus [x^1_{77}]) \\ \{x^1_{77}\} & ; \text{ if } (x^1_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \wedge \neg(x^2_{77} \oplus [x^1_{77}]) \\ \{x^2_{77}\} & ; \text{ if } \neg(x^1_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \wedge (x^2_{77} \oplus [x^1_{77}]) \\ \emptyset & ; \text{ if } \neg(x^1_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \wedge \neg(x^2_{77} \oplus [x^1_{77}]) \end{cases}$$

$$\mathscr{I}_{74}(A^e_{RN3}) = \begin{cases} \{x^1_{77}, x^2_{77}\} & ; \text{ if } (x^1_{77} \oplus [x^2_{77}]) \wedge (x^2_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \\ \{x^1_{77}\} & ; \text{ if } (x^1_{77} \oplus [x^2_{77}]) \wedge \neg(x^2_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \\ \{x^2_{77}\} & ; \text{ if } \neg(x^1_{77} \oplus [x^2_{77}]) \wedge (x^2_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \\ \emptyset & ; \text{ if } \neg(x^1_{77} \oplus [x^2_{77}]) \wedge \neg(x^2_{77} \oplus [\overline{x}^2_{77}\overline{x}^1_{77}]) \end{cases}$$

$$\forall j \in I \setminus \{4\}, \forall \alpha \in \Sigma_{o,7}.\ \ \mathscr{I}_{7j}(\alpha) = \emptyset$$

as follows.

$$\Sigma_{oc,1} = \{RA_{TN}, RA^e_{TN}\} \tag{6.24}$$

$$\Sigma_{oc,2} = \{A_{TN}, A^e_{TN}, RA_{DR1}, RA^e_{RD1}, RA_{DR2}, RA^e_{DR2}\} \tag{6.25}$$

$$\Sigma_{oc,3} = \{A_{DR1}, A^e_{DR1}, RA_{RN1}, RA^e_{RN1}, RA_{RN2}, RA^e_{RN2}\} \tag{6.26}$$

$$\Sigma_{oc,4} = \{A_{DR2}, A^e_{DR2}, RA_{RN3}, RA^e_{RN3}\} \tag{6.27}$$

$$\Sigma_{oc,5} = \{A_{RN1}, A^e_{RN1}\} \tag{6.28}$$

$$\Sigma_{oc,6} = \{A_{RN2}, A^e_{RN2}\} \tag{6.29}$$

$$\Sigma_{oc,7} = \{A_{RN3}, A^e_{RN3}\} \tag{6.30}$$

In column 3 of the table, by "Related supervisor" we mean the modular supervisor in which the event makes a state change. From the table it can also be observed that control nodes **SD**, $\mathbf{RN}_1$, $\mathbf{RN}_2$, and $\mathbf{RN}_3$ each employs 2 Boolean variables for encoding their events and each of the other three nodes uses 3 Boolean variables for this purpose. Moreover, all nodes encodes "ϵ" by their 0 codes. Furthermore, since often both types of an acknowledgement, i.e. error-free- and erroneous-receipt acknowledgements, appear to be in the same supervisor, the codes assigned to them is such that the distance between them[4] is just one bit, so that a change from one acknowledgement to the other induces only one changed bit.

Implementation of information policy 5: Recall from Definition 5.15 that policy 5 requires that each supervisor provides other supervisors, who directly or indirectly depend on it, the latest values of the changed bits of event encodings. Notice that (6.23) specifies informational dependency of nodes and

[4]The distance here is considered in the sense of the number of bits which are different in the two.

Table 6.13: Details of the encoding of the events for communication policy 5

Control node, Number	Event's name	Related supervisor	Boolean encoding
SD, 1	ϵ	—	$(y_{11}^2 y_{11}^1) = (00)$
SD, 1	RA_{TN}	$\mathbf{S}_6$	$(y_{11}^2 y_{11}^1) = (10)$
SD, 1	RA_{TN}^e	$\mathbf{S}_6$	$(y_{11}^2 y_{11}^1) = (11)$
TN, 2	ϵ	—	$(y_{22}^3 y_{22}^2 y_{22}^1) = (000)$
TN, 2	A_{TN}	$\mathbf{S}_6$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (010)$
TN, 2	A_{TN}^e	$\mathbf{S}_6$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (011)$
TN, 2	RA_{DR1}	$\mathbf{S}_4$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (100)$
TN, 2	RA_{DR1}^e	$\mathbf{S}_4$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (101)$
TN, 2	RA_{DR2}	$\mathbf{S}_5$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (110)$
TN, 2	RA_{DR2}^e	$\mathbf{S}_5$	$(y_{22}^3 y_{22}^2 y_{22}^1) = (111)$
$\mathbf{DR}_1$, 3	ϵ	—	$(y_{33}^3 y_{33}^2 y_{33}^1) = (000)$
$\mathbf{DR}_1$, 3	A_{DR1}	$\mathbf{S}_4$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (010)$
$\mathbf{DR}_1$, 3	A_{DR1}^e	$\mathbf{S}_4$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (011)$
$\mathbf{DR}_1$, 3	RA_{RN1}	$\mathbf{S}_1$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (100)$
$\mathbf{DR}_1$, 3	RA_{RN1}^e	$\mathbf{S}_1$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (101)$
$\mathbf{DR}_1$, 3	RA_{RN2}	$\mathbf{S}_2$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (110)$
$\mathbf{DR}_1$, 3	RA_{RN2}^e	$\mathbf{S}_2$	$(y_{33}^3 y_{33}^2 y_{33}^1) = (111)$
$\mathbf{DR}_2$, 4	ϵ	—	$(y_{44}^3 y_{44}^2 y_{44}^1) = (000)$
$\mathbf{DR}_2$, 4	A_{DR2}	$\mathbf{S}_5$	$(y_{44}^3 y_{44}^2 y_{44}^1) = (010)$
$\mathbf{DR}_2$, 4	A_{DR2}^e	$\mathbf{S}_5$	$(y_{44}^3 y_{44}^2 y_{44}^1) = (011)$
$\mathbf{DR}_2$, 4	RA_{RN3}	$\mathbf{S}_3$	$(y_{44}^3 y_{44}^2 y_{44}^1) = (100)$
$\mathbf{DR}_2$, 4	RA_{RN3}^e	$\mathbf{S}_3$	$(y_{44}^3 y_{44}^2 y_{44}^1) = (101)$
$\mathbf{RN}_1$, 5	ϵ	—	$(y_{55}^2 y_{55}^1) = (00)$
$\mathbf{RN}_1$, 5	A_{RN1}	$\mathbf{S}_1$	$(y_{55}^2 y_{55}^1) = (10)$
$\mathbf{RN}_1$, 5	A_{RN1}^e	$\mathbf{S}_1$	$(y_{55}^2 y_{55}^1) = (11)$
$\mathbf{RN}_2$, 6	ϵ	—	$(y_{66}^2 y_{66}^1) = (00)$
$\mathbf{RN}_2$, 6	A_{RN2}	$\mathbf{S}_2$	$(y_{66}^2 y_{66}^1) = (10)$
$\mathbf{RN}_2$, 6	A_{RN2}^e	$\mathbf{S}_2$	$(y_{66}^2 y_{66}^1) = (11)$
$\mathbf{RN}_3$, 7	ϵ	—	$(y_{77}^2 y_{77}^1) = (00)$
$\mathbf{RN}_3$, 7	A_{RN3}	$\mathbf{S}_3$	$(y_{77}^2 y_{77}^1) = (10)$
$\mathbf{RN}_3$, 7	A_{RN3}^e	$\mathbf{S}_3$	$(y_{77}^2 y_{77}^1) = (11)$

each event in Table 6.13 is observable by only one node. Table 6.14 and Table 6.15 presents the implementation of policy 5 for the whole network.

6.5 Conclusion

This chapter applies the theoretical results of the previous chapters to the modeling and synthesis of an industrial-size communication protocol, called RMTP. After briefly reviewing the essential properties of RMTP, its network, and its associated nodes, a network consisting of seven nodes is modeled through introducing the events and the automata of each node. Next, the specifications of the network are modeled and shown to satisfy certain properties such as controllability and observability. This paves the way to design modular supervisors which enforce the specifications. These supervisors are then represented in DSDES framework and their PDS representations are obtained. The chapter ends by deriving two RMTP-like information policies for the network.

Table 6.14: Communication amongst supervisors based on policy 5 (Part 1)

$$\forall \alpha \in \Sigma_{oc,1}.\ \mathscr{I}_{12}(\alpha) = \begin{cases} \{\hat{y}_{11}^1, \hat{y}_{11}^2\} & ;\ \text{if } (\hat{y}_{11}^1 \neq y_{11}^1) \wedge (\hat{y}_{11}^2 \neq y_{11}^2) \\ \{\hat{y}_{11}^1\} & ;\ \text{if } (\hat{y}_{11}^1 \neq y_{11}^1) \wedge (\hat{y}_{11}^2 = y_{11}^2) \\ \{\hat{y}_{11}^2\} & ;\ \text{if } (\hat{y}_{11}^1 = y_{11}^1) \wedge (\hat{y}_{11}^2 \neq y_{11}^2) \\ \emptyset & ;\ \text{if } (\hat{y}_{11}^1 = y_{11}^1) \wedge (\hat{y}_{11}^2 = y_{11}^2) \end{cases}$$

$$\forall \sigma \in \Sigma_{o,1} \setminus \Sigma_{oc,1}, \forall j \in I \setminus \{2\}.\ \mathscr{I}_{1j}(\sigma) = \emptyset$$

$$\forall \alpha \in \Sigma_{oc,2}, \forall j \in \{1,3,4\}.$$

$$\mathscr{I}_{2j}(\alpha) = \begin{cases} \{\hat{y}_{22}^1, \hat{y}_{22}^2, \hat{y}_{22}^3\} & ;\ \text{if } (\hat{y}_{22}^1 \neq y_{22}^1) \wedge (\hat{y}_{22}^2 \neq y_{22}^2) \wedge (\hat{y}_{22}^3 \neq y_{22}^3) \\ \{\hat{y}_{22}^1, \hat{y}_{22}^2\} & ;\ \text{if } (\hat{y}_{22}^1 \neq y_{22}^1) \wedge (\hat{y}_{22}^2 \neq y_{22}^2) \wedge (\hat{y}_{22}^3 = y_{22}^3) \\ \{\hat{y}_{22}^1, \hat{y}_{22}^3\} & ;\ \text{if } (\hat{y}_{22}^1 \neq y_{22}^1) \wedge (\hat{y}_{22}^2 = y_{22}^2) \wedge (\hat{y}_{22}^3 \neq y_{22}^3) \\ \{\hat{y}_{22}^2, \hat{y}_{22}^3\} & ;\ \text{if } (\hat{y}_{22}^1 = y_{22}^1) \wedge (\hat{y}_{22}^2 \neq y_{22}^2) \wedge (\hat{y}_{22}^3 \neq y_{22}^3) \\ \{\hat{y}_{22}^1\} & ;\ \text{if } (\hat{y}_{22}^1 \neq y_{22}^1) \wedge (\hat{y}_{22}^2 = y_{22}^2) \wedge (\hat{y}_{22}^3 = y_{22}^3) \\ \{\hat{y}_{22}^2\} & ;\ \text{if } (\hat{y}_{22}^1 = y_{22}^1) \wedge (\hat{y}_{22}^2 \neq y_{22}^2) \wedge (\hat{y}_{22}^3 = y_{22}^3) \\ \{\hat{y}_{22}^3\} & ;\ \text{if } (\hat{y}_{22}^1 = y_{22}^1) \wedge (\hat{y}_{22}^2 = y_{22}^2) \wedge (\hat{y}_{22}^3 \neq y_{22}^3) \\ \emptyset & ;\ \text{if } (\hat{y}_{22}^1 = y_{22}^1) \wedge (\hat{y}_{22}^2 = y_{22}^2) \wedge (\hat{y}_{22}^3 = y_{22}^3) \end{cases}$$

$$\forall \sigma \in \Sigma_{o,2} \setminus \Sigma_{oc,2}, \forall j \in I \setminus \{1,3,4\}.\ \mathscr{I}_{2j}(\sigma) = \emptyset$$

$$\forall \alpha \in \Sigma_{oc,3}, \forall j \in \{2,5,6\}.$$

$$\mathscr{I}_{3j}(\alpha) = \begin{cases} \{\hat{y}_{33}^1, \hat{y}_{33}^2, \hat{y}_{33}^3\} & ;\ \text{if } (\hat{y}_{33}^1 \neq y_{33}^1) \wedge (\hat{y}_{33}^2 \neq y_{33}^2) \wedge (\hat{y}_{33}^3 \neq y_{33}^3) \\ \{\hat{y}_{33}^1, \hat{y}_{33}^2\} & ;\ \text{if } (\hat{y}_{33}^1 \neq y_{33}^1) \wedge (\hat{y}_{33}^2 \neq y_{33}^2) \wedge (\hat{y}_{33}^3 = y_{33}^3) \\ \{\hat{y}_{33}^1, \hat{y}_{33}^3\} & ;\ \text{if } (\hat{y}_{33}^1 \neq y_{33}^1) \wedge (\hat{y}_{33}^2 = y_{33}^2) \wedge (\hat{y}_{33}^3 \neq y_{33}^3) \\ \{\hat{y}_{33}^2, \hat{y}_{33}^3\} & ;\ \text{if } (\hat{y}_{33}^1 = y_{33}^1) \wedge (\hat{y}_{33}^2 \neq y_{33}^2) \wedge (\hat{y}_{33}^3 \neq y_{33}^3) \\ \{\hat{y}_{33}^1\} & ;\ \text{if } (\hat{y}_{33}^1 \neq y_{33}^1) \wedge (\hat{y}_{33}^2 = y_{33}^2) \wedge (\hat{y}_{33}^3 = y_{33}^3) \\ \{\hat{y}_{33}^2\} & ;\ \text{if } (\hat{y}_{33}^1 = y_{33}^1) \wedge (\hat{y}_{33}^2 \neq y_{33}^2) \wedge (\hat{y}_{33}^3 = y_{33}^3) \\ \{\hat{y}_{33}^3\} & ;\ \text{if } (\hat{y}_{33}^1 = y_{33}^1) \wedge (\hat{y}_{33}^2 = y_{33}^2) \wedge (\hat{y}_{33}^3 \neq y_{33}^3) \\ \emptyset & ;\ \text{if } (\hat{y}_{33}^1 = y_{33}^1) \wedge (\hat{y}_{33}^2 = y_{33}^2) \wedge (\hat{y}_{33}^3 = y_{33}^3) \end{cases}$$

$$\forall \sigma \in \Sigma_{o,3} \setminus \Sigma_{oc,3}, \forall j \in I \setminus \{2,5,6\}.\ \mathscr{I}_{3j}(\sigma) = \emptyset$$

$$\forall \alpha \in \Sigma_{oc,4}, \forall j \in \{2,7\}.$$

$$\mathscr{I}_{4j}(\alpha) = \begin{cases} \{\hat{y}_{44}^1, \hat{y}_{44}^2, \hat{y}_{44}^3\} & ;\ \text{if } (\hat{y}_{44}^1 \neq y_{44}^1) \wedge (\hat{y}_{44}^2 \neq y_{44}^2) \wedge (\hat{y}_{44}^3 \neq y_{44}^3) \\ \{\hat{y}_{44}^1, \hat{y}_{44}^2\} & ;\ \text{if } (\hat{y}_{44}^1 \neq y_{44}^1) \wedge (\hat{y}_{44}^2 \neq y_{44}^2) \wedge (\hat{y}_{44}^3 = y_{44}^3) \\ \{\hat{y}_{44}^1, \hat{y}_{44}^3\} & ;\ \text{if } (\hat{y}_{44}^1 \neq y_{44}^1) \wedge (\hat{y}_{44}^2 = y_{44}^2) \wedge (\hat{y}_{44}^3 \neq y_{44}^3) \\ \{\hat{y}_{44}^2, \hat{y}_{44}^3\} & ;\ \text{if } (\hat{y}_{44}^1 = y_{44}^1) \wedge (\hat{y}_{44}^2 \neq y_{44}^2) \wedge (\hat{y}_{44}^3 \neq y_{44}^3) \\ \{\hat{y}_{44}^1\} & ;\ \text{if } (\hat{y}_{44}^1 \neq y_{44}^1) \wedge (\hat{y}_{44}^2 = y_{44}^2) \wedge (\hat{y}_{44}^3 = y_{44}^3) \\ \{\hat{y}_{44}^2\} & ;\ \text{if } (\hat{y}_{44}^1 = y_{44}^1) \wedge (\hat{y}_{44}^2 \neq y_{44}^2) \wedge (\hat{y}_{44}^3 = y_{44}^3) \\ \{\hat{y}_{44}^3\} & ;\ \text{if } (\hat{y}_{44}^1 = y_{44}^1) \wedge (\hat{y}_{44}^2 = y_{44}^2) \wedge (\hat{y}_{44}^3 \neq y_{44}^3) \\ \emptyset & ;\ \text{if } (\hat{y}_{44}^1 = y_{44}^1) \wedge (\hat{y}_{44}^2 = y_{44}^2) \wedge (\hat{y}_{44}^3 = y_{44}^3) \end{cases}$$

$$\forall \sigma \in \Sigma_{o,4} \setminus \Sigma_{oc,4}, \forall j \in I \setminus \{2,7\}.\ \mathscr{I}_{4j}(\sigma) = \emptyset$$

Table 6.15: Communication amongst supervisors based on policy 5 (Part 2)

$\forall i \in \{5,6\}, \forall \alpha \in \Sigma_{oc,i}.\ \mathscr{I}_{i3}(\alpha) = \begin{cases} \{\hat{y}_{ii}^1, \hat{y}_{ii}^2\} & ; \text{ if } (\hat{y}_{ii}^1 \neq y_{ii}^1) \wedge (\hat{y}_{ii}^2 \neq y_{ii}^2) \\ \{\hat{y}_{ii}^1\} & ; \text{ if } (\hat{y}_{ii}^1 \neq y_{ii}^1) \wedge (\hat{y}_{ii}^2 = y_{ii}^2) \\ \{\hat{y}_{ii}^2\} & ; \text{ if } (\hat{y}_{ii}^1 = y_{ii}^1) \wedge (\hat{y}_{ii}^2 \neq y_{ii}^2) \\ \emptyset & ; \text{ if } (\hat{y}_{ii}^1 = y_{ii}^1) \wedge (\hat{y}_{ii}^2 = y_{ii}^2) \end{cases}$
$\forall i \in \{5,6\}, \forall \sigma \in \Sigma_{o,i} \setminus \Sigma_{oc,i}, \forall j \in I \setminus \{3\}.\ \mathscr{I}_{ij}(\sigma) = \emptyset$
$\forall \alpha \in \Sigma_{oc,7}.\ \mathscr{I}_{74}(\alpha) = \begin{cases} \{\hat{y}_{77}^1, \hat{y}_{77}^2\} & ; \text{ if } (\hat{y}_{77}^1 \neq y_{77}^1) \wedge (\hat{y}_{77}^2 \neq y_{77}^2) \\ \{\hat{y}_{77}^1\} & ; \text{ if } (\hat{y}_{77}^1 \neq y_{77}^1) \wedge (\hat{y}_{77}^2 = y_{77}^2) \\ \{\hat{y}_{77}^2\} & ; \text{ if } (\hat{y}_{77}^1 = y_{77}^1) \wedge (\hat{y}_{77}^2 \neq y_{77}^2) \\ \emptyset & ; \text{ if } (\hat{y}_{77}^1 = y_{77}^1) \wedge (\hat{y}_{77}^2 = y_{77}^2) \end{cases}$
$\forall \sigma \in \Sigma_{o,7} \setminus \Sigma_{oc,7}, \forall j \in I \setminus \{4\}.\ \mathscr{I}_{7j}(\sigma) = \emptyset$

Chapter 7

Conclusions and future work

Given a (distributed) DES as a plant and some specifications, this thesis studies the problem of "synthesis of communicating decentralized supervisors," which insures the satisfaction of the specifications by the plant's behavior under the joint supervision of decentralized supervisors which communicate (part of) their private information amongst them.

The proposed approach suggests that this problem can be reduced to decentralized implementation of an already designed centralized supervisor, which itself insures that the plant's behavior satisfies the specifications. Thereby, the original problem is reduced to synthesis of communication among supervisors using a polynomial-dynamical-system representation of the centralized supervisor. The communication, formalized by its associated "communication events," is designed in two levels, called "information policy" and "routing policy." Whereas the former represents the "logical informational dependency" of decentralized supervisors on each other, the latter is to "implement" the exchange of the required information among supervisors in the absence of some mutual communication channels or in the presence of communication losses. This thesis derives only information policies, whose primary role is to reflect

the supervisors' mutual informational dependencies. However, they can be implemented by themselves, i.e. with no routing policy, in strongly connected networks whose communication channels are delay-free. When either condition is violated, a routing policy should be designed, on top of an information policy, to implement the communication among supervisors.

Besides deriving six solutions in the form of information policies, the research provides a finer partitioning of the class of communicating supervisors and also establishes some relations between communication and the state representation of the centralized supervisor.

The theoretical results are applied to design two RMTP-like information policies for multicasting data over asymmetric networks.

7.1 Conclusions

The advantages of the proposed distributed SDES and EFSM frameworks are explained i n Chapter 1. Here the conclusions of the thesis are listed for each chapter as follows.

1. The formulation of the class of protocol synthesis problems (including ABP) in Chapter 3 makes it plausible to think that over ideal channels the problem of "protocol design" for communication processes with non-coobservable specifications can be reduced to the synthesis of communicating decentralized supervisors. This chapter presents solutions to a special class of problems where the processes need to communicate amongst themselves only for control, and a positive result is stated when channels are unreliable.

2. An important contribution of Chapter 3, which has the central role in this work, is the introduction of agent-wise labeling maps (ALMs) and

their proof of existence, which is essentially a beautiful result, enjoying importance, applicability, and simplicity. This result establishes the foundations for decentralized representation of the information structure of a DES, and in particular, a centralized supervisor.

3. Chapter 4 introduces the (distributed) SDES framework for synthesis of communicating decentralized supervisors out of a given centralized supervisor. DSDES framework, in which analysis and design are tractable and insightful, encompasses EFSM framework, as its special implementation-oriented case; hence the two enjoy their complementary facets. Also the two frameworks complement the behavioral study of DESs, which is the mainstream in DES study, providing a second viewpoint similar to state space analysis of control systems. The central tools employed by a DSDES are its ALM, which reflect the structure of the centralized controller in a distributed manner, and its updating and guard functions, which represent the observation and control-related information of the centralized supervisor. When recast as a PDS over a field, a DSDES becomes amenable to standard mathematical analysis tools, including computational algebraic machinery.

4. Within DSDES framework, communication among decentralized supervisors appears naturally as a means to help them reevaluate their guard and updating functions. In Chapter 5 the communication problem is formally defined with the help of communication-related events, which are in turn defined as maps from observed events to a subset of Boolean variables. The synthesis of communication becomes then equivalent to completely defining these maps. Each such synthesis is done in two levels corresponding to synthesis of information policies and synthesis of routing policies, respectively. The former, being studied in this thesis,

reveals the mutual informational dependency of decentralized supervisors on each other, regardless of the network topology and communication channels. An information policy can be implemented by itself in strongly connected networks whose communication channels are delay-free. However, when some mutual communication channels are missing or delayed, a routing policy should be designed, on top of an information policy, which implements the information exchange in the presence of physical (environmental) constraints. Therefore, design of information policies is the first step of a communication design, but may have to be followed by design of routing policies. To demonstrate the modeling power of the proposed framework and synthesis procedure, 6 information policies are derived and proved to guarantee the satisfaction of the given specifications in (distributed) DESs.

