Discrete-event multiagent coordination : framework and algorithms

Pham, Manh Tung

2011

Pham, M. T. (2011). Discrete-event multiagent coordination : framework and algorithms.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/46271>

<https://doi.org/10.32657/10356/46271>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**DISCRETE-EVENT MULTIAGENT COORDINATION:
FRAMEWORK AND ALGORITHMS**

PHAM MANH TUNG

SCHOOL OF COMPUTER ENGINEERING

2011

DISCRETE-EVENT MULTIAGENT COORDINATION: FRAMEWORK AND ALGORITHMS

PHAM MANH TUNG

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2011

Dedication

This thesis is dedicated to my parents.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my thesis advisor, Dr. Seow Kiam Tian, for his guidance and mentoring. Not just on my thesis, he has contributed in my development as an individual.

I would also like to thank my parents for their love and support, and for never losing hope in me during this graduate school adventure.

Finally, I gratefully acknowledge the School of Computer Engineering, Nanyang Technological University for providing me with the financial support and excellent environment to work on this project.

Pham Manh Tung
Singapore, 2011

Summary

This thesis proposes a formal design framework for three inter-agent coordination problems that have not been addressed in the literature: A fundamental coordination problem, a networked coordination problem and an on-line coordination problem. The problems arise when distributed agents need to interact and communicate among themselves in order to guarantee conformance to some inter-agent constraints that restrict their behaviors. These problems are formulated and addressed for a class of agents in a discrete-event behavioral formalism. The basic solution proposed is to equip agents with some distributed, local cooperation mechanism called coordination modules (CM's), by which they can compute their local coordinating actions in response to each other's information update. The framework developed supports the automated synthesis of CM's for discrete-event agents.

The fundamental problem is formulated as synthesizing CM's for multiple agents coordinating to satisfy one pre-specified inter-agent constraint. A fundamental insight unearthed is that discrete-event agent coordination shares the same algorithmic foundation with related problems in the control literature although they are conceptually different domains. This insight leads to the adaptation and extension of related control and observation concepts for the development of a new concept called coordinable language, which is shown to be the necessary and sufficient existence condition of CM's for distributed agents. A new CM synthesis algorithm is developed for the fundamental problem.

The networked coordination problem generalizes the fundamental problem to multiple pre-specified inter-agent constraints distributed among agents. In addressing the problem, the originality of our solution approach emanates from two major research ideas. The first is in proposing a new formalism called the Distributed Constraint Specification Network (DCSN) that can comprehensibly describe the networking constraint relationships among the agents, with every constraint and the group of agents that it is relevant for forming a basic subnet. Each basic subnet of a DCSN therefore specifies a fundamental coordination

problem. The second is in generating and using AND/OR graphs to represent and encompass conflict resolution plans for a DCSN. A conflict resolution plan shows how to detect and resolve any conflict between agents that can arise, due to blocking when coordinating on two or more different constraints that prevents agents from reaching their local design goals. Conflicts are resolved by designing deconflicting CM's for the agents involved. The overall result is a two-step compositional approach to automated design of agent CM's for a DCSN. The first step involves synthesizing CM's for every basic subnet of the DCSN. The second step involves systematically composing these CM's together to form a correct solution for the DCSN, and is supported or facilitated by the following new research developments.

1. A plan generation algorithm is developed to recursively construct an AND/OR graph representation of conflict resolution plans. The novel idea behind the algorithm is in converting a DCSN to an alternative representation called constraint relational network (CRN) that is amenable to efficient decomposition by applying cut-set theory for recursive plan generation. That the algorithm can be easily extended to support problem-dependent planning such as for multiagent nonblocking reconfigurability is also discussed.
2. Efficient algorithms for designing deconflicting CM's are developed based on computing some abstractions of agent models. To effectively manage automated coordination design for a DCSN, these algorithms can in turn be invoked by some criterion-based optimal plan selected from an AND/OR graph representation of conflict resolution plans. The algorithms are invoked through the 'AND' operations defining subnet compositions, whose execution precedence is partially ordered in the optimal plan.
3. To support the selection of an optimal conflict resolution plan, a heuristic search algorithm is developed. The proposed algorithm is guaranteed to return an optimal plan when used with an admissible heuristic designed for an optimization metric formulated for some selection criterion. To this end, a heuristic for selecting a plan that meets the criterion of maximal simultaneity in the execution of subnet compositions is proposed, thereby effectively managing synthesis complexity and speeding up the process of automated coordination design.

Finally, the on-line coordination problem is formulated as developing distributed on-line

strategies for agents to interact and communicate continually between themselves to conform to a given global predicate specifying an inter-agent constraint. A predicate constraint prescribes a subset of the composite state space that the agents should always remain in during interaction. An on-line coordination strategy enables each agent to compute its local coordinating actions in response to situational changes so as to respect the constraint; and is said to support the on-line, partial synthesis of agent CM's. This on-line coordination problem is shown to be solvable in some autonomy permitted setting for coordinable predicates not less restrictive than a given predicate constraint. The solution basis developed is an optimal policy by which agents can interact and communicate to guarantee that transitions of their composite states always remain confined to the largest feasible state subset of the state set defined by a given predicate constraint, and can reach all the states in this state subset. To implement the optimal policy strategically, a key concept called coordination-readiness and its stronger notion called co-stability are developed. Without off-line computed CM's, agents in on-line coordination need to establish or re-establish coordination-readiness upon each situational change before they can correctly determine their next local coordinating actions required by the optimal policy. Inter-agent communication can often be significantly reduced if an agent reasons and communicates based on the conditions for co-stability rather than coordination-readiness. Unlike the latter strategy, the former strategy does not require agents to always inform everyone else whenever they enter a new local state. This reduction in communication is confirmed by an empirical study.

Contents

Acknowledgements	i
Summary	ii
List of Figures	ix
List of Tables	xii
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Objective	3
1.3 Problem Statement and Solution Approach	4
1.4 Thesis Contributions	7
1.5 Related Work	10
1.5.1 Discrete-Event Systems	11
1.5.2 Languages and Automata Theory	12
1.5.3 Supervisory Control of Discrete-Event Systems	13
1.5.4 Distributed Constraint Satisfaction Problem	15
1.5.5 DEC-MDP/DEC-POMDP	16
1.5.6 Commitments and Conventions Hypothesis	17
1.6 Thesis Organization	18
2 BACKGROUND AND NOTATION	20
2.1 Chapter Overview	20
2.2 Languages and Automata	21
2.2.1 Languages	21

2.2.2	Automata	22
2.2.3	Composition of Automata	23
2.3	Supervisory Control of Discrete-Event Systems	25
2.3.1	Discrete-Event Control Problem	25
2.3.2	Desired Behavior - Specification Automata	27
2.3.3	Supervisor Synthesis	28
2.4	Multiagent Coordination as Control Synthesis	31
2.4.1	Multiagent Coordination Planning	32
2.4.2	Coordination Planning Methodology	34
2.4.3	Illustrative Example	35
2.5	Chapter Summary	37
3	FUNDAMENTAL AGENT COORDINATION PROBLEM	38
3.1	Chapter Overview	38
3.2	Discrete-Event Agents and Coordination	40
3.2.1	Discrete-Event Agents and Constraints	40
3.2.2	Uncontrollable and Controllable Events in an Agent Model	41
3.2.3	Coordination Modules and Coordinated Behavior	42
3.3	Problem Statement and Solution Properties	49
3.4	Coordination Module Synthesis	53
3.5	Illustrative Example	55
3.6	Discrete-Event Control versus Coordination	59
3.7	Chapter Summary	63
4	NETWORKED AGENT COORDINATION PROBLEM - PART I: CONFLICT RESOLUTION	64
4.1	Chapter Overview	64
4.2	Preliminaries	67
4.2.1	Natural Projection	67
4.2.2	Natural Observer	69
4.2.3	Output Control Consistency	70
4.3	Distributed Constraint Specification Network (DCSN)	71
4.4	A Manufacturing Transfer Line Example	73

4.5	Networked Coordination Problem Statement	74
4.6	Basic Subnet Synthesis	75
4.7	Composing Subnets with Conflict Resolution	78
4.7.1	Composing Basic Subnets	78
4.7.2	Composing Arbitrary Subnets	86
4.8	Chapter Summary	91
5	NETWORKED AGENT COORDINATION PROBLEM - PART II: REPRESENTATION AND GENERATION OF CONFLICT RESOLUTION PLANS	93
5.1	Chapter Overview	93
5.2	Representations of Conflict Resolution Plans	96
5.2.1	Directed Graph Representation of Conflict Resolution Plans	97
5.2.2	AND/OR Graph Representation of Conflict Resolution Plans	101
5.3	Generation of AND/OR Graph Representation of Conflict Resolution Plans	107
5.4	Selection of Optimal Conflict Resolution Plan	112
5.4.1	General Heuristic Search for Optimal Conflict Resolution Plan	113
5.4.2	Reducing Execution Time Through Parallel Compositions of Subnets	115
5.5	Chapter Summary	120
6	ON-LINE AGENT COORDINATION PROBLEM	122
6.1	Chapter Overview	122
6.2	Problem Formulation	124
6.3	Coordination Synthesis	127
6.4	On-line Coordination Strategies	131
6.4.1	With Full Communication	131
6.4.2	With Reduced Communication	132
6.5	Illustrative Example	137
6.6	Experimental Evaluation	138
6.7	Multiagent On-line Coordination	141
6.8	Chapter Summary	142
7	CONCLUSION AND FUTURE WORK	144
7.1	Contributions of The Thesis	144
7.2	Future Work	147

CONTENTS	viii
List of Publications	150
Appendix: Modular Control of Discrete-Event Systems	152
References	156

List of Figures

1.1	A general coordination design framework: Given a set of n interacting agents (arbitrarily depicted for $n = 3$) and inter-agent constraints, synthesize CM's for every agent, specifying how the agents should interact and communicate with one another to achieve conformance to each constraint between them. The notation $\parallel$ denotes an interaction operator which is formulated differently in different agent coordination frameworks. A formal definition of the operator as used in our framework will be given later in this thesis.	2
1.2	A sample DCSN of 7 agents and 3 distributed constraints: Constraint $C_{\{1,2,3,4\}}^1$ restricting the behaviors of agents A_1, A_2, A_3 and A_4 ; constraint $C_{\{2,5,7\}}^2$ restricting the behaviors of agents A_2, A_5 , and A_7 ; and constraint $C_{\{5,6,7\}}^3$ restricting the behaviors of agents A_5, A_6 , and A_7	6
1.3	Illustrating the basic difference: The notion of event-feedback fundamental to implementing supervisory control is absent in multiagent coordination. For the supervisory control problem, the supervisor S is the required solution but for the multiagent coordination problem, the coordination modules S'_1 and S'_2 are the required solution.	14
2.1	The control synthesis approach: Given a DES A and a specification automaton C , the objective is to synthesize a supervisor to control DES A such that the constraint specified by C is never violated.	27
2.2	Multiagent coordination planning for a distributed train system.	35
2.3	A simplified supervisor S	37
2.4	Coordination modules for the distributed train agents.	37

3.1	Two agents A_1 and A_2 coordinating through their respective CM's S_1 and S_2 : Following the execution of a string $s \in L(A_1 \parallel A_2)$, agent A_i updates the state of its CM S_i to $x_i = \delta^{S_i}(P_{\Sigma A, \Sigma S_i}(s), x_0^{S_i})$. Only every event $\sigma_i \in \Sigma^{A_i}$ that is defined at x_i (i.e., $\delta^{S_i}(\sigma_i, x_i)!$) is enabled and so can be executed next by A_i . The result is that the system behavior is restricted to a sublanguage of $L(A_1 \parallel A_2)$	42
3.2	Coordination planning for a manufacturing system.	57
3.3	Automaton S with $L_m(S)$ - the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$	58
3.4	Coordination modules for the manufacturing agents before applying the state-size reduction procedure $CMreduce$	58
3.5	Coordination modules for the manufacturing agents.	59
3.6	Non-optimal coordination modules for the manufacturing agents.	60
4.1	A manufacturing transfer line system.	73
4.2	Constraints for a manufacturing transfer line system.	74
4.3	A DCSN for a manufacturing transfer line system.	74
4.4	Procedure $CMBasicSubnet$ for synthesizing a CM S_i^k for every agent A_i in $\mathcal{N}_1^k = (J_k, C_{J_k}^k)$ such that $(A_1 \parallel S_1^k) \parallel \dots \parallel (A_n \parallel S_n^k) \equiv SUP^k$	76
4.5	CM's of A_1 and A_2 for $E_{\{1,2\}}^1$	77
4.6	CM's of A_1 and A_2 for $E_{\{1,2\}}^2$	77
4.7	Procedure $DeconflictBasicSubnet$ for resolving conflict between two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$	84
4.8	A DCSN of 4 agents and 3 constraints. It is possible that in ensuring overall network nonblockingness, agents A_2 and A_4 might need to communicate with each other even though they are coordinating on different constraints with different agents.	89
4.9	Complete CM solution for a manufacturing transfer line system.	91
5.1	A specific DCSN of 10 agents and 4 constraints, and its subnets.	98
5.2	The directed graph of conflict resolution plans for the DCSN in Figure 5.1(a).	100
5.3	Four complete planning sequences for the DCSN in Figure 5.1(a).	100
5.4	The AND/OR graph of conflict resolution plans for the DCSN of Figure 5.1(a).	104