5. A second group of results of Chapter 5 are those which provide a finer partitioning of the class of communicating supervisors. This partitioning, being invoked by informational dependency of guard and updating functions of external variables, has consequences in designing information policies and fault recovery in the communicating network of supervisors and complexity of communication implementation.

6. A third group of results in Chapter 5 are on the communication-oriented state representation of centralized supervisors. This subject, which deserves a separate work of its own, has roots in realization theory and is of interest to control theorists, specially due to its important role in state space methods. The findings of this research opens a new window into it.

7. Motivated by the observation on ABP synthesis in Chapter 3, Chapter 6

takes the first steps into the automatic synthesis of communication protocols by modeling and synthesis of RMTP-like protocols, which are used in multicasting data over asymmetric networks. The theoretical results of Chapter 5 are applied to this study and proves useful in this regard.

To the best of author's knowledge, the proposed framework and procedure of this thesis for the synthesis of communicating supervisors is among the most general and flexible one for this purpose. The proved properties of the derived information policies in Chapter 5 are examples of the capabilities of the proposed approach, which is missing in other similar works. Central to the proposed frameworks and synthesis procedures is the notion of ALMs, which always exist and can be computed efficiently. Moreover, the results on communication partitioning and communication-oriented state representations open windows to interesting and important branches of research in communicating supervisors, which are unique to this work.

7.2 Future work

Details of some of the extensions of the results may be found throughout the thesis. From a general viewpoint, the research can be continued in certain directions as follows.

1. As shown in the proof of Theorem 3.2, an agent-wise labeling map can be defined using Latin hypercubes. However, often there is more than one arrangement of a Latin hypercube which can be used for this purpose. Furthermore, there might be other ways to compute an ALM. Each way of defining an ALM may enjoy specific properties which can be of interest in cases where more structure is imposed on the ALM.

2. Currently the symbolic representation of a DSDES is derived solely for

the purpose of computing communication rules. However, use of a symbolic representation is one of the ways to tackle computational complexity of the centralized supervisor synthesis (see Chapter 1). It would be desirable if relations between these two symbolic representations can be found and the two are integrated in one synthesis procedure in which a centralized supervisor and its decentralized implementation both are derived from a plant and a specification.

3. Study of routing policies is one of the key steps in applying the theoretical results to practical problems. First of all, this requires a study of network delay and its effect on supervisor synthesis. On the other hand, design of routing policies can potentially affect the definition of an associated already-designed information policy. These issues are within natural continuations of this research.

4. "Inference" and "time" are two important issues associated with communication. It is then important to encompass these issues in DSDES framework, too. Inference would help a supervisors use its information better and may even eliminate the need for some communication, thereby affecting the minimality of communication. Also time issues are important in real-time implementation communication policies and protocols.

5. The emphasis of this thesis is on the introduction and development of DSDES framework for the synthesis of communicating supervisors. As a result, solutions to the communication problem are derived mainly to show the capabilities of the framework and the synthesis procedure. However, it is also quite important to improve the computational aspects of the synthesis procedures, formulate quantitative measures (such as the one for communication size) in more details, and solve them using

discrete optimization techniques and tools. The quantitative analysis will be a complementary facet of the qualitative study of DESs.

6. Modeling, analysis, and synthesis of real-life problems directly depend on the availability of universal modeling schemes and a good realization theory. In author's viewpoint these two are bottlenecks of the application-side of supervisory control theory and every theory, which is built on it, including the proposed synthesis theory.

A major requirement of most of the above extensions is the availability of powerful software tools which can handle large plants and advanced algebraic computations.

Bibliography

[1] G. Barrett and S. Lafortune, "Decentralized supervisory control with communicating controllers," *IEEE Trans. Automat. Contr.*, vol. 45, pp. 1620–1638, Sept. 2000.

[2] B. Whetten and G. Taskale, "An overview of reliable multicast transport protocol II," *IEEE Network*, vol. 90, p. 3747, January-February 2000.

[3] W. M. Wonham. (2009) Supervisory control of discrete-event systems. URL: http://www.control.utoronto.ca/DES/. [Online]. Available: http://www.control.utoronto.ca/DES/

[4] T. Kropf, *Introduction to Formal Hardware Verification*, 1st ed. Springer Verlag, 1999.

[5] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *volume 999 of LNCS: Hybrid System II*, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. Springer-Verlag, 1995.

[6] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[7] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Information Sciences*, vol. 44, no. 2, pp. 173–198, 1988.

[8] ——, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE transactions on automatic control*, vol. 35, no. 12, pp. 1330–1337, December 1990.

[9] J. Tsitsiklis, "On the control of discrete-event dynamical systems," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 95–107, 1989.

[10] W. M. Wonham and P. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of Control, Signals, and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

[11] F. Lin and W. M. Wonham, "Decentralized supervisory control of discrete-event systems," *Information sciences*, vol. 44, no. 2, pp. 199–224, 1988.

[12] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1692–1708, Nov. 1992.

[13] R. Cieslak, C. Deslaux, A. S. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 249–260, Mar. 1988.

[14] K. Rudie and J. C. Willems, "The computational complexity of decentralized discrete-event control problems," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 1313–1319, July 1995.

[15] T.-S. Yoo and S. Lafortune, "A general architecture for decentralized supervisory control of discrete-event systems," *Discrete-Event Dynamic Systems: Theory and Applications*, vol. 12, no. 3, pp. 335–377, July 2002.

[16] S. Takai, R. Kumar, and T. Ushio, "Characterization of co-observable languages and formulas for their super/sublanguages," *IEEE Trans. Automat. Contr.*, vol. 50, pp. 434–447, Apr. 2005.

[17] H. S. Witsenhasusen, "Separation of estimation and control for discrete time systems," *Proc. IEEE*, vol. 59, pp. 1557–1566, Nov. 1971.

[18] P. Kozak and W. M. Wonham, "Fully decentralized solutions of supervisory control problems," *IEEE Trans. Automat. Contr.*, vol. 40, no. 12, pp. 2094–2097, Dec. 1995.

[19] J. H. van Schuppen, "Decentralized control with communication between controllers," in *Unsolved Problems in Mathematical Systems and Control Theory*, V. D. Blondel and A. Megretski, Eds. Princeton, USA: Princeton University Press, 2004.

[20] K. C. Wong and J. H. van Schuppen, "Decentralized supervisory control of discrete-event systems with communication," in *Proc. Workshop on Discrete-Event Systems (WODES'96)*, Edinburgh, UK.

[21] D. Teneketzis, "On information structures and nonsequential stochastic control," *CWI Quarterly*, vol. 10, no. 2, pp. 179–199, 1997.

[22] J. H. van Schuppen, "Decentralized supervisory control with information structures," in *Proc. Workshop on Discrete-Event Systems (WODES'98)*, Cagliari, Italy, Aug 1998, pp. 36–41.

[23] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*, 2nd ed. The MIT Press, 2003.

[24] S. L. Ricker and K. Rudie, "Incorporating communication and knowledge into decentralized discrete-event systems," in *Proc. IEEE Conference on*

Decision and Control (CDC'99), Phoenix, USA, Dec. 1999, pp. 1326–1332.

[25] ——, "Know means no: Incorporating knowledge into discrete-event control systems," *IEEE Trans. Automat. Contr.*, vol. 45, pp. 1656–2000, Sept. 2000.

[26] ——, "Knowledge is a terrible thing to waste: Using inference in discrete-event control problems," in *Proc. American Control Conference (ACC'03)*, Denver, USA, June 2003, pp. 2246–2251.

[27] K. Rudie, S. Lafortune, and F. Lin, "Minimal communication in a distributed discrete-event system," *IEEE Trans. Automat. Contr.*, vol. 48, pp. 957–975, June 2003.

[28] S. Tripakis, "Undecidable problems of decentralized observation and control," in *Proc. IEEE Conference on Decision and Control (CDC'01)*, ,, Dec. 2001.

[29] ——, "Decentralized control of discrete-event systems with bounded or unbounded delay communication," *IEEE Trans. Automat. Contr.*, vol. 49, pp. 1489–1501, Sept. 2004.

[30] A. Mazurkiewicz, "Introduction to trace theory," in *The book of traces*, V. Diekert and G. Rozenberg, Eds. New Jersey: World Scientific, 1995.

[31] S. Tripakis, "Decentralized observation problems," in *Proc. IEEE Conference on Decision and Control-European Control Conference (CDC-ECC'05)*, Seville, Spain, Dec. 2005, pp. 6–11.

[32] F. Lin, K. Rudie, and S. Lafortune, "Minimal communication for essential transitions in a distributed discrete-event system," *IEEE Trans. Automat. Contr.*, vol. 52, no. 8, pp. 1495–1502, Aug. 2007.

[33] W. Wang, S. Lafortune, and F. Lin, "Minimization of communication in distributed discrete-event systems," in *Proc. of the European Control Conference (ECC'07)*, Kos, Greece, July 2007, pp. 4960–4967.

[34] S. L. Ricker and J. H. van Schuppen, "Asynchronous communication in timed discrete-event systems," in *Proc. American Control Conference (ACC'01)*, Arlington, USA, June 2001, pp. 305–306.

[35] S.-J. Park and K.-H. Cho, "Delay-robust supervisory control of discrete-event systems with bounded communication delays," *IEEE Trans. Automat. Contr.*, vol. 51, no. 5, pp. 911–915, May 2006.

[36] ——, "Supervisory control of discrete-event systems with communication delays and partial observations," *Systems & Control Letters*, vol. 56, pp. 106–112, Feb. 2007.

[37] ——, "Decentralized supervisory control of discrete-event systems with communication delays based on conjunctive and permissive decision structures," *Automatica*, vol. 43, pp. 738–743, Mar. 2007.

[38] X. Guan and L. Holloway, "Distributed discrete event control structures with controller interactions," in *Proc. of the American Control Conference (ACC'95)*, 1995, pp. 3151–3156.

[39] J. H. Prossert, M. Kamt, and H. G. Kwatnyt, "Decision fusion and supervisor synthesis in decentralized discrete-event systems," in *Proc. of the American Control Conference (ACC'97)*, June 1997, pp. 2251–2255.

[40] T.-S. Yoo and S. Lafortune, "Decentralized supervisory control with conditional decisions: Supervisor existence," *IEEE Trans. Automat. Contr.*, vol. 49, no. 11, pp. 1886–1904, Nov. 2004.

[41] ——, "Decentralized supervisory control with conditional decisions: Supervisor realization," *IEEE Trans. Automat. Contr.*, vol. 50, no. 8, pp. 1205–1211, Aug. 2005.

[42] Y. Wang, T.-S. Yoo, and S. Lafortune, "Decentralized diagnosis of discrete event systems using unconditional and conditional decisions," in *Proc. of the IEEE Conference on Decision and Control and the European Control Conference (CDC-ECC'05)*, Seville, Spain, Dec. 2005, pp. 6298–6304.

[43] S. L. Ricker and K. Rudie, "Distributed knowledge for communication in decentralized discrete-event systems," in *Proc. IEEE Conference on Decision and Control (CDC'00)*, Sydney, Australia, Dec. 2000, pp. 9–15.

[44] ——, "Knowledge is a terrible thing to waste: Using inference in discrete-event control problems," *IEEE Trans. Automat. Contr.*, vol. 52, no. 3, pp. 428–441, Mar. 2007.

[45] W. Qiu, R. Kumar, and V. Chandra, "Decentralized control of discrete-event systems using prioritized composition with exclusion," in *Proc. of the American Control Conference (ACC'94)*, 2004, pp. 4483–4487.

[46] R. Kumar and S. Takai, "Inference-based ambiguity management in decentralized decision-making: Decentralize control of discrete-event systems," in *Proc. of the IEEE Conference on decison and Control-European Control Conference (CDC-ECC'05)*, Seville, Spain, Dec. 2005, pp. 3480–3485.

[47] A. Mannani, Y. Yang, and P. Gohari, "Distributed extended finite-state machines: Communication and control," in *Proc. of the IEEE 8th International Workshop on Discrete-Event Systems (WODES'06)*, Ann Arbor, USA, July 2006, pp. 161–167.

[48] Y. Yang, A. Mannani, and P. Gohari, "Implementation of supervisory control using extended finite-state machines," *Internationa Journal of Systems Science*, vol. 39, no. 12, pp. 1115–1125, Dec. 2008.

[49] Y. Yang and P. Gohari, "Embedded supervisory control of discrete-event systems," in *IEEE Conference on Automation Science and Engineering (ASE)*, August 2005, pp. 410–415.

[50] A. Mannani and P. Gohari, "Decentralized supervisory control of discrete-event systems over communication networks," *IEEE Trans. on Automat. Contr.*, vol. 53, no. 2, pp. 547–559, Mar. 2008.

[51] K.-T. Cheng and A. S. Krishnakumar, "Automatic generation of functional vectors using the extended finite state machine model," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1.1, no. 1, pp. 57–79, Jan. 1996.

[52] W. C. Lai, "Embedded software-based self-test for system-on-a-chip design," Ph.D. dissertation, University of California, Santa Barbara, Newark, USA, Mar. 2002.

[53] R. C.-Y. Huang and K.-T. Cheng, "A new extended finite state machine (efsm) model for rtl design verification," in *Proc. IEEE International High Level Design Validation and Test Workshop (HLDVT'98)*, Nov. 1998, p. 4753.

[54] J. S. Ostroff, *Temporal Logic for Real-Time Systems*, 1st ed. John Wiley and Sons Inc., 1989.

[55] Y.-L. Chen and F. Lin, "Modeling of discrete-event systems with finite state machines with parameters," in *Proc. IEEE Conference on Control Applications(CCA'00)*, Anchorage, USA, Sept. 2000, pp. 941–946.

[56] M. L. Borgne, A. Benveniste, and P. L. Guernic, "Polynomial dynamical systems over finite fields," in *Algebraic Computing in Control.* Heidelberg: Springer Berlin, 1991, vol. 165/1991, pp. 212–222.

[57] R. Germundsson and T. Glad, "A unified constructive study of linear, nonlinear and discrete event systems," Dept of EE. Linköping University, S-581 83 Linköping, Sweden, Tech. Rep. LiTH-ISY-R-1748, Feb. 1995. [Online]. Available: http://www.control.isy.liu.se/publications/doc?id=131

[58] J. Gunnarsson, "Symbolic methods and tools for discrete event dynamic systems," Ph.D. dissertation, Univ. of Linköping, S-581 83 Linköping, Sweden.

[59] G. Hoffmann and H. Wong-Toi, "Symbolic synthesis of supervisory controllers," in *Proc. of the American Control Conference (ACC'92)*, Chicago, USA, June 1992, pp. 2789–2793.

[60] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. IEEE 8th International Workshop on Discrete Event Systems (WODES'06)*, Ann Arbor, MI, USA, July 2006, pp. 384 – 385.

[61] Z. Zhang and W. M. Wonham, "Stct: an efficient algorithm for supervisory control design," in *Proc. of the Symposium on Supervisory Control of Discrete Event Systems (SCODES'01)*, Paris, France, July 2001.

[62] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Trans. on Automat. Contr.*, vol. 51, no. 5, pp. 782–793, May 2006.

[63] A. Overkamp and J. H. van Schuppen, "Maximal solutions in decentralized supervisory control," *SIAM Journal on Control and Optimization*, vol. 39, no. 2, pp. 492–511, 2000.

[64] A. Mannani and P. Gohari, "Decentralized supervisory control of discrete-event systems: A game theory perspective," in *Proc. of European Control Conference (ECC'07)*, Kos, Greece, July 2007, pp. 4968–4975.

[65] E. Kushilevitz and N. Nisan, *Communication Complexity*, 1st ed. Cambridge University Press, 1997.

[66] G. Fertin and A. Raspaud, "A survey on kn'odel graphs," *Discrete Applied Mathematics*, vol. 137, no. 2, pp. 173–195, Mar. 2004.

[67] R. L. Probert and K. Saleh, "Synthesis of communication protocols: Survey and assessment," *IEEE Transactions on Computers*, vol. 40, pp. 468–476, Apr. 1991.

[68] G. Holzmann, "Protocol design: Redefining the state of the art," *IEEE Software*, vol. 9, pp. 17–22, Jan. 1992.

[69] K. Rudie and W. M. Wonham, "Supervisory control of communicating processes," in *Protocol Specification, Testing, and Verification*, L. Logrippo, R. L. Probert, and H. Ural, Eds. B. V., North Holland: Elsevier Science Publishers, 1990.

[70] P. Gohari, "Alternating bit protocol: A supervisory control perspective," in *Decentralized Discrete Event Systems: Structure, Communication and Control*, B. I. R. Station, Ed., May 2004, pp. 975–980.

[71] P. J. Ramadge and W. M. Wonham, "The control of discrete-event systems," *Proc.IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[72] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation*, 2nd ed. Addison Wesley, 2001.

[73] M. M. Mano, *Digital design*, 3rd ed. Prentice-Hall, 2002.

[74] A. Mannani and P. Gohari, "A framework for modeling communication among decentralized supervisors for discrete-event systems," Control and Robotics Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, Technical Report CRG-TR0407, Apr. 2007. [Online]. Available: http://users.encs.concordia.ca/ crg/CRG.html

[75] W. C. Lynch, "Computer system: Reliable full-duplex file tranmisstion over half-duplex telephone line," *Communications of the ACM*, vol. 11, no. 6, pp. 407–410, 1968.

[76] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Communications of the ACM*, vol. 12, pp. 260–261, 1969.

[77] J. Denes and A. D. Keedwell, *Latin suares and their applications*, 1st ed. Academic Press, 1974.

[78] C. J. Colbourn and J. H. Dinitz, Eds., *The CRC handbook of combinatorial designs*, 1st ed. New York, USA: CRC Press, 1996.

[79] A. Mannani and P. Gohari, "Decentralized supervisory control of discrete-event systems over communication networks," Control and Robotics Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, Technical Report CRG-TR0207, Feb. 2007. [Online]. Available: http://users.encs.concordia.ca/ crg/CRG.html

[80] R. Germundsson, "Symbolic systems: Theory, computation, and applications," Ph.D. dissertation, Univ. of Linköping, S-581 83 Linköping, Sweden, Sept. 1995.

[81] R. Lidl and H. Niederreiter, *Finite Fields*, 1st ed. Cambridge University Press, 1997.

[82] R. Germundsson, "Basic results on ideals and varieties in finite fields," 1991. [Online]. Available: citeseer.ist.psu.edu/germundsson91basic.html

[83] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, 3rd ed. Springer, 2007.

[84] S. Tripakis, "Undecidable problems of decentralized observation and control on regular languages," *Information Processing Letters*, vol. 90, p. 2128, 2004.

[85] M. Sipser, *Introduction to the Theory of Computation*, 2nd ed. PWS Publishing Company, february 2005.

[86] C. Linder, private communication, November 2007.

[87] H. Zimmermann, "OSI reference model- the ISO model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr. 1980.

[88] Wikipedia. (2009, Apr.) Multicast. [Online]. Available: http://en.wikipedia.org

[89] B. Whetten, M. Basavaiah, S. Paul, T. Montgomery, N. Rastogi, J. Conlan, and T. Yeh. (1998, Apr.) The RMTP-II protocol. Internet draft; work in progress. [Online]. Available: http://www.talarian.com/rmtp-ii

[90] W. M. Wonham. (2008) XPTCT website. [Online]. Available: http://www.control.utoronto.ca/cgi-bin/dlxptct.cgi

[91] ——, private communication, August 2009.

Appendix A

Missing proofs of the claims of Chapter 4

Proof of Lemma 4.1

First note that the function $\mathcal{A}$ is well-defined. Proof is done by induction on the length of strings.

- Base:

$$\begin{aligned} \mathcal{A}(\epsilon, 0) = \ell(r) &\iff 0 = \ell(r) \\ &\iff r = r_0 && \text{[Item 1, Defn. 4.3]} \\ &\iff r = \xi(r_0, \epsilon) \end{aligned}$$

- Inductive step: Assume that $s\sigma \in L(\mathbf{S})$. Then $s \in L(\mathbf{S})$, i.e. there exists $r \in R$ such that $r = \xi(r_0, s)$. It follows from the induction assumption that $\mathcal{A}(s, 0) = \ell(r) \iff r = \xi(r_0, s)$. Let $r' = \xi(r, \sigma)$. We have:

$$
\begin{aligned}
A(s\sigma, 0) = \ell(r') &\iff \mathcal{A}(\sigma, \mathcal{A}(s, 0)) = \ell(r') && \text{[Defn. of } \mathcal{A}\text{]} \\
&\iff \mathcal{A}(\sigma, \ell(r)) = \ell(r') && \text{[Induction assumption]} \\
&\iff r' = \xi(r, \sigma) && \text{[Eq. (4.1), Item 2 of Defn. 4.3]} \\
&\iff r' = \xi(r_0, s\sigma). && \blacksquare
\end{aligned}
$$

Proof of Proposition 4.2

In the nontrivial case where E is nonempty the proof is by induction on the length of strings, where the base is trivial since $\epsilon \in \overline{E}$ and $\epsilon \in L(\mathcal{D})$. For the inductive step let $\sigma \in \Sigma$, $s \in \Sigma^*$ and $r = \xi(r_0, s)$. By the induction assumption,

$$
s \in \overline{E} \iff s \in L(\mathcal{D}). \tag{A.1}
$$

Then we have

$$
\begin{aligned}
s\sigma \in \overline{E} &\iff s \in \overline{E} \wedge s\sigma \in \overline{L} \wedge [\exists r \in R.\ r = \xi(r_0, s) \wedge \xi(r, \sigma)!] \\
&\iff s \in \overline{E} \wedge s\sigma \in \overline{L} \wedge [\exists r \in R.\ \mathcal{A}(s, 0) = \ell(r) \wedge \xi(r, \sigma)!] && \text{[Lem. 4.1]} \\
&\iff s \in L(\mathcal{D}) \wedge s\sigma \in \overline{L} \wedge \ell(r) \in \mathcal{G}(\sigma) && \text{[Eq. (A.1), Eq. (4.2)]} \\
&\iff s\sigma \in L(\mathcal{D}). && \text{[Defn. 4.2]}
\end{aligned}
$$

This proves that $L(\mathcal{D}) = \overline{E}$. For the next part by Definition 4.2 and the fact that E is $L_m(\mathbf{G})$-closed, we have $L_m(\mathcal{D}) = L(\mathcal{D}) \cap L_m(\mathbf{G}) = \overline{E} \cap L_m(\mathbf{G}) = E$. $\blacksquare$

Proof of Lemma 4.3

Proof is done by induction on the length of strings.