5.5	Three AND/OR conflict resolution plans for the DCSN in Figure 5.1(a). . .	105
5.6	The CRN of the DCSN in Figure 5.1(a).	108
5.7	Procedure <i>GenerateANDORGraph</i> for generating an AND/OR graph of conflict resolution plans for a DCSN.	110
5.8	A subgraph of the AND/OR graph in Figure 5.4. This subgraph is generated by the first call to <i>GenerateANDORGraph</i> to decompose the CRN in Figure 5.6.	112
5.9	Procedure <i>HeuristicPlanSelection</i> for selecting an optimal conflict reso- lution plan for a DCSN $\mathcal{N}$	114
5.10	The DCSN and CRN for the manufacturing transfer line system presented in Chapter 4.	118
5.11	The AND/OR graph of conflict resolution plans for the manufacturing trans- fer line system.	119
5.12	A conflict resolution plan for the manufacturing transfer line system. . . .	120
6.1	OnlineCoAgent-ComFull(A_i)($i \in \{1, 2\}$)- On-line coordination strategy with full communication for agent A_i . For definiteness of description, the strategy instance for A_1 is shown; that for A_2 is the same except that its reciprocal agent is A_1	132
6.2	OnlineCoAgent-ComReduce(A_i) ($i \in \{1, 2\}$) - On-line coordination strat- egy with reduced communication for agent A_i . For definiteness of descrip- tion, the strategy instance for A_1 is shown; that for A_2 is the same except that its reciprocal agent is A_1	135
6.3	A multiagent coordination example	138

List of Tables

5.1	Number of nodes in directed graph and AND/OR graph representations of conflict resolution plans for strongly connected DCSN's	105
5.2	Number of nodes in the directed graph and AND/OR graph representations of conflict resolution plans for weakly connected DCSN's	106
6.1	Bandwidth reduction (in %) of OnlineCoAgent-ComReduce over OnlineCoAgent-ComFull for varying degrees of CS permissiveness α	139

Chapter 1

INTRODUCTION

1.1 Motivation

In modern electronic environments, it is becoming increasingly important to deploy multiple agents with the autonomy capability of coordinating among themselves in conformance to distributed inter-agent constraints [49, 16, 97, 84, 105, 106]. In general, these are social constraints that specify mandatory policies for regulating the interaction behaviors of agents. Typically, social constraints include requirements of deadlock avoidance, mutual exclusion and others associated with ensuring fair competition for resources in a cooperative fashion, without blocking any agent from achieving their local goals. Intelligent schedulers [62, 90], distributed sensor networks [109, 108, 83], unmanned vehicles [7, 52, 18, 93] and distributed networked resource allocation [1, 41] are some examples of multiagent applications where the aforementioned agent regulation capability is a mandatory requirement.

In the abovementioned application domains, agents must follow certain built-in strategies to coordinate their activity in order to ensure the success of their design goals. For example, providers and consumers of a networked resource must operate cooperatively by following a negotiation protocol that regulates their interaction process [1]; distributed sensors must coordinate the allocation of their sensing resources in order to accurately track multiple targets [109, 51]; and unmanned train agents must coordinate their entry to and exit from a shared tunnel on a, say, first arrival first access basis [93, 80]. A common characteristic of these examples is that the local actions taken by agents and the interactions between them can have significant implications on the performance and integrity of the multiagent

Figure 1.1: A general coordination design framework: Given a set of n interacting agents (arbitrarily depicted for $n = 3$) and inter-agent constraints, synthesize CM's for every agent, specifying how the agents should interact and communicate with one another to achieve conformance to each constraint between them. The notation $\parallel$ denotes an interaction operator which is formulated differently in different agent coordination frameworks. A formal definition of the operator as used in our framework will be given later in this thesis.

system as a whole.

This research is concerned with the multiagent coordination problem which can be described in a design framework as follows. The starting point is that we are given a set of interacting agents that have been designed to have different local plans in pursuit of different local goals, and a set of inter-agent constraints that govern the high-level behavior of these interacting agents. In this context, constraints can be seen as a means to prescribe inter-agent negotiation protocols [1, 41], to model an inter-agent agreement [71] that promotes fair use of a shared resource, or to specify some social laws [85, 86] that the agents must obey to ensure the integrity of the multiagent system. The problem of interest is then on how to synthesize built-in strategies realized as local coordination modules (CM's) for every agent so as to guarantee that their coordinated behavior will conform to the given inter-agent constraints. These synthesized CM's are "pluggable" onto individual agents' local plans via some interaction operator, and used as interfaces through which the agents interact and communicate among themselves, as depicted in Figure 1.1.

Cast in the general design framework shown in Figure 1.1, there have been extensive research efforts concentrating on different aspects of coordination. For example, Rosenschein and Zlotkin [71] investigate on how to design inter-agent constraints, referred to as “rules of encounters”, to restrict individual self-interested agents to act in a certain way that benefits the whole system. Shoham and Tennenholtz [85, 86] study the problem of designing social constraints to ensure that interacting agents will not arrive at situations that can lead to conflict. There have also been numerous attempts to take an economic approach and consider market mechanisms as design tools for efficiently allocating resources in multiagent systems [97, 15]. One of the features that differentiates our work from these research efforts is that we assume inter-agent constraints as already designed and focus on synthesizing CM’s for interacting agents to respect the constraints. In a related work, Yokoo et al. [109, 108, 107, 110, 51] study a distributed constraint satisfaction / optimization problem (DCSP/DCOP) and also assume that the inter-agent constraints are already given. However, as will be discussed later, while the well-known DCSP/DCOP framework concentrates on a class of agents modeled as variables, we concentrate on a different class of agents. Finally, another major research stream in multiagent coordination uses DEC-MDP/DEC-POMDP¹ as multiagent system models [6, 50, 3, 31], and often assumes no inherent inter-agent constraints. The coordination problem in their context is to design a joint plan for the agents to optimize some quantities of interest.

1.2 Objective

In this thesis, we investigate the multiagent coordination design problem in a discrete-event setting. We propose a formal framework to analyze the interactions between local agent actions and support the design of CM’s for a class of agents modeled as distributed discrete-event processes (DEP’s). Previous work has shown that Discrete-Event Systems (DES’s) [12, 42, 66, 65, 101] is an important domain independent framework for the modeling and control of complex man-made systems such as communication systems [75], manufacturing systems [13, 43, 68, 14], autonomous mobile robots [40] and transportation systems [81, 4, 29]. By attempting to develop a generic coordination theory for distributed discrete-event agents, we hope to provide a theoretical basis for a wide range of applications.

¹DEC-MDP/DEC-POMDP is a standard abbreviation for Decentralized-Markov Decision Process / Decentralized-Partial Observation Markov Decision Process.

Our development of the coordination framework is founded on three important questions that are not investigated in the literature. Firstly, how to describe the inter-agent constraint relationships among discrete-event agents in a formal way such that the overall problem specification is comprehensible? As a matter of fact, the coordination among a large number of agents is organized in some distributed network. Therefore, some appropriate networking showing whom each agent will need to coordinate with, and what their relevant inter-agent constraints are, should provide a clear basis for understanding the problem. A constraint is said to be relevant for an agent provided it restricts the agent's behavior. While some constraint network bases for multiagent coordination do currently exist, they are mostly developed for agents modeled as variables with finite domains [109, 51], or as probabilistic decision makers [50, 89], and not for discrete-event agents.

Secondly, how to automate the coordination design process? Previous control work on DES's often (implicitly) assumes some degree of human ingenuity in planning design decisions towards obtaining the final solution, which may be in the form of modular controllers controlling a complex DES [28] or local supervisors supervising distributed DEP's [8, 9]. Relying on human ingenuity would be scientifically inadequate for coordination design because, for a complex system involving a large number of agents and constraints, it is often hard or even impossible for a human expert to identify the best solution-process plan among a possibly huge number of candidate plans.

Finally, what tools and techniques aid in the efficient design of discrete-event multiagent coordination? To the best of our knowledge, this thesis is the first attempt to formulate and address the problem of coordination among discrete-event agents. Thus it is necessary to develop suitable tools and techniques for the efficient design of CM's for discrete-event agents, and lay a foundation for future research.

1.3 Problem Statement and Solution Approach

The principal problem addressed in this thesis is as follows.

How to automatically synthesize CM's for a network of agents so that using these synthesized CM's, the agents can interact and communicate among themselves to achieve conformance to a set of pre-specified inter-agent constraints distributed among them?

We make the following assumptions about agents and distributed constraints.

- A coordinating agent is modeled as a DEP and represented by a (deterministic) automaton [36]. Besides providing a simple yet powerful modeling formalism, automata are also amenable to composite operations and mathematical analysis [12]. From an agent planning viewpoint, an agent automaton is a transition structure that represents all possible ways to achieve the agent's local goal, and specifies all the event sequences that can be generated during agent execution. An automaton has an initial state and a set of marker states, and to accomplish its local goal, an agent may execute one of possibly many event sequences that bring its automaton from the initial state to a marker state. The sequences that the agent actually executes would depend on contingencies arising in the course of its operation.
- A system of multiple discrete-event agents freely interacting can then be represented by a model of interleaving events formalized by the synchronous product [12] of all individual agent automata. The agents would need to coordinate among themselves if, due to system needs or limitations, some of the event sequences generated by the product automaton are undesirable and must be prevented from happening. The coordination goal is then specified as a set of inter-agent constraint automata stipulating the set of desirable event sequences that is to be conformed to when agents interact to accomplish their local goals.

Agents in our framework coordinate by following built-in coordination strategies realized as CM's to appropriately enable and disable their events at each event execution step, so that none of the undesirable event sequences implied by a given set of inter-agent constraints are ever executed. Upon an agent executing one of its enabled events, the agent would need to inform all other agents whose CM's contain the executed event, so that these informed agents can determine their next set of events to enable.

In addressing the coordination design problem, we develop a formal framework which supports an organizational formalism that is naturally suited for modeling the inter-agent constraint relationships among agents, and fully automates the process of designing CM's.

The organizational formalism proposed is called the Distributed Constraint Specification Network (DCSN). A DCSN describes the constraint relationships among agents by associating every agent with its relevant constraints, thereby specifying "who needs to coordinate with whom over what constraints". Graphically depicted, a DCSN is a specification

tool which is designer comprehensible in that it clearly shows the networking of constraints among agents. Importantly, a DCSN can be analyzed by systematically decomposing it into basic constraint subnets, with each consisting of a constraint and the group of agents that the constraint is relevant for. In general, a constraint subnet of a DCSN consists of one or more such basic subnets. Figure 1.2 shows an example of a DCSN of 7 agents and 3 constraints. This DCSN has three basic subnets: The first includes constraint $C^1_{\{1,2,3,4\}}$ and agents A_1 , A_2 , A_3 and A_4 , the second includes constraint $C^2_{\{2,5,7\}}$ and agents A_2 , A_5 , and A_7 , and the third includes constraint $C^3_{\{5,6,7\}}$ and agents A_5 , A_6 , and A_7 .

Figure 1.2: A sample DCSN of 7 agents and 3 distributed constraints: Constraint $C^1_{\{1,2,3,4\}}$ restricting the behaviors of agents A_1 , A_2 , A_3 and A_4 ; constraint $C^2_{\{2,5,7\}}$ restricting the behaviors of agents A_2 , A_5 , and A_7 ; and constraint $C^3_{\{5,6,7\}}$ restricting the behaviors of agents A_5 , A_6 , and A_7 .

In synthesizing CM's for a given DCSN, our approach stems from the basic premise that it is often not computationally feasible to examine every composite state of a multiagent system and determine the set of enabled events for each agent at each composite state, owing to state space explosion. Thus the approach taken is to synthesize CM's for every basic subnet of the DCSN, and then systematically compose them together to form a correct solution for the complete DCSN.

We show that this compositional synthesis approach has a number of benefits. Firstly, it enables a systematic analysis of complex distributed multiagent systems. Secondly, it often enables a computationally efficient construction of CM's for a large network of interacting agents and constraints. Finally, it lays a foundation for automating the design process of distributed coordinating agents.

1.4 Thesis Contributions

This thesis makes several contributions to multiagent coordination research. These contributions are discussed in the following.

Fundamental Coordination Synthesis (Chapter 3)

In Chapter 3, we formulate and address the most fundamental discrete-event coordination problem as synthesizing CM's for multiple agents coordinating to satisfy one pre-specified inter-agent constraint.

A fundamental insight unearthed is that discrete-event agent coordination shares the same algorithmic foundation with related problems in the control literature although they are conceptually different domains. This insight leads us to adapt and extend related control [66] and observation [45] concepts for the development of a new concept called coordinable language, which is shown to be the necessary and sufficient existence condition of CM's for distributed agents. A new CM synthesis algorithm is then developed for the fundamental problem. Importantly, we show that the synthesized CM's are minimally interventive with the agents' local plans and ensure that communication between the agents is made only when necessary. Each of the CM's can also be efficiently implemented in terms of memory requirements as their state size may be greatly reduced, though not necessarily minimized, for each agent model. In comparison with an existing synthesis algorithm [80, 82], we explain and demonstrate the design improvement in terms of reduction in inter-agent communication and CM state sizes that the new synthesis algorithm offers.