- Base:

$$\begin{aligned}
\mathcal{A}(\epsilon, \mathbf{0}) \in \ell(r) &\iff \mathbf{0} \in \ell(r) && \text{[Eq. (4.4)]} \\
&\iff r = r_0 && \text{[Item 1, Defn. 4.6]} \\
&\iff r = \xi(r_0, \epsilon)
\end{aligned}$$

- Inductive step: Assume that $s\sigma \in L(\mathbf{S})$. Then $s \in L(\mathbf{S})$, i.e. there exists $r \in R$ such that

  $r = \xi(r_0, s)$. It follows from the induction assumption that $\mathcal{A}(s, \mathbf{0}) \in \ell(r) \iff r = \xi(r_0, s)$. Let $r' = \xi(r, \sigma)$. We have:

$$\begin{aligned}
A(s\sigma, \mathbf{0}) \in \ell(r') &\iff \Big(\mathcal{A}_i(\sigma, \mathcal{A}(s, \mathbf{0}))\Big)_{i \in I} \in \ell(r') && \text{[Eq. (4.4)]} \\
&\iff \Big(\pi_i \mu(\sigma, \mathcal{A}(s, \mathbf{0}))\Big)_{i \in I} \in \ell(r') && \text{[Eq. (4.5)]} \\
&\iff \mu(\sigma, \mathcal{A}(s, \mathbf{0})) \in \ell(r') \\
&\iff r' = \xi(r, \sigma) && \text{[Lem. 4.2]} \\
&\iff r' = \xi(r_0, s\sigma) && \blacksquare
\end{aligned}$$

Proof of Proposition 4.3

In the nontrivial case where E is nonempty the proof is by induction on the length of strings, where the base is trivial since $\epsilon \in \overline{E}$ and $\epsilon \in L(\mathcal{D})$. For the inductive step let $\sigma \in \Sigma$, $s \in \Sigma^*$ and $r = \xi(r_0, s)$. By the induction assumption $s \in \overline{E} \iff s \in L(\mathcal{D})$. We have:

$$\begin{aligned}
s\sigma \in \overline{E} &\iff s \in \overline{E} \wedge s\sigma \in \overline{L} \wedge [\exists r \in R.\ r = \xi(r_0, s) \wedge \xi(r, \sigma)!] \\
&\iff s \in \overline{E} \wedge s\sigma \in \overline{L} \wedge [\exists r \in R.\ \mathcal{A}(s, \mathbf{0}) \in \ell(r) \wedge \xi(r, \sigma)!] && \text{[Lem. 4.3]} \\
&\iff s \in L(\mathcal{D}) \wedge s\sigma \in \overline{L} \wedge [\exists r \in R.\ \mathcal{A}(s, \mathbf{0}) \in \ell(r) \wedge \xi(r, \sigma)!] && \text{[Induction}
\end{aligned}$$

Assum.]

$$\iff s \in \cap_{i\in I} L(\mathcal{D}_i) \wedge s\sigma \in \overline{L} \wedge \mathcal{A}(s,\mathbf{0}) \in \bigcap_{i\in I} \mathcal{G}_i(\sigma) \qquad \text{[Defn. 4.5]}$$

$$\iff \forall i \in I.\ s \in L(\mathcal{D}_i) \wedge s\sigma \in \overline{L} \wedge \mathcal{A}(s,\mathbf{0}) \in \mathcal{G}_i(\sigma)$$

$$\iff \forall i \in I.\ s\sigma \in L(\mathcal{D}_i) \qquad \text{[Defn. 4.4]}$$

$$\iff s\sigma \in \cap_{i\in I} L(\mathcal{D}_i)$$

$$\iff s\sigma \in L(\mathcal{D}). \qquad \text{[Defn. 4.5]}$$

This proves that $L(\mathcal{D}) = \overline{E}$. For the next part we have:

$$L_m(\mathcal{D}) = \bigcap_{i\in I} L_m(\mathcal{D}_i) \qquad \text{[Defn. 4.5]}$$

$$= \bigcap_{i\in I} \big(L(\mathcal{D}_i) \cap L_m(\mathbf{G})\big) \qquad \text{[Defn. 4.2]}$$

$$= \big(\bigcap_{i\in I} L(\mathcal{D}_i)\big) \cap L_m(\mathbf{G}) = \overline{E} \cap L_m(\mathbf{G}) \qquad \text{[Part 1]}$$

which, by $L_m(\mathbf{G})$-closure of E, is equal to E. ■

Proof of Proposition 4.4

First we show that within the set of state labels, $\mathfrak{g}_i^\sigma(.)$ is indeed the characteristic equation of the guard function $\mathcal{G}_i(\sigma)$ and hence correctly represents it. From (4.6)

$$\forall \sigma \in \Sigma. \qquad \mathcal{G}_i(\sigma) = \begin{cases} \{\ell(r) \mid r \in R \wedge \xi(r,\sigma)!\}; & \text{if } \sigma \in \Sigma_{c,i}, \\ \cup_{r\in R} \ell(r); & \text{if } \sigma \in \Sigma_{uc,i}. \end{cases},$$

i.e. $\mathcal{G}_i(.)$ does not consider a vector $\mathbf{v} \in \mathbb{F}_p^n \setminus \bigcup_{r\in R} \ell(r)$. On the other hand, by (4.10) we have

$$\forall \mathbf{v} \in \bigcup_{r\in R} \ell(r). \quad \mathfrak{g}_i^\sigma(\mathbf{v}) = 1 \iff \mathbf{v} \in \mathcal{G}_i(\sigma). \qquad \text{(A.2)}$$

This proves the claim. Next, recall the definition of updating functions in

(4.5), i.e.

$$\forall r, r' \in R, \forall \sigma \in \Sigma, \forall \mathbf{v} \in \mathbb{N}^n.\ r' = \xi(r,\sigma) \wedge \mathbf{v} \in \ell(r) \Longrightarrow \mathcal{A}_i(\sigma,\mathbf{v}) = \pi_i(\mu(\sigma,\mathbf{v})).$$

We show that $\mathfrak{a}_i^\sigma(\mathbf{v})$ is equal to $\mathcal{A}_i(\sigma,\mathbf{v})$ when the latter is defined. First we show that the premise of (4.5), which states the condition under which $\mathcal{A}_i(.,.)$ is defined, is equivalent to $\mathbf{v} \in \mathcal{G}_i(\sigma)$ as stated in (4.11). To this end, observe that

$$\begin{aligned}
&\forall r, r' \in R, \forall \sigma \in \Sigma, \forall \mathbf{v} \in \mathbb{N}^n.\ r' = \xi(r,\sigma) \ \wedge\ \mathbf{v} \in \ell(r) \\
&\quad \Longrightarrow\ \left[[\sigma \in \Sigma_{c,i} \Longrightarrow \mathbf{v} \in \{\ell(r) \mid r \in R \wedge \xi(r,\sigma)!\}] \ \wedge\ [\sigma \in \Sigma_{uc,i} \Longrightarrow \mathbf{v} \in \bigcup_{r\in R} \ell(r)]\right] && \text{[Eq. (4.6)]} \\
&\quad \Longrightarrow\ \mathbf{v} \in \mathcal{G}_i(\sigma) && \text{[Eq. (4.6)]}
\end{aligned}$$

Conversely, by the definition of $\mathcal{G}_i(\sigma)$ in (4.6), we have

$$\begin{aligned}
\forall \sigma \in \Sigma, \forall \mathbf{v} \in \mathbb{N}^n.\ \mathbf{v} \in \mathcal{G}_i(\sigma) &\Longrightarrow \exists r, r' \in R.\ r' = \xi(r,\sigma) \wedge \mathbf{v} \in \ell(r) \\
&\Longrightarrow \mathcal{A}_i(\sigma,\mathbf{v}) = \pi_i(\mu(\sigma,\mathbf{v})) && \text{[Eq. (4.5)]}
\end{aligned}$$

Hence $\mathbf{v} \in \mathcal{G}_i(\sigma)$, as stated in (4.11), determines when $\mathcal{A}_i(.,.)$ is defined. Accordingly $\mathfrak{a}_i^\sigma(\mathbf{v}) = \mathcal{A}_i(\sigma,\mathbf{v})$ in (4.11) correctly replaces the updating functions, when they are defined. Notice that when such an updating function for σ is not defined, i.e when $\mathbf{v} \in \mathbb{F}_p^n \setminus \mathcal{G}_i(\sigma)$, the polynomial $\mathfrak{a}_i^\sigma(\mathbf{v})$ may be assigned arbitrarily because such a σ is disabled by the guard function(s). ■

Proof of Proposition 4.6

For PDS (4.14) Proposition 4.4 insures that $L(\mathcal{D}) = \overline{E}$ and $L_m(\mathcal{D}) = E$. It is then enough to show that the PDS equations are held by $(\mathbf{M}_i)_{i\in I}$ for $\mathbf{x} \in \mathbb{F}_p^n$. To this end, observe that since $(\mathbf{M}_i)_{i\in I}$ implements the PDS, Definition 4.10 insures that, first, for every $i, j \in I, i \neq j$ and $k \in J$, $x_{ij}^k = x_{jj}^k \in X_{jj}$.

Then, by virtue of the second and the third conjuncts in Definition 4.10, actions and guards evaluate equivalently, which proves the claim. ■

Appendix B

Missing proofs of the claims of Chapter 5

Proof of Proposition 5.1

We first show that communication policy 2 leads to correct reevaluation of guard and updating functions of the PDS (4.14), which then by Problem 5.1 and Definition 4.7 proves the claim for this policy. This is done by induction on the centralized supervisors' states seen along the system's evolution. The assumptions of strong connectedness of the network, instantaneous communication, and lossless channels insure that every two supervisors have a direct link between them, through which communication is done with no delay and no losses.

Base: The centralized supervisor starts from state r_0 and since no event has occurred yet, by Definitions 4.6 and 4.7, the label assigned to r_0 is $\underline{0}$. On the other hand, by Definition 4.9, all variables in X are initialized to zero, i.e. for $i \in I$ every $x \in X_{ii}$ and its copies are zero and therefore each supervisor estimates the state of the system to be $\mathbf{x} = (x_1, \cdots, x_n) = \underline{0}$ and evaluate its guards and actions and, respectively, guard and updating functions correctly.

Inductive step: Let $i, j \in I$, $i \neq j$, and the PDS representation (4.14) be at state $\mathbf{x} \in \mathbb{F}_p^n$, where $\mathbf{x} = (x_1, \cdots, x_n)$, and $x_i = (x_{ii}^h, \cdots, x_{ii}^1)$. The induction assumption states that at state $\mathbf{x}$, every $\mathbf{S}_i$ evaluates its guard functions correctly and if an event $\sigma \in \Sigma$ occurs, every $\mathbf{S}_i$ correctly reevaluate its updating functions. By Definition 4.10 this is equivalent to saying that in the EFSM implementation, each $\mathbf{S}_i$ correctly evaluates its guards and, upon the occurrence of σ reevaluates its actions correctly. Let $\hat{\mathbf{x}} \in \mathbb{F}_p^n$ be the next state reached by the PDS through σ, where $\hat{\mathbf{x}} = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$ and $\hat{x}_i = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$. We show that by using communication policy 2, the PDS correctly reevaluates its updating functions and guard formulas at $\hat{\mathbf{x}}$, too. Towards this end, we distinguish two possible cases.

1. If $\sigma \in \Sigma_{uo}$, no $\mathbf{S}_i$ observes σ and therefore no $x \in X_{ii}$ changes. Thus for every i, $\hat{x}_i = x_i$ and the PDS would remain in $\mathbf{x}$. In other words, no private variable changes and each $\mathbf{S}_i$ still has the latest copies of others' private variables, which it had before. Therefore, by induction assumption, it can correctly evaluates its guards and, upon the occurrence of a next event, its actions, which then by Definition 4.10, leads to correct evaluation of guard and updating functions. This justifies the fact that, by Definition 5.3, we have $\mathscr{I}_{ji}(\sigma) = \emptyset$.

2. If $\sigma \in \Sigma_o$, it is observed by every $\mathbf{S}_j$ for which $j \in I_o(\sigma)$. By communication policy 2, this may fire a set of communication events $\mathscr{I}_{ji}(\sigma)$ such that the following hold.

$$\forall j \in I_o(\sigma).\ \ \hat{x}_j := \mathfrak{a}_j^\sigma(\mathbf{x}) \implies \hat{x}_j \neq x_j$$

[Lemma 4.2, Eq. (4.5), and Prop. 4.4]

$$\implies \left[\exists k \in J.\ \ \hat{x}_{jj}^k := a_j(x_{jj}^k, \sigma)(X_j) \implies \hat{x}_{jj}^k \neq x_{jj}^k\right] \qquad \text{[Defn. 4.10]}$$

$$\Longrightarrow \big[\forall i \in I, i \neq j, \forall y \in X_{ii}, \forall \sigma' \in \Sigma.\ \ x_{ij}^k \in \arg(a_i(y,\sigma'))$$

$$\vee\, x_{ij}^k \in \arg(g_i(\sigma')) \implies \hat{x}_{jj}^k \in \mathscr{I}_{ji}(\sigma)\big] \qquad \text{[Defn. 5.3]}$$

$$\Longrightarrow \big[x_{ij}^k := \hat{x}_{jj}^k\big] \qquad \text{[Eq. (5.1)]}$$

$$\Longrightarrow \big[\forall k \in J.\ \ (\hat{x}_{jj}^k \neq x_{jj}^k \implies \hat{x}_{ij}^k = \hat{x}_{jj}^k)$$

$$\wedge\ (\hat{x}_{jj}^k = x_{jj}^k \implies \hat{x}_{ij}^k = x_{ij}^k = x_{jj}^k)\big]$$

$$\Longrightarrow \big[\forall k \in J.\ \ \hat{x}_{ij}^k = \hat{x}_{jj}^k\big]$$

$$\Longrightarrow \big[\forall \sigma'' \in \Sigma_{o,i}, \forall z \in X_{ii}.$$

$$a_i(z,\sigma'')(\hat{X}_i) \text{ and } g_i(\sigma'')|_{\hat{X}_i} \text{ are reevaluated correctly.}\big]$$

$$\Longrightarrow \big[\forall \sigma'' \in \Sigma, \forall z \in X_{ii}.$$

$$a_i(z,\sigma'')(\hat{X}_i) \text{ and } g_i(\sigma'')|_{\hat{X}_i} \text{ are reevaluated correctly.}\big] \quad \text{[Rem. 4.3]}$$

$$\Longrightarrow \big[\forall \sigma'' \in \Sigma.\ \ \mathfrak{a}_i^{\sigma''}(\hat{\mathbf{x}}) \text{ and } \mathfrak{g}_i^{\sigma''}(\hat{\mathbf{x}}) \text{ are reevaluated correctly.}\big]$$

[Defn. 4.10]

In other words, every $\mathbf{S}_i$ receives the latest copy of all the external variables it requires for reevaluating its updating and guard functions at state $\hat{x}$. This completes the proof for policy 2. Since policy 1 requires the exchange of the same changed variables in policy 2 besides the unchanged variables of $\mathbf{S}_j$, the proof works out for policy 1, too. ■

Proof of Proposition 5.2:

We show that communication policy 3 leads to correct reevaluation of guard and updating functions of the PDS (4.14), which then by Problem 5.1 and Definition 4.7 proves the claim for this policy. This is done by induction on the centralized supervisors' states seen along the system's evolution. The

assumptions of strong connectedness of the network, instantaneous communication, and lossless channels insure that every two supervisors have a direct link between them, through which communication is done with no delay and no losses.

Base: The centralized supervisor starts from state r_0 and since no event has occurred yet, by Definitions 4.6 and 4.7, the label assigned to r_0 is $\mathbf{0}$. On the other hand, by Definition 4.9, all variables in X are initialized to zero, i.e. for $i \in I$ every $x \in X_{ii}$ and its copies are zero and therefore each supervisor estimates the state of the system to be $\mathbf{x} = (x_1, \cdots, x_n) = \mathbf{0}$ and evaluate its guards and actions and, respectively, guard and updating functions correctly.

Inductive step: Let $i, j \in I$, $i \neq j$, and the PDS representation (4.14) be at state $\mathbf{x} \in \mathbb{F}_p^n$, where $\mathbf{x} = (x_1, \cdots, x_n)$, and $x_i = (x_{ii}^h, \cdots, x_{ii}^1)$. The induction assumption states that at state $\mathbf{x}$, every $\mathbf{S}_i$ evaluates its guard functions correctly and if an event $\sigma \in \Sigma$ occurs, every $\mathbf{S}_i$ correctly reevaluates its updating functions. By Definition 4.10 this is equivalent to saying that in the EFSM implementation, each $\mathbf{S}_i$ correctly evaluates its guards and, upon the occurrence of σ, reevaluates its actions correctly. Let $\hat{\mathbf{x}} \in \mathbb{F}_p^n$ be the next state reached by the PDS through σ, where $\hat{\mathbf{x}} = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$ and $\hat{x}_i = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$. We show that by using communication policy 3, the PDS correctly reevaluates its updating functions and guard formulas at $\hat{\mathbf{x}}$, too. Towards this end, we distinguish two possible cases.

1. If $\sigma \in \Sigma_{uo}$, no $\mathbf{S}_i$ observes σ and therefore no $x \in X_{ii}$ changes. Thus for every i, $\hat{x}_i = x_i$ and the PDS would remain in $\mathbf{x}$. In other words, no private variable changes and each $\mathbf{S}_i$ still has the latest copies of others' private variables, which it had before. Therefore, by induction assumption, it can correctly evaluates its guards and, upon the occurrence of a next event, its actions, which then by Definition 4.10, leads to correct

evaluation of guard and updating functions. This justifies the fact that, by Definition 5.6, we have $\mathscr{I}_{ji}(\sigma) = \emptyset$.

2. If $\sigma \in \Sigma_o$, it is observed by every $\mathbf{S}_j$ for which $j \in I_o(\sigma)$. By communication policy 3, this may fire a set of communication events $\mathscr{I}_{ji}(\sigma)$ such that the following hold.

$$\forall j \in I_o(\sigma).\ \hat{x}_j := \mathfrak{a}_j^{\sigma}(\mathbf{x}) \implies \hat{x}_j \neq x_j$$

[Lemma 4.2, Eq. (4.5), and Prop. 4.4]

$$\implies \left[\exists k \in J.\ \hat{x}_{jj}^k := a_j(x_{jj}^k, \sigma)(X_j) \implies \hat{x}_{jj}^k \neq x_{jj}^k\right] \qquad \text{[Defn. 4.10]}$$

$$\implies \Big[\forall i \in I, i \neq j, \forall y \in X_{ii}, \forall \sigma' \in \Sigma.$$

$$\left(x_{ij}^k \in \arg(a_i(y, \sigma')) \wedge [\hat{x}_j]_{\mathfrak{a}_i^{\sigma'}, j} \neq [x_j]_{\mathfrak{a}_i^{\sigma'}, j}\right)$$

$$\vee \left(x_{ij}^k \in \arg(g_i(\sigma')) \wedge\ [\hat{x}_j]_{\mathfrak{g}_i^{\sigma'}, j} \neq [x_j]_{\mathfrak{g}_i^{\sigma'}, j}\right) \implies \hat{x}_{jj}^k \in \mathscr{I}_{ji}(\sigma)\Big]$$

[Defn. 5.6]

$$\implies \left[\hat{x}_{ij}^k := \hat{x}_{jj}^k\right] \qquad \text{[Eq. (5.1)]}$$

$$\implies \Big[\forall k \in J.\ \left(\hat{x}_{jj}^k = x_{jj}^k \implies \hat{x}_{ij}^k = x_{ij}^k = x_{jj}^k\right)$$

$$\wedge \Big(\left[\hat{x}_{jj}^k \neq x_{jj}^k \wedge \left([\hat{x}_j]_{\mathfrak{a}_i^{\sigma'}, j} \neq [x_j]_{\mathfrak{a}_i^{\sigma'}, j} \vee [\hat{x}_j]_{\mathfrak{g}_i^{\sigma'}, j} \neq [x_j]_{\mathfrak{g}_i^{\sigma'}, j}\right)\right]$$

$$\implies \hat{x}_{ij}^k = \hat{x}_{jj}^k\Big)$$

$$\wedge \Big(\left[\hat{x}_{jj}^k \neq x_{jj}^k \wedge [\hat{x}_j]_{\mathfrak{a}_i^{\sigma'}, j} = [x_j]_{\mathfrak{a}_i^{\sigma'}, j} \wedge [\hat{x}_j]_{\mathfrak{g}_i^{\sigma'}, j} = [x_j]_{\mathfrak{g}_i^{\sigma'}, j}\right]$$

$$\implies \hat{x}_{ij}^k = x_{ij}^k \neq \hat{x}_{jj}^k\Big)\Big]$$

$$\implies \Big[\forall k \in J.\ \left(\hat{x}_{jj}^k = x_{jj}^k \implies \hat{x}_{ij}^k = x_{ij}^k = x_{jj}^k\right)$$

$$\wedge \Big(\left[\hat{x}_{jj}^k \neq x_{jj}^k \wedge \left(\exists \mathbf{x}_{-j} \in \mathbb{F}_p^{n-1}.\ \mathfrak{a}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) \neq \mathfrak{a}_i^{\sigma'}(x_j, \mathbf{x}_{-j})\right.\right.$$

$$\left.\left. \vee\ \mathfrak{g}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) \neq \mathfrak{g}_i^{\sigma'}(x_j, \mathbf{x}_{-j}))\right] \implies \hat{x}_{ij}^k = \hat{x}_{jj}^k\Big)$$

$$\wedge \Big(\left[\hat{x}_{jj}^k \neq x_{jj}^k \wedge \left(\forall \mathbf{x}_{-j} \in \mathbb{F}_p^{n-1}.\ \mathfrak{a}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) = \mathfrak{a}_i^{\sigma'}(x_j, \mathbf{x}_{-j})\right.\right.$$

$$\left.\left. \wedge\ \mathfrak{g}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) = \mathfrak{g}_i^{\sigma'}(x_j, \mathbf{x}_{-j}))\right]$$

$$\implies \hat{x}_{ij}^k = x_{ij}^k \neq \hat{x}_{jj}^k\Big)\Big] \qquad \text{[Defn. 5.5]}$$

$$
\begin{aligned}
\Longrightarrow\ & \big[\forall k \in J.\ \ (\hat{x}^k_{jj} = x^k_{jj} \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} = x^k_{jj}) \\
& \wedge \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\exists X_{i-j} \in \mathbb{B}^{(n-1)h}, k' \in J. \\
& \quad a_i(x^{k'}_{ii}, \sigma')((\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j}) \\
& \qquad\qquad \neq a_i(x^{k'}_{ii}, \sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}) \\
& \quad \vee g_i(\sigma')(\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j}) \\
& \qquad\qquad \neq g_i(\sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}))] \\
& \qquad\qquad\qquad\qquad \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj}\big) \\
& \wedge \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\forall X_{i-j} \in \mathbb{B}^{(n-1)h}, \forall k' \in J. \\
& \quad a_i(x^{k'}_{ii}, \sigma')((\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j}) \\
& \qquad\qquad = a_i(x^{k'}_{ii}, \sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}) \\
& \quad \wedge g_i(\sigma')(\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j}) \\
& \qquad\qquad = g_i(\sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}))] \\
& \qquad\qquad\qquad\qquad \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj})\ \ \big] \qquad \text{[Lem. 5.4]}
\end{aligned}
$$

$$
\begin{aligned}
\Longrightarrow\ & \big[\forall i \in I, \forall \sigma'' \in \Sigma, \forall z \in X_{ii}. \\
& \qquad a_i(z, \sigma'')(\hat{X}_i) \text{ and } g_i(\sigma'')|_{\hat{X}_i} \text{ are reevaluated correctly.}\big]
\end{aligned}
$$

$$
\begin{aligned}
\Longrightarrow\ & \big[\forall i \in I, \forall \sigma'' \in \Sigma. \\
& \qquad \mathfrak{a}^{\sigma''}_i(\hat{\mathbf{x}}) \text{ and } \mathfrak{g}^{\sigma''}_i(\hat{\mathbf{x}}) \text{ are reevaluated correctly.}\big] \qquad \text{[Defn. 4.10]}
\end{aligned}
$$

In other words, every $\mathbf{S}_i$ receives the latest copy of those external variables which are required to reevaluate its updating and guard functions at state $\hat{x}$. This completes the inductive step of the proof, hence proving the claim. ■

Proof of Lemma 5.5:

Comparing Definitions 5.4 and 5.6 to each other implies the following.