Networked Coordination Synthesis (Chapters 4 and 5)

In Chapters 4 and 5, we generalize the fundamental problem to a networked coordination problem. In contrast to the fundamental problem where there is only one global inter-agent constraint restricting every agent, in the networked coordination problem, there are multiple inter-agent constraints, each of which restricts a subgroup of agents. This situation is pervasive in large-scale distributed systems, where every agent has to coordinate with some but not necessarily all the other agents in a system. The agent subgroups can be overlapping, meaning that there can exist an agent that is involved in different subnets, and so needs to coordinate on two or more different inter-agent constraints. This presents a

challenging design problem since conflicts can arise due to agents in overlapping groups, and if these agents cannot properly resolve the conflicts, they can get into a situation of blocking where some agents fail to reach their design goals.

As already discussed, agents and constraints of a networked coordination problem can be organized into a DCSN. Our compositional synthesis approach for a given DCSN consists of two main steps as follows. In the first step, we apply the synthesis algorithm developed for the fundamental problem to every basic constraint subnet of the DCSN. This is to construct, for each agent, one local CM for each of the agent's relevant constraints. These local CM's can be used by the agents to satisfy individual constraints separately, but not jointly in general due to possible conflicts as pointed out earlier. So in the second step, we generate and execute a conflict resolution plan to systematically compose (the CM solutions of) subnets together to form (a CM solution of) the complete DCSN. Each step of a conflict resolution plan entails composing two subnets together by designing deconflicting CM's for agents in the subnets concerned and interposing them between the agents' local CM's. As will be detailed in Chapter 4, these deconflicting CM's are used to resolve the conflicts between agents due to their using local CM's synthesized for different subnets. In the chapter, algorithms for designing deconflicting CM's are presented. The algorithms are developed based on computing some abstractions of agent models, and often enable constructing deconflicting CM's in an efficient manner. Local and deconflicting CM's are then shown to constitute a correct solution to the networked coordination problem.

The theoretical results and algorithms for generating a conflict resolution plan are presented in Chapter 5. Therein, we propose to represent and encompass conflict resolution plans for a given DCSN using AND/OR graphs [53]. We then develop an algorithm for generating an AND/OR graph representation of conflict resolution plans for a given DCSN. Using this representation, each plan partially orders the execution precedence of subnet compositions defined by 'AND' operations, through which the algorithms for designing deconflicting CM's are invoked. The novel idea behind the algorithm is in converting a DCSN to an alternative representation called constraint relational network (CRN), which is amenable to efficient decomposition by applying cut-set theory [20] for recursive plan generation. As will be discussed, the algorithm can be easily extended to generate only conflict resolution plans which satisfy some problem-dependent requirements. In this direction, we explain how the algorithm can be extended to support planning for multiagent nonblocking reconfigurability. Following, we develop a heuristic search algorithm to select

an optimal plan from an AND/OR graph representation of conflict resolution plans. The proposed algorithm is guaranteed to return an optimal plan when used with an admissible heuristic designed for an optimization metric formulated for some selection criterion. To this end, we design an important heuristic for selecting a plan that meets the criterion of maximal simultaneity in the execution of subnet compositions.

In addressing the networked coordination problem, the originality of our approach lies in proposing DCSN's to specify the networking constraint relationships among agents, and in generating and using AND/OR graphs to represent and encompass conflict resolution plans for a DCSN. Together with the algorithm for basic subnet synthesis presented in Chapter 3, Chapters 4 and 5 provide a foundation for automating the design of a network of coordinating discrete-event agents.

On-line Coordination Synthesis (Chapter 6)

In Chapters 3, 4 and 5, the synthesis approach developed entails computing off-line the complete coordinating actions for all anticipated interacting situations of each agent and storing them as CM's. At run-time, the correct coordinating actions are then simply retrieved from the CM's and enforced accordingly by the agent in interaction with other agents in the system. Thus the approach taken can be referred to as an off-line synthesis approach. This off-line synthesis approach has both merits and demerits. On the one hand, the approach is suitable for applications where coordinating decisions or actions have to be made as fast as possible during the agents' run-time interaction. The coordination actions of each agent can be retrieved from its coordination modules (CM's) or plans, synthesized off-line. On the other hand, the off-line planning associated with the approach may be computationally too expensive for applications where the number of anticipated interacting situations to consider is too large (i.e., the CM's to be constructed are too large), or when off-line planning time is limited.

In Chapter 6, we study an on-line coordination problem as developing distributed on-line strategies for agents to interact and communicate continually between themselves to conform to a given global predicate specifying an inter-agent constraint. A predicate constraint prescribes a subset of the composite state space that the agents should always remain in during interaction. An on-line coordination strategy enables each agent to compute its local coordinating actions in response to situational changes so as to respect the constraint;

and is said to support the on-line, partial synthesis of agent CM's. A significant merit of this on-line approach is that it avoids altogether the off-line construction of CM's as with the off-line approach. Thus the on-line synthesis approach complements the off-line approach proposed for the fundamental problem, and is more suitable for applications in which off-line planning time is limited or agents have limited memory to store large CM's.

The on-line coordination problem is shown to be solvable in some autonomy permitted setting for coordinable predicates not less restrictive than a given predicate constraint. In essence, a coordinable predicate defines a feasible composite state subset that the agents can coordinate to remain in during their interaction. The solution basis developed is an optimal policy by which agents can interact and communicate to guarantee that transitions of their composite states always remain confined to the largest feasible state subset of the state set defined by a given predicate constraint, and can reach all the states in this state subset. By implementing the optimal policy, the agents, while respecting the constraint, can visit as many states as possible during their interaction, and therefore, have maximal autonomy over their own actions.

To enable the agents to implement the optimal policy in an on-line fashion, a novel concept called coordination-readiness is developed. Without off-line computed CM's, agents in on-line coordination need to establish or re-establish coordination-readiness upon each situational change before they can correctly determine their next local coordinating actions required by the optimal policy. The most conservative strategy to do so is to allow every agent to inform everyone else of its local state change whenever it enters a new local state. While this strategy is easy to implement, it requires excessive communication which might not always be necessary for the agents to attain coordination-readiness. To reduce inter-agent communication, a stronger notion of coordination-readiness called co-stability is developed. Based on the conditions of co-stability, another strategy is developed such that, upon entering a new local state, an agent reasons and only updates other agents of its local state change if it locally detects that the agents are no longer co-stable. Our empirical study shows that when compared with the first strategy, the second strategy can often significantly reduce inter-agent communication.