$$
\begin{aligned}
{}^2\mathscr{I}_{ji}(\sigma) \setminus {}^3\mathscr{I}_{ji}(\sigma) \;&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \\
&\wedge \left[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\; \left(x^k_{ij} \in \arg(a_i(x', \sigma'))\right) \vee \left(x^k_{ij} \in \arg(g_i(\sigma'))\right)\right]\} \\
&\setminus \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \\
&\wedge \left[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\; \left(x^k_{ij} \in \arg(a_i(x', \sigma')) \wedge [\hat{x}_j]_{a_i^{\sigma'},j} \neq [x_j]_{a_i^{\sigma'},j}\right)\right. \\
&\left. \vee \left(x^k_{ij} \in \arg(g_i(\sigma')) \wedge [\hat{x}_j]_{g_i^{\sigma'},j} \neq [x_j]_{g_i^{\sigma'},j}\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \\
&\wedge \neg\left[\exists \sigma' \in \Sigma_i, \exists x' \in X_{ii}.\; \left(x^k_{ij} \in \arg(a_i(x', \sigma')) \wedge [\hat{x}_j]_{a_i^{\sigma'},j} \neq [x_j]_{a_i^{\sigma'},j}\right)\right. \\
&\left. \vee \left(x^k_{ij} \in \arg(g_i(\sigma')) \wedge [\hat{x}_j]_{g_i^{\sigma'},j} \neq [x_j]_{g_i^{\sigma'},j}\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \\
&\wedge \left[\forall \sigma' \in \Sigma_i, \forall x' \in X_{ii}.\; \neg\left(x^k_{ij} \in \arg(a_i(x', \sigma')) \wedge [\hat{x}_j]_{a_i^{\sigma'},j} \neq [x_j]_{a_i^{\sigma'},j}\right)\right. \\
&\left. \wedge \neg\left(x^k_{ij} \in \arg(g_i(\sigma')) \wedge [\hat{x}_j]_{g_i^{\sigma'},j} \neq [x_j]_{g_i^{\sigma'},j}\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \\
&\wedge \left[\forall \sigma' \in \Sigma_i, \forall x' \in X_{ii}.\; \left(x^k_{ij} \in \arg(a_i(x', \sigma')) \Longrightarrow [\hat{x}_j]_{a_i^{\sigma'},j} = [x_j]_{a_i^{\sigma'},j}\right)\right. \\
&\left. \wedge \left(x^k_{ij} \in \arg(g_i(\sigma')) \Longrightarrow [\hat{x}_j]_{g_i^{\sigma'},j} = [x_j]_{g_i^{\sigma'},j}\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \;\wedge \\
&\left[\forall \sigma' \in \Sigma_i.\; \left([\hat{x}_j]_{a_i^{\sigma'},j} = [x_j]_{a_i^{\sigma'},j}\right) \wedge \left([\hat{x}_j]_{g_i^{\sigma'},j} = [x_j]_{g_i^{\sigma'},j}\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \;\wedge \\
&\left[\forall \sigma' \in \Sigma_i.\; \left(\hat{x}_j \equiv_{a_i^{\sigma'},j} x_j\right) \wedge \left(\hat{x}_j \equiv_{g_i^{\sigma'},j} x_j\right)\right]\} \\
&= \{\hat{x}^k_{jj} \in X_{jj} \mid \hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j) \;\wedge\; \hat{x}^k_{jj} \neq x^k_{jj} \;\wedge\; \hat{x}_j \equiv_{i,j} x_j\}
\end{aligned}
$$

As a result if as in part 1, ${}^2\mathscr{I}_{ji}(\sigma) \setminus {}^3\mathscr{I}_{ji}(\sigma) \neq \emptyset$, there exists at least two distinct $\hat{x}_j$ and x_j which belong to the same equivalence class under $\equiv_{i,j}$, which is equivalent to saying that $\ker(id_{\mathbb{F}_p}) < \equiv_{i,j}$. Since $|\ker(id_{\mathbb{F}_p})| = p$, we have then $|\equiv_{i,j}| < p$. This proves part 1.

To see part 2, notice that since $\sigma \in \Sigma_{o,j}$, by Definition 4.6 and Algorithm 4.1, $x_j \neq \hat{x}_j$. As a result there exists some $\hat{x}^k_{jj} \in X_{jj}$ such that $\hat{x}^k_{jj} := a_j(x^k_{jj}, \sigma)(X_j)$,

$\hat{x}^k_{jj} \neq x^k_{jj}$. By assumption $x_j \in \arg(\mathfrak{a}_i^{\sigma'})$ or $x_j \in \arg(\mathfrak{g}_i^{\sigma'})$ hold(s). Therefore, by Definition 4.10, there exists $x' \in X_{ii}$ such that $x^k_{ij} \in \arg(a_i(x', \sigma'))$ or $x^k_{ij} \in \arg(g_i(\sigma'))$, respectively. Thus, by Definition 5.4, $x^k_{jj} \in {}^2\mathscr{J}_{ji}(\sigma)$. However, the fact that $x_j \equiv_{i,j} \hat{x}_j$ implies that $[\hat{x}_j]_{\mathfrak{a}_i^{\sigma'},j} = [x_j]_{\mathfrak{a}_i^{\sigma'},j}$ and $[\hat{x}_j]_{\mathfrak{g}_i^{\sigma'},j} = [x_j]_{\mathfrak{g}_i^{\sigma'},j}$, which then by Definition 5.6 implies that $x^k_{jj} \notin {}^3\mathscr{J}_{ji}$, and $|{}^3\mathscr{J}_{ji}(\sigma)| < |{}^2\mathscr{J}_{ji}(\sigma)|$.

■

Proof of Lemma 5.8

From Definition 5.9, for each $i, j, k \in I, j \neq i, k \neq j, f \in \mathfrak{a}_i \cup \mathfrak{g}_i$, we have the following.

$$\Lambda_{i,-j,k} \subseteq \Lambda_{f,-j,k} \implies [\Lambda_{i,-j} = \prod_{k \in I \setminus \{j\}} \Lambda_{i,-j,k} \subseteq \prod_{k \in I \setminus \{j\}} \Lambda_{f,-j,k} = \Lambda_{f,-j}]$$
$$\implies [\forall v, v' \in \mathbb{F}_p.\ v \equiv_{f,j,\Lambda_{f,-j}} v' \implies v \equiv_{f,j,\Lambda_{i,-j}} v'],$$

i.e. $\equiv_{f,j,\Lambda_{i,-j}}$ is well defined and, since $\equiv_{i,j,\Lambda_{i,-j}}$ is the meet of such relations on a common set, it is well defined, too. ■

Proof of Proposition 5.3

We show that communication policy 4 leads to correct reevaluation of guard and updating functions of the PDS (4.14), which then by Problem 5.1 and Definition 4.7 proves the claim for this policy. This is done by induction on the centralized supervisors' states seen along the system's evolution. The assumptions of strong connectedness of the network, instantaneous communication, and lossless channels insure that every two supervisors have a direct link between them, through which communication is done with no delay and no losses.

Base: Base is the same as what appears in the proof of Proposition 5.2 and is not repeated.

Inductive step: Let $i, j \in I$, $i \neq j$, and the PDS representation (4.14) be at state $\mathbf{x} \in \mathbb{F}_p^n$, where $\mathbf{x} = (x_1, \cdots, x_n)$, and $x_i = (x^h_{ii}, \cdots, x^1_{ii})$. The induction

assumption states that at state $\mathbf{x}$, every $\mathbf{S}_i$ evaluates its guard functions correctly and if an event $\sigma \in \Sigma$ occurs, every $\mathbf{S}_i$ correctly reevaluates its updating functions. By Definition 4.10 this is equivalent to saying that in the EFSM implementation, each $\mathbf{S}_i$ correctly evaluates its guards and, upon the occurrence of σ, reevaluates its actions correctly. Let $\hat{\mathbf{x}} \in \mathbb{F}_p^n$ be the next state reached by the PDS through σ, where $\hat{\mathbf{x}} = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$ and $\hat{x}_i = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$. We show that by using communication policy 4, the PDS correctly reevaluates its updating functions and guard formulas at $\hat{\mathbf{x}}$, too. Towards this end, we distinguish two possible cases.

1. If $\sigma \in \Sigma_{uo}$, no $\mathbf{S}_i$ observes σ and therefore no $x \in X_{ii}$ changes. Thus for every i, $\hat{x}_i = x_i$ and the PDS would remain in $\mathbf{x}$. In other words, no private variable changes and each $\mathbf{S}_i$ still has the latest copies of others' private variables, which it had before. Therefore, by induction assumption, it can correctly evaluates its guards and, upon the occurrence of a next event, its actions, which then by Definition 4.10, leads to correct evaluation of guard and updating functions. This justifies the fact that, by Definition 5.10, we have $\mathscr{I}_{ji}(\sigma) = \emptyset$.

2. If $\sigma \in \Sigma_o$, it is observed by every $\mathbf{S}_j$ for which $j \in I_o(\sigma)$. By communication policy 4, this may fire a set of communication events $\mathscr{I}_{ji}(\sigma)$ such that the following hold.

$$\forall j \in I_o(\sigma).\ \ \hat{x}_j := \mathfrak{a}_j^\sigma(\mathbf{x}) \implies \hat{x}_j \neq x_j$$

[Lemma 4.2, Eq. (4.5), and Prop. 4.4]

$$\implies \left[\exists k \in J.\ \ \hat{x}_{jj}^k := a_j(x_{jj}^k, \sigma)(X_j) \implies \hat{x}_{jj}^k \neq x_{jj}^k\right] \qquad \text{[Defn. 4.10]}$$

$$\Longrightarrow \big[\forall i, l \in I, i \neq j, l \neq i.\ \big((|\equiv_{l,i,\Lambda_{l,-i}}| < p) \wedge (x_j \in \Lambda_{l,-i,j} \Longrightarrow \hat{x}_j \notin \Lambda_{l,-i,j})$$

$$\wedge\ (x_j \notin \Lambda_{l,-i,j} \Longrightarrow \hat{x}_j \in \Lambda_{l,-i,j})\big) \Longrightarrow\ \hat{x}^k_{jj} \in \mathscr{I}_{ji}(\sigma) \Longrightarrow \hat{x}^k_{ij} := \hat{x}^k_{jj}\big]\ (*)$$

[Defn. 5.10-Step 1, Eq. (5.1)]

$$\Longrightarrow \big[\forall i, l \in I, i \neq j, l \neq i.\ (|\equiv_{l,i,\Lambda_{l,-i}}| < p) \Longrightarrow\ \mathbf{S}_i \text{ knows } [\hat{x}_i]_{l,i,\Lambda_{l,-i}}\big]$$

[Defn. 5.8, Defn. 5.9]

$$\Longrightarrow \big[\forall i \in I \setminus \{j\}, \forall y \in X_{ii}, \forall \sigma' \in \Sigma.$$

$$\big(x^k_{ij} \in \arg(a_i(y, \sigma')) \vee x^k_{ij} \in \arg(g_i(\sigma'))\big) \wedge$$

$$\big((|\equiv_{i,j,\Lambda_{i,-j}}| < p) \Longrightarrow \neg(\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j)\big) \wedge \big(\forall \sigma \in \Sigma_{o,j}.\ \hat{x}^k_{jj} \notin \mathscr{I}_{ji}(\sigma)\big)$$

$$\Longrightarrow\ \hat{x}^k_{jj} \in \mathscr{I}_{ji}(\{\mathscr{I}_{mj}(\sigma) |\ m \in I, m \neq j, \sigma \in \Sigma_{o,m}\}) \Longrightarrow\ \hat{x}^k_{ij} := \hat{x}^k_{jj}\big]$$

(**) [Defn. 5.10-Step 2, Eq. (5.1)]

$$\Longrightarrow \big[\forall i \in I \setminus \{j\}.\ \neg(\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j) \Longrightarrow\ \hat{x}^k_{ij} := \hat{x}^k_{jj}\big] \qquad [(*), (**)]$$

$$\Longrightarrow \big[\forall i, j \in I, i \neq j, \forall k \in J.\ (\hat{x}^k_{jj} = x^k_{jj} \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} = x^k_{jj})$$

$$\wedge\ \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge \neg(\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j)] \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj}\big)$$

$$\wedge\ \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\hat{x}_j \equiv_{i,j,\Lambda_{i,-j}} x_j)] \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} \neq \hat{x}^k_{jj}\big)\big]$$

$$\Longrightarrow \big[\forall i, j \in I, i \neq j, \forall k \in J, \forall \sigma' \in \Sigma.\ (\hat{x}^k_{jj} = x^k_{jj} \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} = x^k_{jj})$$

$$\wedge\ \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\exists \mathbf{x}_{-j} \in \mathbb{F}_p^{n-1}.\ \mathfrak{a}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) \neq \mathfrak{a}_i^{\sigma'}(x_j, \mathbf{x}_{-j})$$

$$\vee\ \mathfrak{g}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) \neq \mathfrak{g}_i^{\sigma'}(x_j, \mathbf{x}_{-j}))] \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj}\big)$$

$$\wedge\ \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\forall \mathbf{x}_{-j} \in \mathbb{F}_p^{n-1}.\ \mathfrak{a}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) = \mathfrak{a}_i^{\sigma'}(x_j, \mathbf{x}_{-j})$$

$$\wedge\ \mathfrak{g}_i^{\sigma'}(\hat{x}_j, \mathbf{x}_{-j}) = \mathfrak{g}_i^{\sigma'}(x_j, \mathbf{x}_{-j}))] \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} \neq \hat{x}^k_{jj}\big)\big]$$ [Defn. 5.8]

$$\Longrightarrow \big[\forall i,j \in I, i \neq j, \forall k \in J, \forall \sigma' \in \Sigma.\ \ (\hat{x}^k_{jj} = x^k_{jj} \Longrightarrow \hat{x}^k_{ij} = x^k_{ij} = x^k_{jj})$$

$$\wedge \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\exists X_{i-j} \in \mathbb{B}^{(n-1)h}, k' \in J.$$

$$a_i(x^{k'}_{ii}, \sigma')((\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j})$$

$$\neq a_i(x^{k'}_{ii}, \sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j})$$

$$\vee\ g_i(\sigma')(\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j})$$

$$\neq g_i(\sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}))] \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj}\big)$$

$$\wedge \big([\hat{x}^k_{jj} \neq x^k_{jj} \wedge (\forall X_{i-j} \in \mathbb{B}^{(n-1)h}, \forall k' \in J.$$

$$a_i(x^{k'}_{ii}, \sigma')((\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j})$$

$$= a_i(x^{k'}_{ii}, \sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j})$$

$$\wedge\ g_i(\sigma')(\hat{x}^h_{jj}, \cdots, \hat{x}^k_{jj}, \cdots, \hat{x}^1_{jj}), X_{i-j})$$

$$= g_i(\sigma')((x^h_{ij}, \cdots, x^k_{jj}, \cdots, x^1_{ij}), X_{i-j}))] \Longrightarrow \hat{x}^k_{ij} = \hat{x}^k_{jj}\big)\ \big]$$

[Defn. 4.10]

$$\Longrightarrow \big[\forall i \in I, \forall \sigma'' \in \Sigma, \forall z \in X_{ii}.$$

$$a_i(z, \sigma'')(\hat{X}_i) \text{ and } g_i(\sigma'')|_{\hat{X}_i} \text{ are reevaluated correctly.}\big]$$

$$\Longrightarrow \big[\forall i \in I, \forall \sigma'' \in \Sigma.\ \ \mathfrak{a}^{\sigma''}_i(\hat{\mathbf{x}}) \text{ and } \mathfrak{g}^{\sigma''}_i(\hat{\mathbf{x}}) \text{ are reevaluated correctly.}\big]$$

[Defn. 4.10]

In other words, every $\mathbf{S}_i$ receives the latest copy of those external variables which are required to reevaluate its updating and guard functions at state $\hat{x}$. This completes the inductive step of the proof, hence proving the claim. ■

Proof of Lemma 5.10

We have the following.

$$\forall i \in I.\ \ F(\mathcal{O}_i) = \mathcal{O}_i \cup \bigcup_{j \in \mathcal{O}_i} \mathcal{N}_{\mathfrak{a}_j} \qquad \text{[Defn. of } F\text{]}$$

$$\wedge\ [\forall j \in \mathcal{O}_i.\ \mathcal{N}_{\mathfrak{a}_j} \subseteq \mathcal{O}_i] \qquad \text{[Defn. 5.11]}$$

$$\Longrightarrow \forall i \in I.\ \ [F(\mathcal{O}_i) = \mathcal{O}_i \cup \bigcup_{j \in \mathcal{O}_i} \mathcal{N}_{\mathfrak{a}_j}]\ \wedge\ [\bigcup_{j \in \mathcal{O}_i} \mathcal{N}_{\mathfrak{a}_j} \subseteq \mathcal{O}_i]$$

$$\Longrightarrow F(\mathcal{O}_i) = \mathcal{O}_i,$$

i.e. $\mathcal{O}_i$ is a fixed point of F. ■

Proof of Proposition 5.4

We show that communication policy 5 leads to correct reevaluation of guard and updating functions of the PDS (4.14), which then by Problem 5.1 and Definition 4.7 proves the claim for this policy. This is done by induction on the centralized supervisors' states seen along the system's evolution. The assumptions of strong connectedness of the network, instantaneous communication, and lossless channels insure that every two supervisors have a direct link between them, through which communication is done with no delay and no losses.

Base: The centralized supervisor starts from state r_0 and since no event has occurred yet, by Definitions 4.6 and 4.7, the label assigned to r_0 is $\mathbf{0}$, i.e. $\mathbf{x} = \mathbf{0}$. On the other hand, by Definition 5.12, for every $i \in I, j \in \mathcal{O}_i \setminus \{i\}$, all copy variables, i.e. x_j^is, are initialized to 0, therefore $\mathbf{x}^i = \mathbf{0}$. As a result, for all $\alpha \in \Sigma_{c,i}$ we have $\mathfrak{g}_i^\alpha(\mathbf{x}^i) = \mathfrak{g}_i^\alpha(\mathbf{x})$, and for all $\sigma \in \Sigma_{o,i}$ we have $\hat{x}_i = \mathfrak{a}_i^\sigma(\mathbf{x}^i) = \mathfrak{a}_i^\sigma(\mathbf{x})$. In other words, every guard and updating function is evaluated correctly. Notice that since no event has occurred, no supervisor has seen any event and thus, by (5.4), there is no issued $\mathscr{I}$ events.

Inductive step: Let $i \in I$, $j \in \mathcal{O}_i \setminus \{i\}$, and PDS representation (4.14) be at state $\mathbf{x} \in \mathbb{F}_p^n$, where $\mathbf{x} = (x_1, \cdots, x_n)$. The induction assumption states that at state $\mathbf{x}$, we have $\mathbf{x}^i = \mathbf{x}$, and therefore, for all $\alpha \in \Sigma_{c,i}$ we have $\mathfrak{g}_i^\alpha(\mathbf{x}^i) = \mathfrak{g}_i^\alpha(\mathbf{x})$, and for all $\sigma \in \Sigma_{o,i}$ we have $\hat{x}_i = \mathfrak{a}_i^\sigma(\mathbf{x}^i) = \mathfrak{a}_i^\sigma(\mathbf{x})$, i.e. every $\mathbf{S}_i$ evaluates its guard functions correctly, and if an event $\sigma \in \Sigma$ occurs, every $\mathbf{S}_i$ can reevaluate its private variable correctly. Let $\hat{\mathbf{x}} \in \mathbb{F}_p^n$ be the next state reached by the PDS through σ, where $\hat{\mathbf{x}} = (\hat{x}_1, \cdots, \hat{x}_n)$. We show that by using communication policy 5, every $\mathbf{S}_i$ correctly reevaluates its updating and guard functions at $\hat{\mathbf{x}}$ by proving that $\hat{\mathbf{x}}^i = \hat{\mathbf{x}}$. Towards this end, we distinguish two possible cases.

1. If $\sigma \in \Sigma_{uo}$, no $\mathbf{S}_j$ observes σ and therefore no x_j changes. Thus for every j, $\hat{x}_j = x_j$ and the PDS would remain in $\mathbf{x}$. In other words, no private variable changes and, since by (5.5) no communication takes place following the occurrence of the events in Σ_{uo}, $x_j^i = x_j$ holds by induction assumption. Therefore, by induction assumption, every $\mathbf{S}_i$ can correctly evaluates its guard functions and, upon the occurrence of every next event, its corresponding updating function.

2. Now assume that $\sigma \in \Sigma_o$ occurs, thus it is observed by every $\mathbf{S}_j$ for which $j \in I_o(\sigma)$. For $i \in I_o(\sigma)$ the following holds.

$$\forall i \in I, \forall m \in \mathcal{O}_i \setminus \{i\}, \forall \sigma \in \Sigma_{o,i}.$$

$$\left[[\sigma \in \Sigma_{o,m} \Longrightarrow \hat{x}_j^i := \mathfrak{a}_{im}^\sigma(\mathbf{x}^i) = \mathfrak{a}_{im}^\sigma(\mathbf{x})] \wedge [\sigma \notin \Sigma_{o,m} \Longrightarrow \hat{x}_j^i = x_j^i = x_j]\right]$$

[Assumption 5.3, Induction assumption: $\mathbf{x}^i = \mathbf{x}$]

$$\Longrightarrow \left[[\sigma \in \Sigma_{o,m} \Longrightarrow \hat{x}_m^i := \mathfrak{a}_{im}^\sigma(\mathbf{x}) = \mathfrak{a}_m^\sigma(\mathbf{x}) = \hat{x}_m] \wedge [\sigma \notin \Sigma_{o,m} \Longrightarrow \hat{x}_j^i = x_j^i]\right.$$

[Assumption 5.1]

$$\left. \wedge\, [\hat{x}_i := \mathfrak{a}_i^\sigma(\mathbf{x}^i) = \mathfrak{a}_i^\sigma(\mathbf{x})]\right] \qquad \text{[Induction assumption: } \mathbf{x}^i = \mathbf{x}]$$

$$\Longrightarrow [\forall i \in I, \forall m \in \mathcal{O}_i \setminus \{i\}, \forall \sigma \in \Sigma_{o,i}.\ \ \hat{x}_m^i = \hat{x}_m]$$

$$\Longrightarrow [\forall i \in I.\ \ i \in I_o(\sigma) \implies \hat{\mathbf{x}}^i = \hat{\mathbf{x}}], \qquad (\dagger)$$

i.e. every supervisor which observes σ updates its own private variable and the copies of the private variables of other supervisors, which it stores, correctly.