1.5 Related Work

We now discuss some of the most important related work from the literature.

1.5.1 Discrete-Event Systems

The field of DES's [12] provides a domain independent framework for modeling and control of complex man-made systems. DES's are often contrasted with continuous-state, time-driven systems [23] which are of interest to the classical control community. Historically, continuous time-driven systems are of significant importance for studying natural phenomena in which the quantities of interest, such as the velocity and acceleration of rigid particles or the pressure and temperature of fluids, are modeled as continuous variables which can take any real value as time evolves [12]. However, in today's increasingly computer dependent world, DES's have become more and more important as an "integral part of our life" [12]. Most of the systems we are concerned with today are, at some level of abstraction, DES's in nature. They include, but are not limited to, communication [75], manufacturing [43, 68, 13, 14], autonomous mobile robot [40] and transportation systems [81, 4, 29]. In contrast to continuous time-driven systems [23], the quantities of interest in DES's, such as the number of messages in a router buffer and the number of parts in an inventory, are discrete and event-driven, meaning that they can only take a certain number of values upon the occurrence of some instantaneous event, such as the arrival of a new message and the removal of an inventory part. The common characteristics of DES's are that they are dynamical, mostly technological and often highly complex [101].

Apparently, DES's form the underlying basis for many man-made systems and studying about them could unearth general principles that are common to a wide range of application domains [101]. Like other types of system problems, the problem of coordination among distributed agents modeled as DEP's is of significant importance. However, to the best of our knowledge, none of the currently available coordination frameworks could be easily applied to formulate and address coordination problems for distributed discrete-event agents. The reason is that most of these frameworks have their own merits suited to certain application domains that exploit or justify their inherent modeling assumptions regarding the coordination interfaces and inter-dependencies of interacting agents. For example, the DCSP/DCOP framework [109, 51] assume that agents are modeled as variables with finite domains, the DEC-MDP/DEC-POMDP frameworks [50, 3, 31] assume that agents are probabilistic planners, and most game-theoretic frameworks [71, 84] assume that agents are self-interested and profit-maximizing.

1.5.2 Languages and Automata Theory

In the classical theory originating from computer science, a finite-state automaton [36] is a mathematical model of a self-operating machine and consists of a finite number of states and transitions between those states, each of which is associated with a symbol representing an input that drives the state transition. An automaton can be represented graphically as an edge-labeled directed graph, with states represented by nodes and symbolic transitions represented by labeled edges. In its operation, an automaton starts in one of the states called the initial state and takes as input a finite sequence of symbols called a word. It then reads one symbol after another and transits accordingly from state to state, until the word is completely read. There is a subset of the states of the automaton defined as accepting states. If, after reading the word and making the respective state transitions, the automaton stops at one of the accepting states then the word is said to be accepted by the automaton. Otherwise, the word is said to be rejected. The set of all the words accepted by an automaton is called the language recognized by the automaton.

In computer science [36, 35, 39, 11] and programming languages [47, 34], automata as language “recognizers” have been widely used as the primary tool for the analysis of finite-state systems. In multiagent systems research, automata have also been used to compactly represent coordination policies for probabilistic planning agents [50, 5, 63].

In a fundamental development originating in control science [66, 65, 12, 42, 101], languages and automata theory is applied to model DES’s or DEP’s. The symbolic concepts of a DES include states and events. Here, a state is a discrete instance abstracting the data values and activities that the DES is engaged in at a particular moment in time. An event represents some qualitative change and its occurrence brings the DES from one state to another. For example, a machine could be modeled as a DES consisting of two states “ON” and “OFF”. When at the state “ON”, the machine will immediately move to the state “OFF” when a “machine turned off” event occurs, and when it is at the state “OFF”, it will move to the state “ON” immediately following the occurrence of a “machine turned on” event. Put together, a DES defines a system that evolves from state to state in accordance with the abrupt occurrence of events. A DES can thus be readily modeled by an automaton, with automaton states modeling DES states and automaton state transition symbols modeling DES events. Instead of a language “recognizer”, the automaton is viewed in this context as a “generator” of (regular) languages [36] prescribing the behavior of the DES.

A feature that differentiates the conceptual interpretation of automata in control science [66, 65, 12, 42, 101] with that in classical applications of automata in computer science [36, 35, 39, 11] is that, in the former, events are treated as internally generated by a DES whereas, in the latter, they are treated as external inputs.

In this thesis, we consider a class of agents which are DEP's modeled by automata. Such agents can proactively execute their own events and are called discrete-event agents. In coordinating, these agents would need to interact and communicate their own event information between themselves so that all social constraints between them are respected. This multiagent coordination problem is different from the fundamental problem studied in control science [66, 65, 12, 42, 101]. It arises naturally in application domains such as robotic agents where distributed agent autonomy is a key consideration. The control problem arises in domains where the DEP's are not active agents but passive system components to be controlled. In contrast to a passive system component, a discrete-event agent can autonomously execute a local plan captured by its own DEP model, in tandem with its "built-in" coordination strategy.

1.5.3 Supervisory Control of Discrete-Event Systems

In recent work, Seow et al. [80, 82] establish an explicit connection between discrete-event control and multiagent coordination. In essence, the authors differentiate two conceptually different views - the control and the coordination viewpoints - to the same set of interacting DEP's. Given a constraint to be satisfied, from the control viewpoint, the problem of interest is to synthesize a supervisor that controls the set of interacting DES's through event enablement and disablement to achieve conformance to the constraint. From the coordination viewpoint, however, the problem of interest is to synthesize local CM's for every DEP, interpreted as a coordinating agent, through which to specify how the agents should interact and communicate to achieve conformance to the constraint. Interestingly, the authors show that the two conceptually different problems are mathematically equivalent, in that a set of CM's could be computed from a single supervisor and vice versa [82, Theorem 1]. Importantly, this finding provides the research impetus towards developing a coordination theory for a class of agents founded on a discrete-event framework.

Motivated by the mathematical connection between discrete-event coordination and control [80, 60, 61, 82], one of the contributions of this thesis is the adaptation and applica-

Figure 1.3: Illustrating the basic difference: The notion of event-feedback fundamental to implementing supervisory control is absent in multiagent coordination. For the supervisory control problem, the supervisor S is the required solution but for the multiagent coordination problem, the coordination modules S'_1 and S'_2 are the required solution.

tion of advanced techniques and concepts in supervisory control theory [66, 65, 12, 42, 101] to the development of synthesis methods for multiagent coordination, without “reinventing the wheel”. However, we must emphasize that while mathematically related, the coordination and supervisory control problems are conceptually different. The latter achieves conformance to constraints through an external supervisor interacting with the system of DEP’s by event enablement or disablement based on information feedback on the occurrence of events [see Figure 1.3(a)]. But the former does so by the agents interacting among themselves through their CM’s [see Figure 1.3(b)]. Unlike event-feedback control, coordinating interaction also involves synchronous event message passing, where each agent sends messages to, and receives relevant event messages from the other agents. This form of communication is necessary to maintain conformance to the specified inter-agent con-

straint.