Let us now assume that $i \notin I_o(\sigma)$. Let $\alpha_i \in \Sigma_{o,i} \cup \{\epsilon\}$, encoded as $\alpha_i = (y^e_{ll}, \cdots, y^1_{ll})$, be the last event observed by $\mathbf{S}_i$. By communication policy 5, observation of σ may fire a set of communication events $\mathscr{I}_{ji}(\sigma)$ such that the following holds.

$$\exists!\ j = I_{com}(\sigma) \in I_o(\sigma), \forall i \in I \setminus I_o(\sigma), \forall k \in \mho.$$

$$\big[\big(\exists l \in \mathcal{O}_i \setminus \{i\}.\ l \in I_o(\sigma)\big)\ \wedge \sigma = (\hat{y}^e_{jj}, \cdots, \hat{y}^1_{jj}) \wedge \hat{y}^k_{jj} \neq y^k_{jj}$$

$$\implies \hat{y}^k_{jj} \in \mathscr{I}_{ji}(\sigma)] \qquad (\dagger\dagger)\ \text{[Defn. 5.14, (5.4), Defn. 5.13]}$$

$$\Longrightarrow \big[[\hat{y}^k_{jj} \in \mathscr{I}_{ji}(\sigma) \Longrightarrow \hat{y}^k_{ij} = \hat{y}^k_{jj}] \wedge [\hat{y}^k_{jj} \notin \mathscr{I}_{ji}(\sigma) \Longrightarrow \hat{y}^k_{ij} = y^k_{ij}]\big]$$

[(5.5)-Conjunct 1]

$$\Longrightarrow [(\hat{y}^e_{ij}, \cdots, \hat{y}^1_{ij}) = \sigma] \qquad \text{[Assumption 5.2, premise of } (\dagger\dagger)]$$

$$\Longrightarrow \big[\forall m \in \mathcal{O}_i \setminus \{i\}, \forall \mathbf{x}^i \in \mathbb{F}_p^{|\mathcal{O}_i|}.\ [m \in I_o(\sigma) \Longrightarrow \hat{x}^i_m := \mathfrak{a}^\sigma_{im}(\mathbf{x}^i)]$$

[(5.5)-Conjunct 2]

$$\wedge\ [m \notin I_o(\sigma) \Longrightarrow \hat{x}^i_m := x^i_m]\big]$$

$$\Longrightarrow \big[\forall m \in \mathcal{O}_i \setminus \{i\}, \forall \mathbf{x}^i \in \mathbb{F}_p^{|\mathcal{O}_i|}.\ [m \in I_o(\sigma) \Longrightarrow \hat{x}^i_m = \mathfrak{a}^\sigma_{ij}(\mathbf{x}^i) = \mathfrak{a}^\sigma_j(\mathbf{x}) = \hat{x}_m]$$

[Assumption 5.1, Induction assumption: $\mathbf{x}^i = \mathbf{x}$]

$$\wedge\ [m \notin I_o(\sigma) \Longrightarrow \hat{x}^i_m = x^i_m = x_m]\big]$$

[Cor. 4.1, Defn. 4.7, Induction assumption: $\mathbf{x}^i = \mathbf{x}$]

$$\wedge\ [\hat{x}_i := \mathfrak{a}^\sigma_i(\mathbf{x}^i) = \mathfrak{a}^\sigma_i(\mathbf{x})]\big]$$

[$i \notin I_o(\sigma)$, Induction assumption: $\mathbf{x}^i = \mathbf{x}$]

$$\Longrightarrow [\forall m \in \mathcal{O}_i \ \hat{x}^i_m = \hat{x}_m]$$

$$\Longrightarrow [\forall i \in I.\ \ i \notin I_o(\sigma) \implies \hat{\mathbf{x}}^i = \hat{\mathbf{x}}], \qquad (\clubsuit)$$

i.e. when $\mathbf{S}_i$ does not observe σ, through received communication, it can still correctly reevaluate its copies of other supervisors' private variables, which it stores.

As a result, from (†) and (♣) we can conclude that $\hat{\mathbf{x}}^i = \hat{\mathbf{x}}$, i.e. .

$$\forall i \in I, \forall \beta \in \Sigma_{o,i}, \forall \gamma \in \Sigma_{c,i}. \quad \mathfrak{a}_i^{\beta}(\hat{\mathbf{x}}^i) = \mathfrak{a}_i^{\beta}(\hat{\mathbf{x}}) \wedge \mathfrak{g}_i^{\gamma}(\hat{\mathbf{x}}^i) = \mathfrak{g}_i^{\gamma}(\hat{\mathbf{x}})$$

In other words, at state $\hat{\mathbf{x}}$, every $\mathbf{S}_i$ reevaluates its guard and updating functions correctly by computing correct copies of other supervisors' private variables through using copies of their updating functions and then plugging these copy variables in its updating and guard functions. This completes the inductive step of the proof, hence proving the claim. ■

Proof of Lemma 5.12

The fact that $\mathbf{S}_i$ requires a CFC from $\mathbf{S}_j$ implies that there exists an event $\alpha \in \Sigma_{c,i}$ such that $x_j \in \arg(\mathfrak{g}_i^{\alpha})$. Let us assume that an on-demand information policy is employed. Then at some state $\mathbf{x} \in \mathbb{F}_p^n$, to reevaluate $\mathfrak{g}_i^{\alpha}(\mathbf{x})$, $\mathbf{S}_i$ should have a true copy of x_j. Then, by on-demand policy of Definition 5.18, $\mathbf{S}_j$ issues $\mathscr{I}_{ji}(\mathscr{I}_{ij}(\beta))$, only after it receives $\mathscr{I}_{ij}(\beta)$, where $\beta \in \Sigma_{o,i}$. On the other hand, the fact that $\mathbf{S}_i$ does not require any CFOs from $\mathbf{S}_j$, implies that for all $\sigma \in \Sigma_{o,i}$ we have $x_j \notin \arg(\mathfrak{a}_i^{\sigma})$. Therefore, before reaching at state $\mathbf{x}$, there has not existed any event $\beta \in \Sigma_{o,i}$, upon whose observation $\mathbf{S}_i$ could send a request to $\mathbf{S}_j$. This implies that, unless $\mathbf{S}_i$ sends its request infinitely may times, which

is of course not possible, it cannot receive the required information to build a true copy of x_j at the right time. ■

Proof of Proposition 5.5

We show that communication policy 6 leads to correct reevaluation of guard and updating functions of the PDS (4.14), which then by Problem 5.1 and Definition 4.7 proves the claim for this policy. This is done by induction on the centralized supervisors' states seen along the system's evolution. The assumptions of strong connectedness of the network, instantaneous communication, and lossless channels insure that every two supervisors have a direct link between them, through which communication is done with no delay and no losses.

Base: The centralized supervisor starts from state r_0 and since no event has occurred yet, by Definitions 4.6 and 4.7, the label assigned to r_0 is $\mathbf{0}$. On the other hand, by Definition 4.9, all variables in X are initialized to zero, i.e. for each $i \in I$, every $x \in X_{ii}$ and its copies are zero and therefore each supervisor estimates the state of the system to be $\mathbf{x} = (x_1, \cdots, x_n) = \mathbf{0}$ and evaluate its guards and actions and, respectively, guard and updating functions correctly. Also, by Definition 5.20, for every $i, j \in I, i \neq j$, such that $\mathbf{S}_j$ depends on $\mathbf{S}_i$, all variables in $X_{j,ii}$ are equal to 0, which implies that for all $k \in J$ we have $x^k_{j,ii} = x^k_{ji}$. This means that $\mathbf{S}_i$ has stored in $X_{j,ii}$ the correct copies of the variables in X_{ji}.

Inductive step: Let $i, j \in I$, $i \neq j$, and PDS representation (4.14) be at state $\mathbf{x} \in \mathbb{F}_p^n$, where $\mathbf{x} = (x_1, \cdots, x_n)$ and $x_i = (x^h_{ii}, \cdots, x^1_{ii})$. The induction assumption states that at state $\mathbf{x}$, every $\mathbf{S}_i$ has acquired every x_j, for which an event $\alpha \in \Sigma_{o,i}$ exists such that $x_j \in \arg(\mathfrak{a}^\alpha_i)$, from $\mathbf{S}_j$, i.e. for every $k \in J$ we have $x^k_{ij} = x^k_{jj}$. As a result, $\mathbf{S}_i$ has evaluated its actions, or equivalently by Definition 4.10, its updating function $\mathfrak{a}^\alpha_i(\mathbf{x})$ and hence x_i, correctly. Since

guard functions are all independent, having the correct value of x_i implies the correct evaluation of guards or equivalently, by Definition 4.10, guard functions $\mathfrak{g}_i^{\beta}(x_i)$, for all $\beta \in \Sigma_{c,i}$, by $\mathbf{S}_i$. Furthermore, $\mathbf{S}_j$ stores the correct values of the copies of its variables, which is kept by $\mathbf{S}_i$, i.e. for every $k \in J$ we have $x_{i,jj}^k = x_{ij}^k$. Let $\hat{\mathbf{x}} \in \mathbb{F}_p^n$ be the next state reached by the PDS through σ, where $\hat{\mathbf{x}} = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$ and $\hat{x}_i = (\hat{x}_{ii}^h, \cdots, \hat{x}_{ii}^1)$. We show that by using communication policy 6, the PDS correctly reevaluates its updating functions and guard formulas at $\hat{\mathbf{x}}$, too. Towards this end, we distinguish two possible cases.

1. If $\sigma \in \Sigma_{uo}$, no $\mathbf{S}_i$ observes σ and therefore no $x \in X_{ii}$ changes. Thus for every i, $\hat{x}_i = x_i$ and the PDS would remain in $\mathbf{x}$. In other words, no private variable changes and each $\mathbf{S}_i$ still has the latest copies of others' private variables, which it had before. Therefore, by induction assumption, it can correctly evaluates its actions and guards, which then by Definition 4.10, leads to correct evaluation of updating and guard functions. This justifies the fact that, by Definition 5.21, we have $\mathscr{I}_{ji}(\sigma) = \emptyset$.

2. If $\sigma \in \Sigma_o$, it is observed by every $\mathbf{S}_i$ for which $i \in I_o(\sigma)$. As a result, every such $\mathbf{S}_i$ updates its x_i. By communication policy 6, this may fire a set of communication events $\mathscr{I}_{ij}(\sigma)$ and $\mathscr{I}_{ji}(\mathscr{I}_{ij}(\sigma))$, such that the following hold.

$$[\forall i \in I_o(\sigma), \forall j \in I \setminus \{i\}.\ x_j \in \arg(\mathfrak{a}_i^{\sigma}) \implies \mathscr{I}_{ij}(\sigma) = \{1\}]$$

[Defn. 5.21-Conj. 1]

$$\implies [\forall k \in J.\ x_{jj}^k \neq x_{i,jj}^k \implies \left(x_{jj}^k \in \mathscr{I}_{ji}(\mathscr{I}_{ij}(\sigma))\right) \quad \text{[Defn. 5.21-Conj. 2]}$$

$$\wedge \left(x_{i,jj}^k := x_{jj}^k\right)] \quad \text{[Defn. 5.20]}$$

$$\Longrightarrow [\forall k \in J.\ \left(x^k_{jj} \neq x^k_{i,jj} \implies x^k_{ij} := x^k_{jj} \wedge x^k_{i,jj} := x^k_{jj}\right) \qquad \text{[Eq. (5.1)]}$$

$$\wedge \left(x^k_{jj} = x^k_{i,jj} \implies x^k_{ij} = x^k_{jj} \wedge x^k_{i,jj} = x^k_{jj}\right)]$$

[Induction assumption]

$$\Longrightarrow [\forall k \in J.\ x^k_{ij} = x^k_{jj} \ \wedge\ x^k_{i,jj} = x^k_{jj}]$$

$$\Longrightarrow [x_j \in \mathfrak{a}^\sigma_i \implies X_{ij} = X_{jj} \ \wedge\ X_{i,jj} = X_{ij}]\ (\S)\ \text{[Defn. 4.9, Defn. 5.20]}$$

$$\Longrightarrow [\forall k \in J.\ \hat{x}^k_{ii} := a_i(x^k_{ii}, \sigma)(X_i) \text{ is computed correctly}]$$

$$\Longrightarrow [\hat{x}_i = (\hat{x}^h_{ii}, \cdots, \hat{x}^1_{ii}) \text{ is computed correctly}] \qquad \text{[Defn. 4.10]}$$

$$\Longrightarrow [\forall \alpha \in \Sigma_{c,i}.\ \left(\mathfrak{g}^\alpha_i(\hat{x}_i) \text{ is computed correctly}\right) \qquad [\mathfrak{g}^\alpha_i \text{ is an IUF}]$$

$$\wedge \left(\forall l \in I \backslash I_o(\sigma).\ \hat{x}_l = x_l \wedge \mathfrak{g}^\alpha_l(x_l) \text{ is computed correctly}\right)]$$

[Induction assumption]

$$\Longrightarrow [\forall i \in I, \forall \alpha \in \Sigma_{c,i}, \forall \beta \in \Sigma_{o,i}.$$

$$\mathfrak{g}^\alpha_i(\hat{\mathbf{x}}) \text{ and } \mathfrak{a}^\beta_i(\hat{\mathbf{x}}) \text{ are computed correctly}]$$

In other words, every $\mathbf{S}_i$ who observes σ, receives the latest copy of all external variables on which it depends and therefore computes its associated updating function and guard functions correctly at the new state, $\hat{\mathbf{x}}$. Notice that the second conjunct of §, i.e. $X_{i,jj} = X_{ij}$, serves as part of the induction assumption for the next inductive step. This completes the inductive step of the proof, hence proving the claim. ■

Proof of Lemma 5.19

Let $s, s' \in \overline{K}$ and $\sigma \in \Sigma$. Then we have the following.

$$
\begin{aligned}
& s\sigma \in \overline{K} \ \wedge \ s'\sigma \in \overline{L} \ \wedge \ s \equiv_I s' \\
\Longrightarrow\ & s\sigma \in \overline{L} \ \wedge \ s'\sigma \in \overline{L} \ \wedge \ s\sigma \in \overline{K} \ \wedge \ s\sigma \equiv_I s'\sigma \\
& \qquad\qquad [K \subseteq L, \forall i \in I.\ P_i(s)P_i(\sigma) = P_i(s')P_i(\sigma)] \\
\Longrightarrow\ & s'\sigma \in \overline{K} \qquad\qquad [\text{Eq. (5.6)}]
\end{aligned}
$$

This, together with the fact that (5.7 is the same as (5.9), complete the proof. ■

Proof of Lemma 5.20

Let $s, s' \in \Sigma^*$ be two strings such that $s \equiv_I s'$, $s \equiv_U s'$, and $s \in K$. Then we have

$$
\begin{aligned}
& [s \in K \implies s \in L] && [K \subseteq L] \\
\wedge\ & [s \in L \ \wedge \ s \equiv_I s' \ \wedge \ s \equiv_U s' \implies s' \in L] && [\text{Defn. 5.26}] \\
\Longrightarrow\ & [s, s' \in L \ \wedge \ s \equiv_I s' \ \wedge \ s \in K \implies s' \in K], && [(5.7)]
\end{aligned}
$$

i.e. K is trace-closed. ■

Proof of Theorem 5.6

Proof is by reduction from PCP. For now let K and L be *regular* prefix-closed languages over Σ with $K \subseteq L$ and $I = \{1, 2\}$. If K is WJO with respect to L, for all $s, s' \in K$ and $\sigma \in \Sigma$ we have the following.

$$
s\sigma \in K \ \wedge \ s'\sigma \in L \ \wedge \ [\forall i \in I.\ P_i(s) = P_i(s')] \implies s'\sigma \in K
$$

Let $\mathcal{P} = (\Gamma, \{u_1, \ldots, u_m\}, \{w_1, \ldots, w_m\})$ be an instance of PCP where $u_i \in \Gamma^+$ and $w_i \in \Gamma^+$ are respectively the top and bottom strings of type i dominos, $1 \leq i \leq m$. We define an instance $\mathcal{W} = (\Sigma_{o,1}, \Sigma_{o,2}, K, L)$ of WJO^c

as follows:

$$\Sigma_{o,1} := \Gamma, \quad \Sigma_{o,2} := \{\sigma_1, \ldots, \sigma_m, \sigma\}, \quad \Sigma = \Sigma_{o,1} \dot{\cup} \Sigma_{o,2},$$

$$K = \overline{(u_1\sigma_1\sigma^* + \cdots + u_m\sigma_m\sigma^*)^*} + \overline{(w_1\sigma_1 + \cdots + w_m\sigma_m)^*}, \quad \text{and}$$

$$L = \overline{(u_1\sigma_1\sigma^* + \cdots + u_m\sigma_m\sigma^*)^*} + \overline{(w_1\sigma_1\sigma^* + \cdots + w_m\sigma_m\sigma^*)^*}.$$

Note that assuming that $s \in K$ terminates with a symbol from Σ_2, it can have only one of the two following forms, i.e.

$$s = u_{i_1}\sigma_{i_1}\sigma^{n_1}.u_{i_2}\sigma_{i_2}\sigma^{n_2}.\ldots.u_{i_l}\sigma_{i_l}\sigma^{n_l} \quad \text{or} \quad s = w_{j_1}\sigma_{j_1}.w_{j_2}\sigma_{j_2}.\ldots.u_{j_k}\sigma_{j_k}.$$

Suppose $\mathcal{P} \in \text{PCP}$, i.e. it has a match of length, say, k. Then:

$$u_{i_1}u_{i_2}\ldots u_{i_k} = w_{i_1}w_{i_2}\ldots w_{i_k}.$$

Let $s := (u_{i_1}\sigma_{i_1})(u_{i_2}\sigma_{i_2})\ldots(u_{i_k}\sigma_{i_k})$ and $s' := (w_{i_1}\sigma_{i_1})(w_{i_2}\sigma_{i_2})\ldots(w_{i_k}\sigma_{i_k})$. Clearly $s, s' \in K$, $s\sigma \in K$, $s'\sigma \in L$, $P_1(s) = P_1(s')$ and $P_2(s) = P_2(s')$, but $s'\sigma \notin K$, i.e. $\mathcal{W} \in \text{WJO}^c$.

Conversely, let $\mathcal{W} \in \text{WJO}^c$, that is there exist $s, s' \in K$ and $s\sigma \in K$ such that

$$s'\sigma \in L \ \wedge \ P_1(s) = P_1(s') \ \wedge \ P_2(s) = P_2(s') \ \wedge \ s'\sigma \notin K.$$

Then we must have the following.

$$[s\sigma \in L \cap K \implies s = u_{i_1}\sigma_{i_1}.u_{i_2}\sigma_{i_2}.\ldots.u_{i_l}\sigma_{i_l}]$$
$$\wedge \ [s'\sigma \in L \setminus K \implies s' = w_{j_1}\sigma_{j_1}.w_{j_2}\sigma_{j_2}.\ldots.u_{j_k}\sigma_{j_k}].$$

Since $P_2(s) = P_2(s')$ it follows that $k = l$ and for all $1 \leq r \leq k$, $i_r = j_r$. It follows from $P_1(s) = P_1(s')$ that $u_{i_1}u_{i_2}\ldots u_{i_k} = w_{i_1}w_{i_2}\ldots w_{i_k}$, i.e. $\mathcal{P}$ has a match of length k and therefore $\mathcal{P} \in \text{PCP}$.

Notice that the above argument makes no assumption about the observability of σ, hence it addresses the case where Σ_{uo} is nonempty, too. If the languages are not prefix-closed, then the verification procedure has two parts whose first one is checking the weak joint observability for the closed language

$\overline{K}$. Therefore, regardless of prefix-closure property, WJO is undecidable for general languages. Finally, the extension of the above proof to the case of $|I| \geq 2$ is intuitive, say by regarding Σ_2 as the union of all of the added alphabets, i.e. Σ_3, Σ_4, etc. ■

Proof of Lemma 5.21

Let $s, s' \in L(\mathbf{S})$ so that updating functions are well defined. The proof is by induction on the length of string $t = P_i(s) = P_i(s') \in \Sigma_{o,i}^*$, where base is trivial as $P_i(\epsilon) = \epsilon$.

Inductive step: Let the DSDES be at state $\mathbf{v} \in \mathbb{N}^n$. As the induction hypothesis, assume that for every $s, s' \in L(\mathbf{S})$ such that $P_i(s) = P_i(s')$ we have $\mathcal{A}_i(s, \mathbf{v}) = \mathcal{A}_i(s', \mathbf{v})$. Let s and s' lead the DSDES to states $\mathbf{w}$ and $\mathbf{w}'$, respectively. For every $\sigma \in \Sigma$ we show that $\mathcal{A}_i(s\sigma, \mathbf{v}) = \mathcal{A}_i(s'\sigma, \mathbf{v})$. To begin with, we notice that the induction assumption implies the following.

$$\mathcal{A}_i(s, \mathbf{v}) = \pi_i(\mathcal{A}(s, \mathbf{v})) = \pi_i(\mathbf{w}) \wedge \mathcal{A}_i(s', \mathbf{v}) = \pi_i(\mathcal{A}(s', \mathbf{v})) = \pi_i(\mathbf{w}') \qquad [(4.5)]$$

$$\Longrightarrow w_i = w_i' \qquad (\sharp) \text{ [Induction assump.]}$$

Next, for event σ the following argument holds.

$$\mathcal{A}(s\sigma, \mathbf{v}) = \mathcal{A}(\sigma, \mathcal{A}(s, \mathbf{v})) \wedge \mathcal{A}(s'\sigma, \mathbf{v}) = \mathcal{A}(\sigma, \mathcal{A}(s', \mathbf{v})) \qquad [(4.4)]$$

$$\Longrightarrow \mathcal{A}(s\sigma, \mathbf{v}) = \mathcal{A}(\sigma, \mathbf{w}) \wedge \mathcal{A}(s'\sigma, \mathbf{v}) = \mathcal{A}(\sigma, \mathbf{w}') \qquad \text{[Induction assump.]}$$

$$\Longrightarrow \pi_i\big((\mathcal{A}(s\sigma, \mathbf{v}))\big) = \pi_i(\mathcal{A}(\sigma, \mathbf{w})) \wedge \pi_i\big(\mathcal{A}(s'\sigma, \mathbf{v})\big) = \pi_i(\mathcal{A}(\sigma, \mathbf{w}'))$$

$$\Longrightarrow \mathcal{A}_i(s\sigma, \mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{w}) \wedge \mathcal{A}_i(s'\sigma, \mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{w}') \qquad [(4.5)]$$

$$\Longrightarrow \mathcal{A}_i(s\sigma, \mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{w}) = \mathcal{A}_i(s'\sigma, \mathbf{v}) = \mathcal{A}_i(\sigma, \mathbf{w}') \qquad [\mathcal{A}_i(\sigma, .) \text{ is an IUF, } (\sharp)]$$

This completes the proof. ■

Proof of Proposition 5.7

Assume that $s, s' \in L(\mathbf{G})$ and $\sigma \in \Sigma$ such that $s\sigma \in L(\mathbf{S})$, $s'\sigma \in L(\mathbf{G})$, $s' \in L(\mathbf{S})$, and $s \equiv_I s'$. Then we have:

$$\begin{aligned} & [s, s' \in L(\mathbf{S}) \ \wedge \ s \equiv_I s' \implies \xi(r_0, s) = \xi(r_0, s')] && \text{[Cor. 5.2]} \\ \implies \ & [s, s' \in L(\mathbf{S}) \ \wedge \ s\sigma \in L(\mathbf{S}) \ \wedge \ \xi(r_0, s) = \xi(r_0, s') \implies s'\sigma \in L(\mathbf{S})] \end{aligned}$$

Similarly for any two strings $s, s' \in L(\mathbf{S})$ such that $s \in L_m(\mathbf{S})$ and $s \equiv_I s'$ the following holds.

$$\begin{aligned} & [s, s' \in L(\mathbf{S}) \ \wedge \ s \equiv_I s' \implies \xi(r_0, s) = \xi(r_0, s')] && \text{[Cor. 5.2]} \\ \implies \ & [s \in L_m(\mathbf{S}) \ \wedge \ \xi(r_0, s) = \xi(r_0, s') \implies s' \in L_m(\mathbf{S})] && \blacksquare \end{aligned}$$

Proof of Lemma 5.22

The proof relies on the ALMs designed for each individual recognizer. To define an ALM $\ell_i(.)$ for each recognizer $\mathbf{A}_i$, its selfloops should be modified as follows.

(✠) For an event, say $\alpha \in \Sigma_{o,i} \cap \Sigma_{o,j}, j \in I$, which is selflooped in one state, say $q_1 \in Q_i$, and causes a state change in an state, say $q_2 \in Q_j$, of $\mathbf{A}_j$, a state $\hat{q}_1$ is added to $\mathbf{A}_i$ which inherits all the outgoing non-selfloop transitions of q_1, all selfloops at q_1 labeled with events in $\Sigma_{loop,i} = \Sigma_{uo,i} \cup \{\sigma \in \Sigma_{o,i} \mid \forall q, q' \in Q_i.\ q' = \eta_i(q, \sigma) \Rightarrow q = q'\}$, and q_1's marking, while all selfloop transitions at q_1 which are not labeled by events in $\Sigma_{loop,i}$ are replaced with transitions with the same labels from q_1 to $\hat{q}_1$ and vice versa.

This modification implies that for all $\sigma \in \Sigma$ the following holds.

$$[\exists i \in I(\sigma), q \in Q_i.\ \xi_i(q,\sigma) = q] \implies [\forall i \in I(\sigma), q \in Q_i.\ \xi_i(q,\sigma)! \implies \xi_i(q,\sigma) = q)] \quad \text{(B.1)}$$

The ALM $\ell_i(.)$ may be simply computed by assigning label 0 to $q_{i,0}$ and a unique nonzero label to each of the other states of the (possibly modified) $\tilde{\mathbf{A}}_i$[1]. Notice that labels are chosen from $\mathbb{N}$ and are singleton. Therefore, updating functions $\mathcal{A}_i : \Sigma_i \times \mathbb{N} \to \mathbb{N}$, being IUF by construction[2], can be then computed using Lemma 4.2 and (4.5).

Next we follow the construction of *product automaton* in [72]. Associated with each $\mathbf{A}_i$, compute $\tilde{\mathbf{A}}_i = (\tilde{Q}_i, \Sigma, \tilde{\eta}_i, \tilde{q}_{0,i}, \tilde{Q}_{m,i})$ for which we have

$$\forall \sigma \in \Sigma,\ \forall q \in \tilde{Q}_i.\ \tilde{\eta}_i(q,\sigma) = \begin{cases} \eta_i(q,\sigma) & ;\text{if } \sigma \in \Sigma_i, \\ q & ;\text{if } \sigma \in \Sigma \setminus \Sigma_i. \end{cases} \quad \text{(B.2)}$$

i.e. all events in $\Sigma \setminus \Sigma_i$ participate solely as selfloops around every state. Conditions (B.2) and (B.1) guarantee that

$$\forall i \in I,\ \forall q, q' \in \tilde{Q}_i,\ \forall \sigma \in \Sigma_{uo,i}.\quad q' = \tilde{\eta}_i(q,\sigma) \implies q' = q. \quad \text{(B.3)}$$

Consequently, as in (4.3) the domain of $\mathcal{A}_i$ can be extended to all events, i.e. $\hat{\mathcal{A}}_i : \Sigma \times \mathbb{N} \to \mathbb{N}$.

Next, compute the meet of $\tilde{\mathbf{A}}_i$'s, i.e. $\mathbf{A} = \bigcap_{i \in I} \tilde{\mathbf{A}}_i = (Q, \Sigma, \eta, q_0, Q_m)$,

[1] It can be seen also as a global labeling map (see Definition 4.3).
[2] In Definition 5.22 let $n = 1$ and this follows naturally.

where $Q = \tilde{Q}_1 \times \cdots \times \tilde{Q}_n$, $q_0 = (\tilde{q}_{1,0}, \ldots, \tilde{q}_{n,0})$, $Q_m = \tilde{Q}_{1,m} \times \cdots \times \tilde{Q}_{n,m}$, and

$$\forall \sigma \in \Sigma.\ \eta\big((q_1, \ldots, q_n), \sigma\big) = \begin{cases} \big(\tilde{\eta}_1(q_1, \sigma), \ldots, \tilde{\eta}_n(q_n, \sigma)\big) & ;\text{if } \forall i \in I.\ \ \tilde{\eta}_i(q_i, \sigma)!, \\ \text{not defined} & ;\text{otherwise.} \end{cases} \tag{B.4}$$

As a result of the modification (✠) above, in the transition structure of $\mathbf{A}$, no event can both participate in a selfloop around a state *and* cause an state-change in another state. The reason is that $q = (q_1, \cdots, q_n) = \eta(q, \sigma)$ implies that $\forall i \in I.\ \tilde{\eta}_i(q_i, \sigma) = q_i$, i.e. σ appears as a selfloop around state q_i in $\tilde{\mathbf{A}}_i$, and therefore by the modification (✠) made to $\mathbf{A}_i$ at the beginning of the proof, it cannot cause any state changes in any $\tilde{\mathbf{A}}_i$'s.

The (partial) transition function defined in (B.4) entails the following result, too: For all $s, s' \in \Sigma^*$ we have

$$\begin{aligned} & \eta(q_0, s)! \ \wedge\ \eta(q_0, s')! \ \wedge\ [\forall i \in I.\ P_i(s) = P_i(s')] \\ \Longrightarrow\ & \eta(q_0, s) = \big(\tilde{\eta}_1(q_{1,0}, s), \cdots, \tilde{\eta}_n(q_{n,0}, s)\big) \\ & = \big(\eta_1(q_{1,0}, P_1(s)), \cdots, \eta_n(q_{n,0}, P_n(s))\big) \\ & = \big(\eta_1(q_{1,0}, P_1(s')), \cdots, \eta_n(q_{n,0}, P_n(s'))\big) \\ & = \big(\tilde{\eta}_1(q_{1,0}, s'), \cdots, \tilde{\eta}_n(q_{n,0}, s')\big) = \eta(q_0, s'). \end{aligned} \tag{B.5}$$

In simple words, any two strings which are observationally alike to all agents, lead to the same state. As a result, for any $s, s' \in \Sigma^*$ and two *distinct* states $q, q' \in Q$ with $q = (q_1, \cdots, q_n) = \eta(q_0, s)$ and $q' = (q'_1, \cdots, q'_n) = \eta(q_0, s')$ we

have the following.

$$\exists i \in I.\ P_i(s) \neq P_i(s') \wedge q_i = \eta_i(q_{0,i}, P_i(s)) \wedge q'_i = \eta_i(q_{0,i}, P_i(s')) \wedge q_i \neq q'_i. \tag{B.6}$$

Define for $\mathbf{A}$ the labeling map $\ell((q_1, \cdots, q_n)) := \big(\ell_1(q_1), \cdots, \ell_n(q_n)\big)$. We show that $\ell(.)$ is an ALM for $\mathbf{A}$. Clearly, $\ell((q_0, \cdots, q_0)) := \big(\ell_1(q_{1,0}), \cdots, \ell_n(q_{n,0})\big) = (0, \cdots, 0) = \mathbf{0}$, thus Item 1 of Definition 4.6 is satisfied. Secondly, for any two distinct states $q = (q_1, \cdots, q_n) \in Q$ and $q' = (q'_1, \cdots, q'_n) \in Q$ we have the following.

$$\begin{aligned}
q \neq q' &\implies \exists i \in I.\ \ell_i(q_i) \neq \ell_i(q'_i) && [(B.6), \ell_i\text{: ALM}] \\
&\implies \ell(q) \neq \ell(q') \\
&\implies \ell(q) \cap \ell(q') = \emptyset. && [\ell(.) \text{ singleton}]
\end{aligned}$$

This shows that Items 2 of Definition 4.6 is satisfied, too. To verify the satisfaction of Item 3, notice that for any $q = \big(q_1, \cdots, q_n\big) \in Q$, $q' = \big(q'_1, \cdots, q'_n\big) \in Q$, $\sigma \in \Sigma_o$, and $\mathbf{v} = \ell(q) = \big(\ell_1(q_1), \cdots, \ell_n(q_n)\big) \in \mathbb{N}^n$ we have

$$\begin{aligned}
& q' = \eta(q, \sigma) \wedge\ q' \neq q \\
\implies\ & \exists i \in I_o(\sigma).q_i \neq q'_i && [(B.6)] \\
\implies\ & [\forall i \in I_o(\sigma).\ q_i \neq q'_i] \\
& \wedge\ [\forall i \in I \setminus I_o(\sigma).\ q_i = q'_i] && [(\maltese)] \\
\implies\ & [\forall i \in I_o(\sigma).\ \ell_i(q_i) \neq \ell(q'_i)] \\
& \wedge\ [\forall i \in I \setminus I_o(\sigma).\ \ell_i(q_i) = \ell_i(q'_i)] && [\ell_i : \text{ALM}] \\
\iff\ & \exists!\ \mathbf{v}' = \ell(q') = \big(\ell_1(q'_1), \cdots, \ell_n(q'_n)\big) \\
& \in \mathbb{N}^n.\ [\forall i \in I_o(\sigma).\ v'_i \neq v_i] \\
& \wedge\ [\forall j \in I \setminus I_o(\sigma).\ v'_j = v_j]. && [\ell_i : \text{ALM}]
\end{aligned}$$

Thus we have shown that $\ell(.) = (\ell_1, \cdots, \ell_n)$ is indeed an ALM for **A**. By definition, each ℓ_i assigns its labels independent of the other agents' labels, so its associated updating functions $\hat{\mathcal{A}}_i(.,.)$ implement the evolution of the system through change of the labels. Therefore $\mathcal{A}(.,.) = (\hat{\mathcal{A}}_1(.,.), \cdots, \hat{\mathcal{A}}_n(.,.))$ represent independent updating functions for **A**. Since modifications 1 and 2 are possible for every deterministic recognizer, the above construction holds for every recognizer **A**. Notice that although some states may not be reachable in **A**, this issue is irrelevant as long as the existence proof is concerned. ■

Proof of Proposition 5.8

By Definition 5.29 we have the following.

$\exists z_1 \in \mathbb{N}, \exists \hat{Q} = \{q_0, \cdots, q_{z_1-1}\} \subseteq R.$

$(z_1 \leq |R| \ \wedge \ \eta_\alpha^{i,1} = (q_0, \cdots, q_{z_1-1})) \wedge$

$(\forall q \in R, q' \in \hat{Q}.\ q' = \xi(q,\alpha) \implies q \in \hat{Q}) \wedge$ [Defn. 5.30]

$(\forall q \in R.\ \xi(q,\alpha)! \implies q \in \hat{Q})$ $[|\Delta_\alpha^i| = 1]$

$\implies (\forall q, q' \in R.\ q' = \xi(q,\alpha) \implies q, q' \in \hat{Q})$ [Defn. 5.29]

This insures that all α-labeled transitions are between the states in $\hat{Q}$. Moreover, by Definition 5.29, the target of a transition fired at q_j is q_{j+1}, where addition is modulo z_1. For simplicity of illustration, let $|I| = 2$ and $\alpha \in \Sigma_{o,1}$. Correspondingly, Latin square LS_1 in Fig. B.1-a can be employed to implement an IUF for $\mathcal{A}_i(\alpha,.)$, where arrows specify the image of each column (let $j = 1$ in the figure).

Therefore, if $\hat{Q} = R$, LS_1 yields an ALM with IUFs associated with event α. If $R \setminus \hat{Q}$ is nonempty, then let $\hat{R} = R \setminus \hat{Q}$, where $z_2 = |\hat{R}| = |R| - z_1$. For these two disjoint sets we have $\hat{Q} \cap \hat{R} = \emptyset$. With all the states in $q' \in \hat{R}$, make a second tuple of arbitrary arrangement, say $\eta' = (q'_0, \cdots, q'_{z'-1})$, and form a z_2-side Latin square, LS_2, in an arbitrary way.

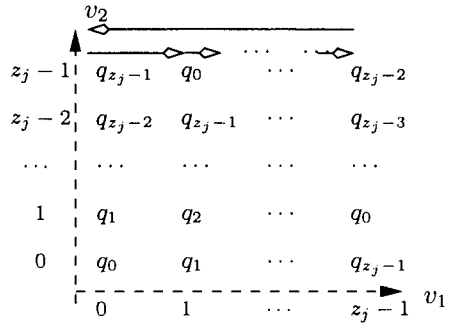


Figure B.1: Latin square LS_1 corresponds to $\eta_\alpha^{i,1}$ and is used to construct an ALM which yields IUF for $\alpha \in \Sigma_{o,1}$.

Without loss of generality and for the ease of illustration, the rest of proof is depicted in Fig. B.2 for the case of two isolated tuples $\eta_\alpha^{1,1} = (r_0, r_1, r_2)$ and $\eta' = (r_3, r_4)$, corresponding to Latin squares LS_1 and LS_2, respectively, which were just described above (not shown). Corresponding to these two tuples, two Latin squares of sides 3 and 2 are constructed, respectively. Then since the least common multiple of $z_1 = 3$ and $z_2 = 2$ is 6, two (respectively, three) copies of the first (respectively, the second) square are put side by side to make two rectangles of length 6 and widths 3 and 2, that are called rectangles 1 and 2, respectively (see Fig. B.2-a). Observe that each rectangle inherits its "white arrows" from its corresponding Latin square and that the while arrows for the two rectangles are the same. Then put one rectangle on top of the other and call the result, rectangle 3, as shown in part (b). Each column of rectangle 3, which is of size $2 + 3 = 5$ by 6, has exactly one copy of each state in it. As white-head arrows show, the updating function associated with α which is computed using this rectangle, would be independent of the row number. However, there are symbols which do not appear in some rows of rectangle 3 and their presence in each row and column is necessary to construct an

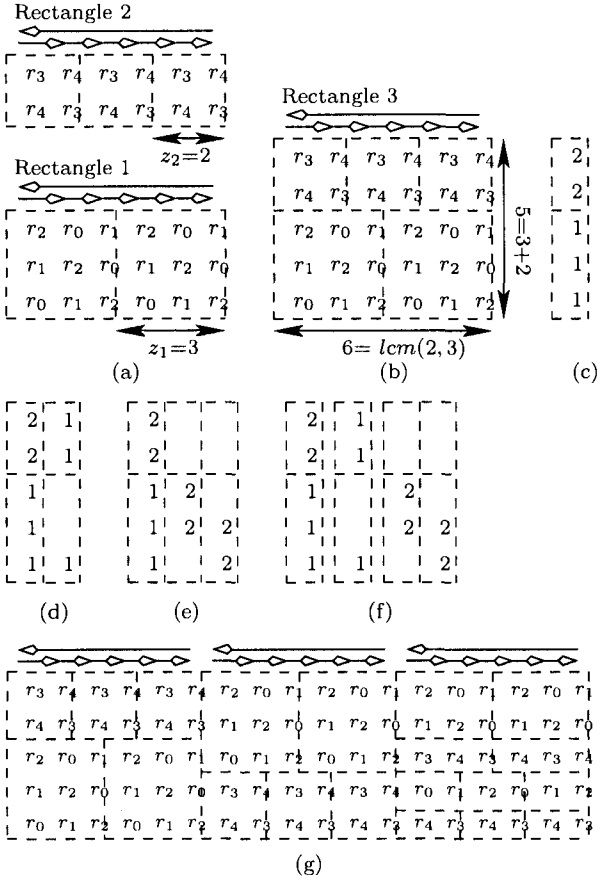


Figure B.2: Construction of an ALM yielding IUFs from Latin squares corresponding to disjoint tuples. (a) Construction of a rectangle of length 6 from Latin squares of sides 3 and 2. (b) Forming the rectangle of size 5 by 6 by putting one rectangle on top of the other one. (c) A compact representation of rectangle 3. (d) Distributing the rows of rectangle 1. (e) Distributing the rows of rectangle 2. (f) The general form of the rectangle used to build an ALM yielding IUF for α. (g) A reduced design.

ALM. Therefore, our next job is to accommodate these missing symbols in the above-mentioned columns.

To provide a copy of each state in every row, we put "sufficient" number of copies of rectangle 3 side by side and "distribute" its rows so that each row appears at least once at each $v_2 = j$, for all $j \in \{0, 1, \cdots, 4\}$. In each copy, α-labeled transitions follow the pattern represented by the white-head arrows in the original rectangle 3. We show that the "sufficient" number of copies is at most $\lceil \frac{|R|}{z_1} \rceil + \lceil \frac{|R|}{z_2} \rceil - (1+1)$, where $\lceil . \rceil$ denotes the *ceiling* function. Towards this end, let us number each row in rectangle 3 by the index of its associated Latin square. As a result, rectangle 3 would be represented as in part (c) which means that elements of $\eta_\alpha^{1,1}$ (respectively, η') exist in rows 1 to 3 (respectively, 4 and 5). Observe in part (d) (respectively, (e)) that to provide all $3 + 2 = 5$ rows with elements of $\eta_\alpha^{1,1}$ (respectively, η') by changing the order of rows in rectangle 3, we need to put at most $\lceil \frac{5}{3} \rceil = 2$ (respectively, $\lceil \frac{5}{2} \rceil = 3$) rearranged copies of rectangle 3, side by side. Notice that we have done this distribution of rows separately for each index 1 and 2 in parts (d) and (e), which is not necessarily the most efficient way of doing this. Also, notice that as long as distribution of index 1 (respectively, 2) is concerned in part (d) (respectively, (e)), the empty rows might be filled with arbitrary indices, where in general there are more than 2 indices. Considering the fact that the first copy of the original rectangle 3 is common to parts (e) and (f), we would need to bring $\lceil \frac{5}{3} \rceil - 1 + \lceil \frac{5}{2} \rceil - 1$ rearranged copies of rectangle 3 and put them side by side, as shown in part (f).

In general, if the z is the least common multiple of z_1 and z_2, rectangles 3 would have z columns and $z_1 + z_2$ rows and $\lceil \frac{|R|}{z_1} \rceil + \lceil \frac{|R|}{z_2} \rceil - (1 + 1)$ copies of it is required (at most) to build the desired ALM. However, the number of copies can be reduced by distributing both indices at the same time (and

not separately). For the current example, a reduced design is shown in part (g) which requires only two extra copies of rectangle 3 (rather than 3 in part (f)). For both designs in parts (f) and (g), notice the white-head arrows which represent the α-labeled transitions in each copy and induce a row-number independent map everywhere in the rectangle, as required by an IUF. This latter Latin square defines an ALM which yields IUF for α.

If $|I| > 2$, then first z_1- and z_2-side Latin hypercubes for $\hat{Q}$ and $\hat{R}$ are formed, respectively. The hyper-rectangle 3 would be of size $(z_1 + z_2)^{n-1} \times z$, and $\lceil \frac{|R|}{z_1} \rceil + \lceil \frac{|R|}{z_2} \rceil - 2$ copies of this hyper-rectangle are required, as before. ■

Proof of Corollary 5.5

Let $J_\alpha + J_\beta = m$ and number the subset of R associated with $\eta_\alpha^{i,d}$ (respectively, $\eta_\beta^{i,d'}$) as $\hat{Q}_d$ (respectively, $\hat{Q}_{J_\alpha + d'}$). Assume that these sets have respectively, $z_1, \cdots, z_m$ elements. Since every $\eta_\alpha^{i,d} \in \Delta_\alpha^i$ is isolated, by Definition 5.30, we have

$$\forall q \in R,\ q' \in \hat{Q}_d.\ \ q' = \xi(q, \alpha) \implies q \in \hat{Q}_d.$$

$$\implies (\forall d_1, d_2 \in \{1, \cdots, J_\alpha\}.\ d_1 \neq d_2 \implies \hat{Q}_{d_1} \cap \hat{Q}_{d_2} = \emptyset),$$

which holds by Definition 5.29. Similarly, we have

$$\implies (\forall d_1', d_2' \in \{1, \cdots, J_\beta\}.\ d_1' \neq d_2' \implies \hat{Q}_{d_1'} \cap \hat{Q}_{d_2'} = \emptyset).$$

Also by 3) above, every pair of sets from $\{\hat{Q}_1, \cdots, \hat{Q}_{J_\alpha}\}$ and $\{\hat{Q}_{J_\alpha+1}, \cdots, \hat{Q}_m\}$ are mutually disjoint. The above three results imply that every two sets i n $\{\hat{Q}_1, \cdots, \hat{Q}_{J_m}\}$ are mutually disjoint. Let $\hat{R} = R \setminus \bigcup_{j \in \{1, \cdots, m\}} \hat{Q}_j$ and assume it has z_{m+1} elements. Following the proof of Proposition 5.8, form $m + 1$ Latin hypercubes of sides $z_1, \cdots, z_m, z_{m+1}$. Let z be the least common multiple of $z_1, \cdots, z_m, z_{m+1}$. For $j \in \{1, \cdots, m + 1\}$, put $\frac{z}{z_j}$ copies of the jth Latin hypercube to build the jth hyper-rectangle of size $(z_j)^{n-1} \times z$. Put these $m + 1$ hyper-rectangles on top of each other to form a hyper-rectangle of size $(z_1 + \cdots + z_m)^{n-1} \times z)$. Provide (at most) $\lceil \frac{|R|}{z_1} \rceil + \cdots + \lceil \frac{|R|}{z_{m+1}} \rceil - (m+1)$ copies

of this last rectangle to form the final hyper-rectangle which yields the desired ALM. ■

Appendix C

Details of computations in Chapter 6

This appendix provides the detailed computations which support the results of Chapter 6. These computations are done in TCT software tool for Windows $XP^{®}$ [90].

C.0.1 Labeling of events

In TCT, controllable and uncontrollable events are represented by odd and even numbers, respectively. Using this convention, Table C.1 assigns 3 digit numbers to each event in Table 6.1, where the first digit is borrowed from the node number in Table 6.2. For example, RA^{e}_{DR2} is assigned 213, where the leftmost digit indicates this event belongs to **TN** and the right digit, being odd, indicates that the event is controllable.

Table C.1: Event numbering used in TCT

Event	S_{SD}	f		RA_{TN}	RA^e_{TN}			T_{SD}		
Number	101	103		107	109			102		
Event		A_{TN}	A^e_{TN}	RA_{DR1}	RA^e_{DR1}	RA_{DR2}	RA^e_{DR2}	T_{TN}		
Number		203	205	207	209	211	213	202		
Event	S_{DR1}	A_{DR1}	A^e_{DR1}	RA_{RN1}	RA^e_{RN1}	RA_{RN2}	RA^e_{RN2}	T_{DR1}	R_{DR1}	R^e_{DR1}
Number	301	303	305	307	309	311	313	302	304	306
Event	S_{DR2}	A_{DR2}	A^e_{DR2}	RA_{RN3}	RA^e_{RN3}			T_{DR2}	R_{DR2}	R^e_{DR2}
Number	401	403	405	407	409			402	404	406
Event		A_{RN1}	A^e_{RN1}	R_{RN1}	R^e_{RN1}			T_{RN1}		
Number		503	505	507	509			502		
Event		A_{RN2}	A^e_{RN2}	R_{RN2}	R^e_{RN2}			T_{RN2}		
Number		603	605	607	609			602		
Event		A_{RN3}	A^e_{RN3}	R_{RN3}	R^e_{RN3}			T_{RN3}		
Number		703	705	707	709			702		

C.0.2 Network modeling in TCT

Models of control nodes are shown in Fig. C.1 to Fig. C.7 and models of supervisors $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4, \mathbf{S}_5$, and $\mathbf{S}_6$ are shown in Fig. C.8 to Fig. C.13. These figures illustrate the standard output files which are generated by TCT. Each figure indicates the name of the control nodes[1], its states, initial state, mark state(s), number of transitions, and the list of transitions[2].

C.0.3 Application of supervisory control theory using TCT

This subsection lists the TCT code of commands which are written to apply RW supervisory control theory for the network and its supervisors. In the following these commands are explained as they implement the theory, where we assume that $k \in \{1, 2, 3, 4, 5, 6\}$.