In summary, we do not claim to further research in supervisory control of DES's. The originality of our work herein lies in formulating and addressing novel coordination problems that arise when discrete-event agents need to interact as depicted in Figure 1.3(b). Where appropriate, existing techniques and concepts in supervisory control theory are adapted or extended to address the new problems. This is expounded at greater length in Chapters 3 and 4.

1.5.4 Distributed Constraint Satisfaction Problem

In its classical form, a DCSP consists of a finite set of agents, each represented as a variable with an associated finite domain of values; and a set of inter-agent constraints, each of which is a relation on the values of some variables. A constraint can be represented by a set of constraint-satisfying (CS) tuples, i.e., those combinations of values that satisfy the constraint. The problem of interest is for the agents to cooperatively assign values to their variables so as to satisfy all the constraints. It should be noted that in this classical form, the DCSP framework cannot be readily exploited to formulate sequential (ordering) constraints for coordinating agents.

Among the first to study the relationship between DCSP and deterministic automata, Vempaty [96] shows that, in principle, any (non-sequential or static) variable constraint can be compactly represented by a deterministic automaton. This is done by defining the same precedence order of variables in every CS-tuple of every constraint, and using an algorithm to construct a constraint automaton from the set of CS-tuples for each constraint. Expressed in the terminology of DES's [12], the algorithm essentially defines an event for every variable assignment, and converts every CS-tuple to an event string marked by the constraint automaton. Representing variable constraints by such deterministic automata enables established results on regular languages [36] to be used to solve the CSP effectively [96]. Importantly, once the constraint automata are constructed [96], this automata-theoretic CSP can also be solved as a specific instance of our problem, in which every variable agent is represented by a one-state automaton with all of its events self-looped at that state, and every constraint automaton is an inter-agent constraint. An interesting feature is that our framework can provide a solution basis for dynamically changing variable assignments without violating constraints.

Pesant [58] formulates a regular language membership constraint problem that generalizes the automata-theoretic CSP formulation [96] from global non-sequential to sequential constraints. He shows that global sequential constraints can also be represented by deterministic automata. A solution to the problem [58] is any sequence of events - each of which represents the assignment of a value to an agent variable - that is marked by all automata modeling the global sequential constraints for the agents. In turn, this regular language membership constraint problem [58] can similarly be solved as a specific instance of our problem.

In our opinion, showing that DCSP can be solved as a discrete-event coordination problem is of theoretical and intellectual interest. We should, however, emphasize that solving DCSP as such is not necessarily the approach we recommend. On the contrary, we recommend using efficient state-of-the-art DCSP agent mechanisms (e.g., [109, 107, 110]) as appropriate to an application domain.

1.5.5 DEC-MDP/DEC-POMDP

Another major research direction is the development of Decentralized MDP/POMDP frameworks for multiagent coordination. In this research direction, researchers on formal models of multiagent systems focus on extensions of a formal agent model called Markov Decision Process (MDP) [64] to multiagent settings [6, 50, 3, 31]. A fundamental feature that differentiates our work from those research efforts is that we treat events rather than states as the fundamental concept, and model them as explicit transitions in an agent structure. This enables interesting characteristics of agents to be modeled using the properties of events. For instance, the autonomy of coordinating agents can be modeled using controllable and uncontrollable events, as will be explained in Chapter 3. Moreover, it is argued that in some technical situations, when compared to the state-based modeling approach, the event-based modeling approach is more flexible and may be computationally more advantageous [10]. This is because the structural information of a system, which is not reflected in state-based models, is readily captured by the concept of events in event-based models.

1.5.6 Commitments and Conventions Hypothesis

Jennings [38, 37] has hypothesized that commitments and conventions constitute coordination in multiagent systems. The former are pledges to undertake specific actions; the latter are means of monitoring commitments as the system actually evolves. Accordingly, our framework, being very general, only supports the design of agent conventions but lends credentials to the Centrality of Commitments and Conventions Hypothesis [38] that coordination mechanisms can be reduced to commitments and their associated conventions, as explained below:

Let automaton A be (the free behavior of) a distributed agent equipped with a CM S . When in a local state that has more than one event being enabled under S , the agent is deemed to be capable of selecting an event to execute via an underlying mechanism. This mechanism, however, is assumed not modeled in our framework. Abstracting away this application dependent mechanism allows us to develop a generic coordination theory. Therefore, within our framework, no postulation is made on how the mechanism selects an event for execution. The actual selection might be decided off-line or on-line among the agents through their mechanism. This might be separately designed for a peculiar application domain by a system designer. In connecting to the two concepts that co-found the idea of coordination, an event σ local to agent A that is enabled and selected for execution is deemed as a commitment by the agent, since the selected event would occur unless preempted by another whose occurrence is uncontrollable. The agent's (local) convention is the language space defined by both automata A and S , since it details all the event sequences in which the agent's commitments can traverse feasibly, i.e., without ever exiting the bounds of the inter-agent constraint on which S is synthesized.

Finally, viewed from a taxonomy of multiagent interactions proposed by Parunak et al [54], our framework can be regarded as collaboration defined as 'coordination based on direct communication plus joint intentions'. In our coordination paradigm, the agents are designed to synchronize or directly communicate with each other on a subset of their events to carry out their joint intentions. The inter-agent constraint can be said to specify their joint intentions.

1.6 Thesis Organization

Besides this introduction, this thesis contains six chapters.

Chapter 2 reviews important concepts and results in the languages and automata theory and the supervisory control theory of discrete-event systems that are most relevant to our work. It also elaborates on the mathematical connection between discrete-event control and agent coordination, and presents how a multiagent coordination problem can be addressed using control techniques.

Chapter 3 studies the fundamental agent coordination problem. The concept of a coordinable language is introduced and a theorem shows that it is the necessary and sufficient condition for the existence of CM's for distributed agents. The development of the synthesis algorithm follows. The chapter ends with a discussion on related work, distinguishing and highlighting the importance of the coordination problem from related problems in the literature.