Building the network model: Table C.2 lists the commands to generate the model of the network, which is obtained as the synchronous product (command "Sync") of the models of the seven control nodes (lines 1 to 6). Since the synchronous product $B = Sync(A_1, A_2)$ is computed for two automata A_1 and A_2, the model of the plant is obtained in 6 steps. In each line, the number of states and transitions of the resulting automaton is shown as (no. of states, no. of transitions). These two numbers which are as high as $(24300, 575100)$ for the plant, called PLANTMIN, are reduced to the pair $(18225, 431325)$ in line 7 upon the application of the "Minstate" command, which computes the minimal automaton.

Checking the existence of a supervisor for each specification: For

[1]In TCT models, the index of a node or supervisor appears in front of its name, rather than being a subscript.

[2]There is no *vocal states* [3] in this design.

Table C.2: The code to generate the whole network.

1	RN1RN2 = Sync(RN1,RN2) (9,30) Blocked_events = None
2	RN1RN2RN3 = Sync(RN3,RN1RN2) (27,135) Blocked_events =None
3	DR1DR2 = Sync(DR1,DR2) (60,720) Blocked_events = None
4	SDTN = Sync(SD,TN) (15,100) Blocked_events = None
5	SDTNDR1DR2 = Sync(SDTN,DR1DR2) (900,16800) Blocked_events = None
6	PLANT = Sync(SDTNDR1DR2,RN1RN2RN3) (24300,575100) Blocked_events = None
7	PLANTMIN = Minstate(PLANT) (18225,431325)

every specification $\mathbf{S}_k$, a supervisor, which enforces it, exists if and only if the intersection of the plant's behavior and the specification is controllable with respect to the plant. "Meet" is the TCT command which computes the intersection of (the languages of) two automata. In lines 1 to 6 of Table C.3, the intersection of (the languages of) each specification $\mathbf{S}_k$ and plant "PLANTMIN" is obtained and called as "SkCAPN."[3] Next, controllability of each such intersection is verified with respect to the (minimal automaton of the) plant using command "Condat," whose result appears at the end of each command line and, if this result turns to be "Uncontrollable," the illegal transitions leading to uncontrollability are listed in the output file at the left side of each assignment. Unfortunately, the sizes of the automata are too large for TCT to perform this operation[4]. Instead, the following argument is used to verify the existence of supervisors using smaller languages.

- In effect, each supervisor enforces the specification for the behavior of 2 control nodes. In other words, as longs as the specification is concerned,

[3]The pair of two numbers appearing at the end of each line indicates the number of states and transitions of the resulting automaton.

[4]Whereas in TCT the maximum state size depend on each application, as a rule of thumb, for procedure "Supcon" to compute the centralized supervisor, "the product of state sizes of plant and specification DESs should not exceed a few hundred thousand" [91].

other control nodes may appear as a single-state automata. Accordingly, to verify the existence of each supervisor, first a "sub-plant" is formed as the synchronous product of the two[5] control nodes for which the supervisor determines the joint behavior. For example, $\mathbf{S}_1$ enforces a specification for $\mathbf{RN}_1$ and $\mathbf{DR}_1$.

Accordingly, the following sub-plant are defined[6].

- To verify the existence of $\mathbf{S}_1, \mathbf{S}_2$, and $\mathbf{S}_4$, a sub-plant is defined as the synchronous product of **TN**, $\mathbf{DR}_1$, $\mathbf{RN}_1$, and $\mathbf{RN}_2$. This is called "TNLEFT" and is computed in line 7 to 10 of Table C.3.

- To verify the existence of $\mathbf{S}_3$ and $\mathbf{S}_5$, a sub-plant is defined as the synchronous product of **TN**, $\mathbf{DR}_2$, and $\mathbf{RN}_3$. This is called "TNRIGHT" and is computed in line 17 to 19 of Table C.3.

- To verify the existence of $\mathbf{S}_6$, a sub-plant is defined as the synchronous product of **SD**, **TN**, $\mathbf{DR}_1$, and $\mathbf{DR}_2$. This is called "SDTNCH" and is computed in line 24 to 27 of Table C.3.

Next, the intersection of each supervisor and its associated sub-plant is computed, respectively, 11 to 13, 20 to 21, and 28 to 30. The last step includes the application of "Condat" command to verify the controllability of each intersection with respect to its associated sub-plant. This step is shown, respectively, in lines, 14 to 16, 22 to 23, and 31 to 33. Observe that all intersections are controllable, implying that there exists a supervisor enforcing each.

[5]It should be clear that if more than two nodes are considered in the product, the computations are still valid.

[6]Although it is possible to define other subplants, the ones computed are later used for the verification of the mutual nonconflictingness of supervisors in the last part of the computations.

Table C.3: The code to verify the existence of a controller for each specification.

1	S1CAPN = Meet(S1,PLANTMIN) (54675,1196775)
2	S2CAPN = Meet(S2,PLANTMIN) (54675,1196775)
3	S3CAPN = Meet(S3,PLANTMIN) (54675,1196775)
4	S4CAPN = Meet(S4,PLANTMIN) (54675,1212975)
5	S5CAPN = Meet(S5,PLANTMIN) (54675,1206495)
6	S6CAPN = Meet(S6,PLANTMIN) (54675,1255095)
7	DR1RN1 = Sync(DR1,RN1) (30,260) Blocked_events = None
8	DR1RN1RN2 = Sync(DR1RN1,RN2) (90,930) Blocked_events = None
9	DR1CH = Sync(DR1RN1RN2,ALL) (90,3180) Blocked_events = None
10	TNLEFT = Sync(TN,DR1CH) (450,15000) Blocked_events = None
11	TNLEFTS4 = Meet(S4,TNLEFT) (1350,42930)
12	TNLEFTS1 = Meet(S1,TNLEFT) (1350,42600)
13	TNLEFTS2 = Meet(S2,TNLEFT) (1350,42600)
14	SMS1DAT = Condat(TNLEFT,TNLEFTS1) Controllable.
15	SMS2DAT = Condat(TNLEFT,TNLEFTS2) Controllable.
16	SMS4DAT = Condat(TNLEFT,TNLEFTS4) Controllable.
17	DR2RN3 = Sync(DR2,RN3) (18,120) Blocked_events = None
18	DR2CH = Sync(DR2RN3,ALL) (18,696) Blocked_events = None
19	TNRIGHT = Sync(DR2CH,TN) (90,3300) Blocked_events = None
20	TNRIGHTS3 = Meet(TNRIGHT,S3) (270,9420)
21	TNRIGHTS5 = Meet(TNRIGHT,S5) (270,9450)
22	SMS3DAT = Condat(TNRIGHT,TNRIGHTS3) Controllable.
23	SMS5DAT = Condat(TNRIGHT,TNRIGHTS5) Controllable.
24	SDTN2 = Sync(SD,TN) (15,100) Blocked_events = None
25	SDTNDR1 = Sync(SDTN2,DR1) (150,2050) Blocked_events = None
26	SDTNDR1DR2 = Sync(SDTNDR1,DR2) (900,16800) Blocked_events = None
27	SDTNCH = Sync(SDTNDR1DR2,ALL) (900,30300) Blocked_events = None
28	SDTNCHS4 = Meet(SDTNCH,S4) (2700,86760)
29	SDTNCHS5 = Meet(SDTNCH,S5) (2700,86400)
30	SDTNCHS6 = Meet(SDTNCH,S6) (2700,88980)
31	SMS4DAT2 = Condat(SDTNCH,SDTNCHS4) Controllable.
32	SMS5DAT2 = Condat(SDTNCH,SDTNCHS5) Controllable.
33	SMS6DAT = Condat(SDTNCH,SDTNCHS6) Controllable.

Checking if each supervisor is nonblocking: Table C.4 lists the code to verify that every specification is nonblocking. To this end, for each specification, first its trim part, i.e. reachable and coreachable part, is obtained using the command "Trim". This step can be seen in lines 1 to 6. Then it is verified that if this trim part is isomorphic to the specification, using the command "Isomorph," in lines 7 to 12. The "true" answer to this test confirms that the specification is reachable and coreachable, where the latter is equivalent to being nonblocking.

Table C.4: The code to verify that each specification is nonblocking and controllable.

1	S1T = Trim(S1) (3,127)
2	S2T = Trim(S2) (3,127)
3	S3T = Trim(S3) (3,127)
4	S4T = Trim(S4) (3,127)
5	S5T = Trim(S5) (3,127)
6	S6T = Trim(S6) (3,127)
7	true = Isomorph(S1T,S1;identity)
8	true = Isomorph(S2T,S2;identity)
9	true = Isomorph(S3T,S3;identity)
10	true = Isomorph(S4T,S4;identity)
11	true = Isomorph(S5T,S5;identity)
12	true = Isomorph(S6T,S6;identity)

Checking if each supervisor is controllable with respect to the plant: As mentioned before, the command to check the controllability of a specification with respect to a plant is "Condat." Shown in lines 1 to 6 of Table C.5, the result of this command appears at the end of the line of each command and, if the specification is uncontrollable, the transitions leading to uncontrollability are listed in the file which is the left-side of the assignment. Observe that in lines 1 to 6, all specifications are declared controllable with respect to plant

"PLANTMIN."

Table C.5: The code to verify that each specification is controllable.

1	S1DAT = Condat(PLANTMIN,S1) Controllable
2	S2DAT = Condat(PLANTMIN,S2) Controllable
3	S3DAT = Condat(PLANTMIN,S3) Controllable
4	S4DAT = Condat(PLANTMIN,S4) Controllable
5	S5DAT = Condat(PLANTMIN,S5) Controllable
6	S6DAT = Condat(PLANTMIN,S6) Controllable

Checking if each supervisor's language is nonconflicting with respect to the network: Two languages A and B, defined over the same alphabet, are nonconflicting (with respect to each other) if and only if $\overline{A \cap B} = \overline{A} \cap \overline{B}$, where $\overline{A}$ and $\overline{B}$ are prefix-closure of A and B, respectively. Command "Meet" computes the intersection of (the automata of) two languages. Also to compute the prefix-closure of a language it is enough to mark all the states of its associated automaton using command "Edit." The code to do the nonconflicting test is shown in Table C.6. In lines 1 to 6 first the intersection of every supervisor $\mathbf{S}_k$'s language and plant "PLANTMIN" is computed as "SkCAPN." Next, in lines 7 to 12 the prefix-closure of each such intersection is computed as "SkCAPNBAR," providing the left side of the nonconflicting test, i.e. $\overline{L(\mathbf{S}_k) \cap L(\mathbf{N})}$. To compte the right side of the test, i.e. $\overline{L(\mathbf{S}_k)} \cap \overline{L(\mathbf{N})}$, first in lines 13 to 18 the prefix-closure of each $\mathbf{S}_k$ is obtained and called "SkBAR" and in line 19 this prefix-closure is obtained for the plant and called "PLANTMINBAR."[7] Lines 20 to 25 compute the intersection of each "SkBAR" and "PLANTMINBAR," called "SkBARNBAR." This is what needed for the right side of the text. Finally the equivalence of each "SkCAPNBAR" and "SkBARNBAR" is verified

[7]Recall that "PLANTMIN" is the minimal automaton of plant "PLANT" and obviously is behaviorally equivalent to it.

using command "Isomorph" in lines 26 to 31.

Verifying that the supervisors are mutually nonconflicting: To verify the nonconflictingness of every two supervisors, first the closed-loop language induced by every supervisor is formed. In fact, this is the intersection of the supervisor and plant's languages, computed by command "Meet." This step is performed in lines 1 to 6 of Table C.7. Next the mutual intersection of every two such closed-loop languages are obtained by another "Meet" operation. Unfortunately, this step cannot be performed in TCT due to the very large state size of the automata. Assuming that this step could be performed, each language in this part should be compared to its counterpart, which is the closed-loop language induced by the synchronous supervision of the two supervisors. The latter language is obtained by computing first the synchronous product of the two supervisors and then the meet of the result and the plant. These first step is performed in lines 7 to 21 and the second step is illustrated in lines 22 to 36. Each closed-loop language obtained in this part should be compared to its counterpart using command "Isomorph." However, the size of the languages are too big for TCT to check the isomorphism. Instead, the following argument is used to verify that the supervisors are mutually nonconflicting using smaller languages.

1. Two supervisors which enforce two specifications for two separate groups of nodes are nonconflicting. For example, $\mathbf{S}_1$ enforces a specification for $\mathbf{RN}_1$ and $\mathbf{DR}_1$ and $\mathbf{S}_6$ enforces a specification between **SD** and **TN**, thus these two supervisors are nonconflicting. The reason is that a conflict happens when two specifications are imposed on the same resource separately.

2. Following the previous argument, nonconflictingness should be verified

Table C.6: The code to verify that each specification is nonconflicting with respect to the network.

1	S1CAPN = Meet(S1,PLANTMIN) (54675,1196775)
2	S2CAPN = Meet(S2,PLANTMIN) (54675,1196775)
3	S3CAPN = Meet(S3,PLANTMIN) (54675,1196775)
4	S4CAPN = Meet(S4,PLANTMIN) (54675,1212975)
5	S5CAPN = Meet(S5,PLANTMIN) (54675,1206495)
6	S6CAPN = Meet(S6,PLANTMIN) (54675,1255095)
7	S1CAPNBAR = Edit(S1CAPN,[mark +[all]]) (54675,1196775)
8	S2CAPNBAR = Edit(S2CAPN,[mark +[all]]) (54675,1196775)
9	S3CAPNBAR = Edit(S3CAPN,[mark +[all]]) (54675,1196775)
10	S4CAPNBAR = Edit(S4CAPN,[mark +[all]]) (54675,1212975)
11	S5CAPNBAR = Edit(S5CAPN,[mark +[all]]) (54675,1206495)
12	S6CAPNBAR = Edit(S6CAPN,[mark +[all]]) (54675,1255095)
13	S1BAR = Edit(S1,[mark +[all]]) (3,127)
14	S2BAR = Edit(S2,[mark +[all]]) (3,127)
15	S3BAR = Edit(S3,[mark +[all]]) (3,127)
16	S4BAR = Edit(S4,[mark +[all]]) (3,127)
17	S5BAR = Edit(S5,[mark +[all]]) (3,127)
18	S6BAR = Edit(S6,[mark +[all]]) (3,127)
19	PLANTMINBAR = Edit(PLANTMIN,[mark +[all]]) (18225,431325)
20	S1BARNBAR = Meet(S1BAR,PLANTMINBAR) (54675,1196775)
21	S2BARNBAR = Meet(S2BAR,PLANTMINBAR) (54675,1196775)
22	S3BARNBAR = Meet(S3BAR,PLANTMINBAR) (54675,1196775)
23	S4BARNBAR = Meet(S4BAR,PLANTMINBAR) (54675,1212975)
24	S5BARNBAR = Meet(S5BAR,PLANTMINBAR) (54675,1206495)
25	S6BARNBAR = Meet(S6BAR,PLANTMINBAR) (54675,1255095)
26	true = Isomorph(S1BARNBAR,S1CAPNBAR;identity)
27	true = Isomorph(S2BARNBAR,S2CAPNBAR;identity)
28	true = Isomorph(S3BARNBAR,S3CAPNBAR;identity)
29	true = Isomorph(S4BARNBAR,S4CAPNBAR;identity)
30	true = Isomorph(S5BARNBAR,S5CAPNBAR;identity)
31	true = Isomorph(S6BARNBAR,S6CAPNBAR;identity)

only for the following pairs of supervisors.

$$(\mathbf{S}_1, \mathbf{S}_2), (\mathbf{S}_1, \mathbf{S}_4), (\mathbf{S}_2, \mathbf{S}_4), (\mathbf{S}_3, \mathbf{S}_5), (\mathbf{S}_4, \mathbf{S}_5), (\mathbf{S}_4, \mathbf{S}_6), (\mathbf{S}_5, \mathbf{S}_6)$$

To this end, for each pair, the control nodes whose events appear as the labels of state-changing transitions of at least one of the supervisors, are synchronized to form the "sub-plant," with respect to which mutual nonconflicting of the two supervisors is verified.

This leads to the following computations.

- To verify that $\mathbf{S}_1, \mathbf{S}_2$, and $\mathbf{S}_4$ are mutually nonconflicting, the sub-plant is defined as the synchronous product of **TN**, $\mathbf{DR}_1$, $\mathbf{RN}_1$, and $\mathbf{RN}_2$. This is called "TNLEFT" and is computed in line 1 to 4 of Table C.8.

- To verify that $\mathbf{S}_3$ and $\mathbf{S}_5$ are nonconflicting, the sub-plant is defined as the synchronous product of **TN**, $\mathbf{DR}_2$, and $\mathbf{RN}_3$. This is called "TNRIGHT" and is computed in line 23 to 25 of Table C.8.

- To verify that $\mathbf{S}_4, \mathbf{S}_5$, and $\mathbf{S}_6$ are mutually nonconflicting, the sub-plant is defined as the synchronous product of **SD**, **TN**, $\mathbf{DR}_1$, and $\mathbf{DR}_2$. This is called "SDTNCH" and is computed in line 1 to 4 of Table C.9.

After forming the sub-plants, in the next step, the closed-loop behavior, induced by each corresponding supervisor and also by the synchronous operation of each pair of supervisors, are computed, respectively, in lines 5 to 10, 26 to 28, of Table C.8, and 5 to 10 of Table C.9. Next, the prefix-closure of every such language is computed, respectively, in lines in lines 17 to 22, 32 to 33, of Table C.8, and 11 to 16 of Table C.9.

Next, the meet of each pair of closed-loop languages is computed, using "Meet," and compared to the prefix-closure of the closed-loop language

induced by the synchronous product of the two supervisor. These two steps are shown , respectively, in lines in lines 11 to 16, 29 to 31, of Table C.8, and 17 to 23 of Table C.9. It can be observed that all supervisors all mutually nonconflicting.

Table C.7: The code to verify that supervisors are mutually nonconflicting (Part 1).

1	S1CAPN = Meet(S1,PLANTMIN) (54675,1196775)
2	S2CAPN = Meet(S2,PLANTMIN) (54675,1196775)
3	S3CAPN = Meet(S3,PLANTMIN) (54675,1196775)
4	S4CAPN = Meet(S4,PLANTMIN) (54675,1212975)
5	S5CAPN = Meet(S5,PLANTMIN) (54675,1206495)
6	S6CAPN = Meet(S6,PLANTMIN) (54675,1255095)
7	S1S2 = Sync(S1,S2) (9,357) Blocked_events = None
8	S1S3 = Sync(S1,S3) (9,357) Blocked_events = None
9	S1S4 = Sync(S1,S4) (9,357) Blocked_events = None
10	S1S5 = Sync(S1,S5) (9,357) Blocked_events = None
11	S1S6 = Sync(S1,S6) (9,357) Blocked_events = None
12	S2S3 = Sync(S2,S3) (9,357) Blocked_events = None
13	S2S4 = Sync(S2,S4) (9,357) Blocked_events = None
14	S2S5 = Sync(S2,S5) (9,357) Blocked_events = None
15	S2S6 = Sync(S2,S6) (9,357) Blocked_events = None
16	S3S4 = Sync(S3,S4) (9,357) Blocked_events = None
17	S3S5 = Sync(S3,S5) (9,357) Blocked_events = None
18	S3S6 = Sync(S3,S6) (9,357) Blocked_events = None
19	S4S5 = Sync(S4,S5) (9,357) Blocked_events = None
20	S4S6 = Sync(S4,S6) (9,357) Blocked_events = None
21	S5S6 = Sync(S5,S6) (9,357) Blocked_events = None
22	S1S2CAPN = Meet(S1S2,PLANTMIN) (164025,3298725)
23	S1S3CAPN = Meet(S1S3,PLANTMIN) (164025,3298725)
24	S1S4CAPN = Meet(S1S4,PLANTMIN) (164025,3347325)
25	S1S5CAPN = Meet(S1S5,PLANTMIN) (164025,3327885)
26	S1S6CAPN = Meet(S1S6,PLANTMIN) (164025,3473685)
27	S2S3CAPN = Meet(S2S3,PLANTMIN) (164025,3298725)
28	S2S4CAPN = Meet(S2S4,PLANTMIN) (164025,3347325)
29	S2S5CAPN = Meet(S2S5,PLANTMIN) (164025,3327885)
30	S2S6CAPN = Meet(S2S6,PLANTMIN) (164025,3473685)
31	S3S4CAPN = Meet(S3S4,PLANTMIN) (164025,3347325)
32	S3S5CAPN = Meet(S3S5,PLANTMIN) (164025,3327885)
33	S3S6CAPN = Meet(S3S6,PLANTMIN) (164025,3473685)
34	S4S5CAPN = Meet(S4S5,PLANTMIN) (164025,3376485)
35	S4S6CAPN = Meet(S4S6,PLANTMIN) (164025,3522285)
36	S5S6CAPN = Meet(S5S6,PLANTMIN) (164025,3502845)

Table C.8: The code to verify that supervisors are mutually nonconflicting (Part 2).

	1 DR1RN1 = Sync(DR1,RN1) (30,260) Blocked_events = None
	2 DR1RN1RN2 = Sync(DR1RN1,RN2) (90,930) Blocked_events = None
	3 DR1CH = Sync(DR1RN1RN2,ALL) (90,3180) Blocked_events = None
	4 TNLEFT = Sync(TN,DR1CH) (450,15000) Blocked_events = None
	5 TNLEFTS4 = Meet(S4,TNLEFT) (1350,42930)
	6 TNLEFTS1 = Meet(S1,TNLEFT) (1350,42600)
	7 TNLEFTS2 = Meet(S2,TNLEFT) (1350,42600)
	8 TNLEFTS1S2 = Meet(S1S2,TNLEFT) (4050,120600)
	9 TNLEFTS1S4 = Meet(S1S4,TNLEFT) (4050,121590)
	10 TNLEFTS2S4 = Meet(S2S4,TNLEFT) (4050,121590)
	11 TNLEFTS1BAR = Edit(TNLEFTS1,[mark +[all]]) (1350,42600)
	12 TNLEFTS2BAR = Edit(TNLEFTS2,[mark +[all]]) (1350,42600)
	13 TNLEFTS4BAR = Edit(TNLEFTS4,[mark +[all]]) (1350,42930)
	14 TNLEFTS1S2BAR = Edit(TNLEFTS1S2,[mark +[all]]) (4050,120600)
	15 TNLEFTS1S4BAR = Edit(TNLEFTS1S4,[mark +[all]]) (4050,121590)
	16 TNLEFTS2S4BAR = Edit(TNLEFTS2S4,[mark +[all]]) (4050,121590)
	17 TNLEFTS1BS2B = Meet(TNLEFTS1BAR,TNLEFTS2BAR) (4050,120600)
	18 TNLEFTS1BS4B = Meet(TNLEFTS1BAR,TNLEFTS4BAR) (4050,121590)
	19 TNLEFTS2BS4B = Meet(TNLEFTS2BAR,TNLEFTS4BAR) (4050,121590)
	20 true = Isomorph(TNLEFTS1S2BAR,TNLEFTS1BS2B;identity)
	21 true = Isomorph(TNLEFTS1S4BAR,TNLEFTS1BS4B;identity)
	22 true = Isomorph(TNLEFTS2S4BAR,TNLEFTS2BS4B;identity)
	23 DR2RN3 = Sync(DR2,RN3) (18,120) Blocked_events = None
	24 DR2CH = Sync(DR2RN3,ALL) (18,696) Blocked_events = None
	25 TNRIGHT = Sync(DR2CH,TN) (90,3300) Blocked_events = None
	26 TNRIGHTS3 = Meet(TNRIGHT,S3) (270,9420)
	27 TNRIGHTS5 = Meet(TNRIGHT,S5) (270,9450)
	28 TNRIGHTS35 = Meet(TNRIGHT,S3S5) (810,26910)
	29 TNRIGHTS3BAR = Edit(TNRIGHTS3,[mark +[all]]) (270,9420)
	30 TNRIGHTS5BAR = Edit(TNRIGHTS5,[mark +[all]]) (270,9450)
	31 TNRIGHTS3S5BAR = Edit(TNRIGHTS3S5,[mark +[all]]) (810,26910)
	32 TNRIGHTS3BS5B = Meet(TNRIGHTS3BAR,TNRIGHTS5BAR) (810,26910)
	33 true = Isomorph(TNRIGHTS3BS5B,TNRIGHTS3S5BAR;identity)

Table C.9: The code to verify that supervisors are mutually nonconflicting (Part 3).

1	SDTN2 = Sync(SD,TN) (15,100) Blocked_events = None
2	SDTNDR1 = Sync(SDTN2,DR1) (150,2050) Blocked_events = None
3	SDTNDR1DR2 = Sync(SDTNDR1,DR2) (900,16800) Blocked_events = None
4	SDTNCH = Sync(SDTNDR1DR2,ALL) (900,30300) Blocked_events = None
5	SDTNCHS4 = Meet(SDTNCH,S4) (2700,86760)
6	SDTNCHS5 = Meet(SDTNCH,S5) (2700,86400)
7	SDTNCHS6 = Meet(SDTNCH,S6) (2700,88980)
8	SDTNCHS4S5 = Meet(SDTNCH,S4S5) (8100,246780)
9	SDTNCHS4S6 = Meet(SDTNCH,S4S6) (8100,254520)
10	SDTNCHS5S6 = Meet(SDTNCH,S5S6) (8100,253440)
11	SDTNCHS4BAR = Edit(SDTNCHS4,[mark +[all]]) (2700,86760)
12	SDTNCHS5BAR = Edit(SDTNCHS5,[mark +[all]]) (2700,86400)
13	SDTNCHS6BAR = Edit(SDTNCHS6,[mark +[all]]) (2700,88980)
14	SDTNCHS4S5BAR = Edit(SDTNCHS4S5,[mark +[all]]) (8100,246780)
15	SDTNCHS4S6BAR = Edit(SDTNCHS4S6,[mark +[all]]) (8100,254520)
16	SDTNCHS5S6BAR = Edit(SDTNCHS5S6,[mark +[all]]) (8100,253440)
17	SDTNCHS4BS5B = Meet(SDTNCHS4BAR,SDTNCHS5BAR) (8100,246780)
18	SDTNCHS5BS6B = Meet(SDTNCHS4BAR,SDTNCHS6BAR) (8100,254520)
19	SDTNCHS4BS6B = Meet(SDTNCHS4BAR,SDTNCHS6BAR) (8100,254520)
20	SDTNCHS5BS6B = Meet(SDTNCHS5BAR,SDTNCHS6BAR) (8100,253440)
21	true = Isomorph(SDTNCHS4BS5B,SDTNCHS4S5BAR;identity)
22	true = Isomorph(SDTNCHS4BS6B,SDTNCHS4S6BAR;identity)
23	true = Isomorph(SDTNCHS5BS6B,SDTNCHS5S6BAR;identity)

```
SD    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 5

transitions:

[    0,103,    1]  [    1,101,    2]  [    2,102,    1]  [    2,107,    0]
[    2,109,    1]
```

Figure C.1: TCT model of control node **SD**

```
TN    # states: 5    state set: 0 ... 4    initial state: 0

marker states:              0

vocal states: none


# transitions: 25

transitions:

[    0,202,    4]  [    0,207,    1]  [    0,209,    4]  [    0,211,    2]
[    0,213,    4]  [    1,202,    4]  [    1,207,    1]  [    1,209,    1]
[    1,211,    3]  [    1,213,    4]  [    2,202,    4]  [    2,207,    3]
[    2,209,    4]  [    2,211,    2]  [    2,213,    2]  [    3,203,    0]
[    3,207,    3]  [    3,209,    3]  [    3,211,    3]  [    3,213,    3]
[    4,205,    0]  [    4,207,    4]  [    4,209,    4]  [    4,211,    4]
[    4,213,    4]
```

Figure C.2: TCT model of control node **TN**

```
DR1    # states: 10    state set: 0 ... 9    initial state: 0

marker states:              0

vocal states: none


# transitions: 70

transitions:

[    0,302,    4]  [    0,304,    5]  [    0,306,    0]  [    0,307,    1]
[    0,309,    4]  [    0,311,    2]  [    0,313,    4]  [    1,302,    4]
[    1,304,    6]  [    1,306,    1]  [    1,307,    1]  [    1,309,    1]
[    1,311,    3]  [    1,313,    4]  [    2,302,    4]  [    2,304,    7]
[    2,306,    2]  [    2,307,    3]  [    2,309,    4]  [    2,311,    2]
[    2,313,    2]  [    3,303,    0]  [    3,304,    8]  [    3,306,    3]
[    3,307,    3]  [    3,309,    3]  [    3,311,    3]  [    3,313,    3]
[    4,304,    9]  [    4,305,    0]  [    4,306,    4]  [    4,307,    4]
[    4,309,    4]  [    4,311,    4]  [    4,313,    4]  [    5,302,    9]
[    5,304,    5]  [    5,306,    5]  [    5,307,    6]  [    5,309,    9]
[    5,311,    7]  [    5,313,    9]  [    6,302,    9]  [    6,304,    6]
[    6,306,    6]  [    6,307,    6]  [    6,309,    6]  [    6,311,    8]
[    6,313,    9]  [    7,302,    9]  [    7,304,    7]  [    7,306,    7]
[    7,307,    8]  [    7,309,    9]  [    7,311,    7]  [    7,313,    7]
[    8,303,    0]  [    8,304,    8]  [    8,306,    8]  [    8,307,    8]
[    8,309,    8]  [    8,311,    8]  [    8,313,    8]  [    9,301,    5]
[    9,304,    9]  [    9,306,    9]  [    9,307,    9]  [    9,309,    9]
[    9,311,    9]  [    9,313,    9]
```

Figure C.3: TCT model of control node **DR1**

```
DR2    # states: 6    state set: 0 ... 5    initial state: 0

marker states:              0

vocal states: none


# transitions: 30

transitions:

[    0,402,    2]  [    0,404,    3]  [    0,406,    0]  [    0,407,    1]
[    0,409,    2]  [    1,403,    0]  [    1,404,    4]  [    1,406,    1]
[    1,407,    1]  [    1,409,    1]  [    2,404,    5]  [    2,405,    0]
[    2,406,    2]  [    2,407,    2]  [    2,409,    2]  [    3,402,    5]
[    3,404,    3]  [    3,406,    3]  [    3,407,    4]  [    3,409,    5]
[    4,403,    0]  [    4,404,    4]  [    4,406,    4]  [    4,407,    4]
[    4,409,    4]  [    5,401,    3]  [    5,404,    5]  [    5,406,    5]
[    5,407,    5]  [    5,409,    5]
```

Figure C.4: TCT model of control node **DR2**

```
RN1    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none

# transitions: 5

transitions:

[    0,502,    2]  [    0,507,    1]  [    0,509,    2]  [    1,503,    0]
[    2,505,    0]
```

Figure C.5: TCT model of control node **RN1**

```
RN2    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none

# transitions: 5

transitions:

[    0,602,    2]  [    0,607,    1]  [    0,609,    2]  [    1,603,    0]
[    2,605,    0]
```

Figure C.6: TCT model of control node **RN2**

```
RN3    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none

# transitions: 5

transitions:

[    0,702,    2]  [    0,707,    1]  [    0,709,    2]  [    1,703,    0]
[    2,705,    0]
```

Figure C.7: TCT model of control node **RN3**

```
S1    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,107,    0]
[    0,109,    0]  [    0,202,    0]  [    0,203,    0]  [    0,205,    0]
[    0,207,    0]  [    0,209,    0]  [    0,211,    0]  [    0,213,    0]
[    0,301,    0]  [    0,302,    0]  [    0,303,    0]  [    0,304,    0]
[    0,305,    0]  [    0,306,    0]  [    0,311,    0]  [    0,313,    0]
[    0,401,    0]  [    0,402,    0]  [    0,403,    0]  [    0,404,    0]
[    0,405,    0]  [    0,406,    0]  [    0,407,    0]  [    0,409,    0]
[    0,502,    0]  [    0,503,    1]  [    0,505,    2]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    0]  [    0,605,    0]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    0]
[    0,705,    0]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    1]  [    1,109,    1]
[    1,202,    1]  [    1,203,    1]  [    1,205,    1]  [    1,207,    1]
[    1,209,    1]  [    1,211,    1]  [    1,213,    1]  [    1,301,    1]
[    1,302,    1]  [    1,303,    1]  [    1,304,    1]  [    1,305,    1]
[    1,306,    1]  [    1,307,    0]  [    1,311,    1]  [    1,313,    1]
[    1,401,    1]  [    1,402,    1]  [    1,403,    1]  [    1,404,    1]
[    1,405,    1]  [    1,406,    1]  [    1,407,    1]  [    1,409,    1]
[    1,502,    1]  [    1,507,    1]  [    1,509,    1]  [    1,602,    1]
[    1,603,    1]  [    1,605,    1]  [    1,607,    1]  [    1,609,    1]
[    1,702,    1]  [    1,703,    1]  [    1,705,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,107,    2]  [    2,109,    2]  [    2,202,    2]  [    2,203,    2]
[    2,205,    2]  [    2,207,    2]  [    2,209,    2]  [    2,211,    2]
[    2,213,    2]  [    2,301,    2]  [    2,302,    2]  [    2,303,    2]
[    2,304,    2]  [    2,305,    2]  [    2,306,    2]  [    2,309,    0]
[    2,311,    2]  [    2,313,    2]  [    2,401,    2]  [    2,402,    2]
[    2,403,    2]  [    2,404,    2]  [    2,405,    2]  [    2,406,    2]
[    2,407,    2]  [    2,409,    2]  [    2,502,    2]  [    2,507,    2]
[    2,509,    2]  [    2,602,    2]  [    2,603,    2]  [    2,605,    2]
[    2,607,    2]  [    2,609,    2]  [    2,702,    2]  [    2,703,    2]
[    2,705,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.8: TCT model of specification $\mathbf{S}_1$

```
S2    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,107,    0]
[    0,109,    0]  [    0,202,    0]  [    0,203,    0]  [    0,205,    0]
[    0,207,    0]  [    0,209,    0]  [    0,211,    0]  [    0,213,    0]
[    0,301,    0]  [    0,302,    0]  [    0,303,    0]  [    0,304,    0]
[    0,305,    0]  [    0,306,    0]  [    0,307,    0]  [    0,309,    0]
[    0,401,    0]  [    0,402,    0]  [    0,403,    0]  [    0,404,    0]
[    0,405,    0]  [    0,406,    0]  [    0,407,    0]  [    0,409,    0]
[    0,502,    0]  [    0,503,    0]  [    0,505,    0]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    1]  [    0,605,    2]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    0]
[    0,705,    0]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    1]  [    1,109,    1]
[    1,202,    1]  [    1,203,    1]  [    1,205,    1]  [    1,207,    1]
[    1,209,    1]  [    1,211,    1]  [    1,213,    1]  [    1,301,    1]
[    1,302,    1]  [    1,303,    1]  [    1,304,    1]  [    1,305,    1]
[    1,306,    1]  [    1,307,    1]  [    1,309,    1]  [    1,311,    0]
[    1,401,    1]  [    1,402,    1]  [    1,403,    1]  [    1,404,    1]
[    1,405,    1]  [    1,406,    1]  [    1,407,    1]  [    1,409,    1]
[    1,502,    1]  [    1,503,    1]  [    1,505,    1]  [    1,507,    1]
[    1,509,    1]  [    1,602,    1]  [    1,607,    1]  [    1,609,    1]
[    1,702,    1]  [    1,703,    1]  [    1,705,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,107,    2]  [    2,109,    2]  [    2,202,    2]  [    2,203,    2]
[    2,205,    2]  [    2,207,    2]  [    2,209,    2]  [    2,211,    2]
[    2,213,    2]  [    2,301,    2]  [    2,302,    2]  [    2,303,    2]
[    2,304,    2]  [    2,305,    2]  [    2,306,    2]  [    2,307,    2]
[    2,309,    2]  [    2,313,    0]  [    2,401,    2]  [    2,402,    2]
[    2,403,    2]  [    2,404,    2]  [    2,405,    2]  [    2,406,    2]
[    2,407,    2]  [    2,409,    2]  [    2,502,    2]  [    2,503,    2]
[    2,505,    2]  [    2,507,    2]  [    2,509,    2]  [    2,602,    2]
[    2,607,    2]  [    2,609,    2]  [    2,702,    2]  [    2,703,    2]
[    2,705,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.9: TCT model of specification $\mathbf{S}_2$

```
S3    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,107,    0]
[    0,109,    0]  [    0,202,    0]  [    0,203,    0]  [    0,205,    0]
[    0,207,    0]  [    0,209,    0]  [    0,211,    0]  [    0,213,    0]
[    0,301,    0]  [    0,302,    0]  [    0,303,    0]  [    0,304,    0]
[    0,305,    0]  [    0,306,    0]  [    0,307,    0]  [    0,309,    0]
[    0,311,    0]  [    0,313,    0]  [    0,401,    0]  [    0,402,    0]
[    0,403,    0]  [    0,404,    0]  [    0,405,    0]  [    0,406,    0]
[    0,502,    0]  [    0,503,    0]  [    0,505,    0]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    0]  [    0,605,    0]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    1]
[    0,705,    2]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    1]  [    1,109,    1]
[    1,202,    1]  [    1,203,    1]  [    1,205,    1]  [    1,207,    1]
[    1,209,    1]  [    1,211,    1]  [    1,213,    1]  [    1,301,    1]
[    1,302,    1]  [    1,303,    1]  [    1,304,    1]  [    1,305,    1]
[    1,306,    1]  [    1,307,    1]  [    1,309,    1]  [    1,311,    1]
[    1,313,    1]  [    1,401,    1]  [    1,402,    1]  [    1,403,    1]
[    1,404,    1]  [    1,405,    1]  [    1,406,    1]  [    1,407,    0]
[    1,502,    1]  [    1,503,    1]  [    1,505,    1]  [    1,507,    1]
[    1,509,    1]  [    1,602,    1]  [    1,603,    1]  [    1,605,    1]
[    1,607,    1]  [    1,609,    1]  [    1,702,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,107,    2]  [    2,109,    2]  [    2,202,    2]  [    2,203,    2]
[    2,205,    2]  [    2,207,    2]  [    2,209,    2]  [    2,211,    2]
[    2,213,    2]  [    2,301,    2]  [    2,302,    2]  [    2,303,    2]
[    2,304,    2]  [    2,305,    2]  [    2,306,    2]  [    2,307,    2]
[    2,309,    2]  [    2,311,    2]  [    2,313,    2]  [    2,401,    2]
[    2,402,    2]  [    2,403,    2]  [    2,404,    2]  [    2,405,    2]
[    2,406,    2]  [    2,409,    0]  [    2,502,    2]  [    2,503,    2]
[    2,505,    2]  [    2,507,    2]  [    2,509,    2]  [    2,602,    2]
[    2,603,    2]  [    2,605,    2]  [    2,607,    2]  [    2,609,    2]
[    2,702,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.10: TCT model of specification $\mathbf{S}_3$

```
S4    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,107,    0]
[    0,109,    0]  [    0,202,    0]  [    0,203,    0]  [    0,205,    0]
[    0,211,    0]  [    0,213,    0]  [    0,301,    0]  [    0,302,    0]
[    0,303,    1]  [    0,304,    0]  [    0,305,    2]  [    0,306,    0]
[    0,307,    0]  [    0,309,    0]  [    0,311,    0]  [    0,313,    0]
[    0,401,    0]  [    0,402,    0]  [    0,403,    0]  [    0,404,    0]
[    0,405,    0]  [    0,406,    0]  [    0,407,    0]  [    0,409,    0]
[    0,502,    0]  [    0,503,    0]  [    0,505,    0]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    0]  [    0,605,    0]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    0]
[    0,705,    0]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    1]  [    1,109,    1]
[    1,202,    1]  [    1,203,    1]  [    1,205,    1]  [    1,207,    0]
[    1,211,    1]  [    1,213,    1]  [    1,301,    1]  [    1,302,    1]
[    1,304,    1]  [    1,306,    1]  [    1,307,    1]  [    1,309,    1]
[    1,311,    1]  [    1,313,    1]  [    1,401,    1]  [    1,402,    1]
[    1,403,    1]  [    1,404,    1]  [    1,405,    1]  [    1,406,    1]
[    1,407,    1]  [    1,409,    1]  [    1,502,    1]  [    1,503,    1]
[    1,505,    1]  [    1,507,    1]  [    1,509,    1]  [    1,602,    1]
[    1,603,    1]  [    1,605,    1]  [    1,607,    1]  [    1,609,    1]
[    1,702,    1]  [    1,703,    1]  [    1,705,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,107,    2]  [    2,109,    2]  [    2,202,    2]  [    2,203,    2]
[    2,205,    2]  [    2,209,    0]  [    2,211,    2]  [    2,213,    2]
[    2,301,    2]  [    2,302,    2]  [    2,304,    2]  [    2,306,    2]
[    2,307,    2]  [    2,309,    2]  [    2,311,    2]  [    2,313,    2]
[    2,401,    2]  [    2,402,    2]  [    2,403,    2]  [    2,404,    2]
[    2,405,    2]  [    2,406,    2]  [    2,407,    2]  [    2,409,    2]
[    2,502,    2]  [    2,503,    2]  [    2,505,    2]  [    2,507,    2]
[    2,509,    2]  [    2,602,    2]  [    2,603,    2]  [    2,605,    2]
[    2,607,    2]  [    2,609,    2]  [    2,702,    2]  [    2,703,    2]
[    2,705,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.11: TCT model of specification $\mathbf{S}_4$

```
S5    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,107,    0]
[    0,109,    0]  [    0,202,    0]  [    0,203,    0]  [    0,205,    0]
[    0,207,    0]  [    0,209,    0]  [    0,301,    0]  [    0,302,    0]
[    0,303,    0]  [    0,304,    0]  [    0,305,    0]  [    0,306,    0]
[    0,307,    0]  [    0,309,    0]  [    0,311,    0]  [    0,313,    0]
[    0,401,    0]  [    0,402,    0]  [    0,403,    1]  [    0,404,    0]
[    0,405,    2]  [    0,406,    0]  [    0,407,    0]  [    0,409,    0]
[    0,502,    0]  [    0,503,    0]  [    0,505,    0]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    0]  [    0,605,    0]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    0]
[    0,705,    0]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    1]  [    1,109,    1]
[    1,202,    1]  [    1,203,    1]  [    1,205,    1]  [    1,207,    1]
[    1,209,    1]  [    1,211,    0]  [    1,301,    1]  [    1,302,    1]
[    1,303,    1]  [    1,304,    1]  [    1,305,    1]  [    1,306,    1]
[    1,307,    1]  [    1,309,    1]  [    1,311,    1]  [    1,313,    1]
[    1,401,    1]  [    1,402,    1]  [    1,404,    1]  [    1,406,    1]
[    1,407,    1]  [    1,409,    1]  [    1,502,    1]  [    1,503,    1]
[    1,505,    1]  [    1,507,    1]  [    1,509,    1]  [    1,602,    1]
[    1,603,    1]  [    1,605,    1]  [    1,607,    1]  [    1,609,    1]
[    1,702,    1]  [    1,703,    1]  [    1,705,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,107,    2]  [    2,109,    2]  [    2,202,    2]  [    2,203,    2]
[    2,205,    2]  [    2,207,    2]  [    2,209,    2]  [    2,213,    0]
[    2,301,    2]  [    2,302,    2]  [    2,303,    2]  [    2,304,    2]
[    2,305,    2]  [    2,306,    2]  [    2,307,    2]  [    2,309,    2]
[    2,311,    2]  [    2,313,    2]  [    2,401,    2]  [    2,402,    2]
[    2,404,    2]  [    2,406,    2]  [    2,407,    2]  [    2,409,    2]
[    2,502,    2]  [    2,503,    2]  [    2,505,    2]  [    2,507,    2]
[    2,509,    2]  [    2,602,    2]  [    2,603,    2]  [    2,605,    2]
[    2,607,    2]  [    2,609,    2]  [    2,702,    2]  [    2,703,    2]
[    2,705,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.12: TCT model of specification $\mathbf{S}_5$

```
S6    # states: 3    state set: 0 ... 2    initial state: 0

marker states:              0

vocal states: none


# transitions: 127

transitions:

[    0,101,    0]  [    0,102,    0]  [    0,103,    0]  [    0,202,    0]
[    0,203,    1]  [    0,205,    2]  [    0,207,    0]  [    0,209,    0]
[    0,211,    0]  [    0,213,    0]  [    0,301,    0]  [    0,302,    0]
[    0,303,    0]  [    0,304,    0]  [    0,305,    0]  [    0,306,    0]
[    0,307,    0]  [    0,309,    0]  [    0,311,    0]  [    0,313,    0]
[    0,401,    0]  [    0,402,    0]  [    0,403,    0]  [    0,404,    0]
[    0,405,    0]  [    0,406,    0]  [    0,407,    0]  [    0,409,    0]
[    0,502,    0]  [    0,503,    0]  [    0,505,    0]  [    0,507,    0]
[    0,509,    0]  [    0,602,    0]  [    0,603,    0]  [    0,605,    0]
[    0,607,    0]  [    0,609,    0]  [    0,702,    0]  [    0,703,    0]
[    0,705,    0]  [    0,707,    0]  [    0,709,    0]  [    1,101,    1]
[    1,102,    1]  [    1,103,    1]  [    1,107,    0]  [    1,202,    1]
[    1,207,    1]  [    1,209,    1]  [    1,211,    1]  [    1,213,    1]
[    1,301,    1]  [    1,302,    1]  [    1,303,    1]  [    1,304,    1]
[    1,305,    1]  [    1,306,    1]  [    1,307,    1]  [    1,309,    1]
[    1,311,    1]  [    1,313,    1]  [    1,401,    1]  [    1,402,    1]
[    1,403,    1]  [    1,404,    1]  [    1,405,    1]  [    1,406,    1]
[    1,407,    1]  [    1,409,    1]  [    1,502,    1]  [    1,503,    1]
[    1,505,    1]  [    1,507,    1]  [    1,509,    1]  [    1,602,    1]
[    1,603,    1]  [    1,605,    1]  [    1,607,    1]  [    1,609,    1]
[    1,702,    1]  [    1,703,    1]  [    1,705,    1]  [    1,707,    1]
[    1,709,    1]  [    2,101,    2]  [    2,102,    2]  [    2,103,    2]
[    2,109,    0]  [    2,202,    2]  [    2,207,    2]  [    2,209,    2]
[    2,211,    2]  [    2,213,    2]  [    2,301,    2]  [    2,302,    2]
[    2,303,    2]  [    2,304,    2]  [    2,305,    2]  [    2,306,    2]
[    2,307,    2]  [    2,309,    2]  [    2,311,    2]  [    2,313,    2]
[    2,401,    2]  [    2,402,    2]  [    2,403,    2]  [    2,404,    2]
[    2,405,    2]  [    2,406,    2]  [    2,407,    2]  [    2,409,    2]
[    2,502,    2]  [    2,503,    2]  [    2,505,    2]  [    2,507,    2]
[    2,509,    2]  [    2,602,    2]  [    2,603,    2]  [    2,605,    2]
[    2,607,    2]  [    2,609,    2]  [    2,702,    2]  [    2,703,    2]
[    2,705,    2]  [    2,707,    2]  [    2,709,    2]
```

Figure C.13: TCT model of specification $\mathbf{S}_6$