Chapters 4 and 5 formulate the networked coordination problem and present the compositional synthesis approach. Chapter 4 proposes the DCSN as a specification basis for the networked coordination problem. It explains and illustrates the problem of conflicts between agents due to their using local CM's synthesized for different subnets of a DCSN. An efficient approach to resolve the conflicts is then proposed. Chapter 5 presents the theory and algorithms for generating conflict resolution plans for a given DCSN. The chapter shows how AND/OR graphs can be used to represent and encompass conflict resolution plans, and presents the algorithm for generating an AND/OR graph representation of conflict resolution plans for a given DCSN. That the algorithm can be easily extended to support problem-dependent planning such as for multiagent nonblocking reconfigurability is also discussed. The development of a heuristic search algorithm to select an optimal plan from an AND/OR graph plan, together with the design of an important heuristic for selecting a conflict resolution plan that meets the criterion of maximal simultaneity in the execution of subnet compositions, is then presented.

Chapter 6 addresses the on-line agent coordination problem. The concept of a coordinable predicate is presented and the optimal coordination policy is developed. The two novel concepts of coordination-readiness and co-stability are then presented, based on which two on-line coordination strategies are proposed, including one that can achieve significant savings in communication bandwidth.

Finally, Chapter 7 concludes this thesis and presents some directions for future research.

Chapter 2

BACKGROUND AND NOTATION

2.1 Chapter Overview

In this thesis, we follow one focused premise: The problem of discrete-event multiagent coordination shares the same mathematical foundation with that of discrete-event supervisory control [66, 65, 12, 42, 101], and for this reason, well-established concepts and techniques from discrete-event control research could be adapted and applied to address the discrete-event coordination problem. In this chapter, we review relevant concepts and results in supervisory control theory that are adapted to our work in later chapters.

Discrete-event control problems are extensively studied in the Ramdage and Wonham (RW) framework [66, 65] which was first introduced in the 1980's. The framework is founded on languages and automata theory [36] which is shown to be extremely suited for formulating DES control-theoretic concepts such as controllability [66], observability [45] and co-observability [76], and addressing various discrete-event control problems of standard types such as optimal control [66], control under incomplete observation [45] and decentralized control [76, 111]. Many other control-related problems, such as efficient computation of DES controllers [48], and communication among decentralized DES controllers [2, 73], have also been addressed in the RW framework.

The rest of this chapter is organized as follows. In Section 2.2, we review relevant concepts in languages and automata theory which form the foundation of the RW framework. Section 2.3 summarizes some fundamental results in supervisory control of DES's. Section 2.4 elaborates on the mathematical connection between discrete-event control and coordi-

nation which is first discussed in [80, 82], and presents how a discrete-event multiagent coordination problem can be addressed using discrete-event control techniques. Finally, Section 2.5 concludes this chapter.

2.2 Languages and Automata

2.2.1 Languages

Let Σ be a finite alphabet of symbols representing individual events. A string is a finite sequence of events from Σ . Denote Σ^* as the set of all strings, including the empty string ε . A string s' is a prefix of s if $(\exists t \in \Sigma^*) s't = s$, where $s't$ is the string obtained by concatenating t to s' .

A language L over Σ is a subset of Σ^* . Say L_1 is a sublanguage of L_2 if $L_1 \subseteq L_2$. The prefix closure $\bar{L}$ of a language L is the language consisting of all prefixes of its strings. Clearly $L \subseteq \bar{L}$, because any string s in Σ^* is a prefix of itself. A language L is prefix-closed if $L = \bar{L}$.

For an event set $\Sigma_o \subseteq \Sigma$, the natural projection $P_{\Sigma, \Sigma_o} : \Sigma^* \rightarrow \Sigma_o^*$ erases every event $\sigma \in (\Sigma - \Sigma_o)$ from a string $s \in \Sigma^*$, and is defined recursively as follows:

$$P_{\Sigma, \Sigma_o}(\varepsilon) = \varepsilon,$$

and $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)$,

$$P_{\Sigma, \Sigma_o}(s\sigma) = \begin{cases} P_{\Sigma, \Sigma_o}(s)\sigma, & \text{if } \sigma \in \Sigma_o; \\ P_{\Sigma, \Sigma_o}(s), & \text{otherwise.} \end{cases}$$

For $L \subseteq \Sigma^*$, $P_{\Sigma, \Sigma_o}(L) \subseteq \Sigma_o^*$ denotes the language $\{P_{\Sigma, \Sigma_o}(s) \mid s \in L\}$. The inverse image of P_{Σ, Σ_o} , denoted by $P_{\Sigma, \Sigma_o}^{-1}$, is a mapping from Σ_o^* to Σ^* and defined as follows: For $L_o \in \Sigma_o^*$, $P_{\Sigma, \Sigma_o}^{-1}(L_o) = \{L' \subseteq \Sigma^* \mid P_{\Sigma, \Sigma_o}(L') = L_o\}$.

For notational simplicity, whenever $\Sigma_o \not\subseteq \Sigma$, we shall use P_{Σ, Σ_o} instead of the notationally correct $P_{\Sigma, \Sigma \cap \Sigma_o}$ to denote the natural projection from Σ^* to $(\Sigma \cap \Sigma_o)^*$.

Given a set of $n \geq 2$ languages $L_i \subseteq \Sigma_i^*$, $1 \leq i \leq n$, with $\Sigma = \bigcup_{i=1}^n \Sigma_i$, the synchronous product $\parallel$ (or parallel composition) of languages is defined as follows:

$$\prod_{i=1}^n L_i = \bigcap_{i=1}^n P_{\Sigma, \Sigma_i}^{-1}(L_i).$$

It can be easily seen that whenever $\Sigma_i = \Sigma_j$ for all $i \neq j$, the language synchronous product reduces to the set intersection operation, i.e.,

$$\prod_{i=1}^n L_i = \bigcap_{i=1}^n L_i.$$

Let $L \subseteq \Sigma^*$. The Nerode equivalence relation on Σ^* with respect to L is defined as follows: $s \equiv s' \pmod{L} \Leftrightarrow (\forall t \in \Sigma^*)[st \in L \Leftrightarrow s't \in L]$. In other words, $s \equiv s' \pmod{L}$ if and only if s and s' can be continued in exactly the same ways to form a string of L .

Let $||L||$ denote the cardinality of the set of equivalence classes of the Nerode equivalence relation on Σ^* with respect to L . Since Σ^* is countable, $||L||$ is at most countably infinite. If $||L|| < \infty$ then L is said to be a regular language.

2.2.2 Automata

If a language is regular, then it can be generated by an (deterministic finite-state) automaton. An automaton A is a 5-tuple $(X^A, \Sigma^A, \delta^A, x_0^A, X_m^A)$, where (i) X^A is the finite set of states, (ii) Σ^A is the finite set of events, (iii) $\delta^A : \Sigma^A \times X^A \rightarrow X^A$ is the (partial) transition function, (iv) x_0^A is the initial state and (v) $X_m^A \subseteq X^A$ is the subset of marker states.

Write $\delta^A(\sigma, x)!$ to denote that $\delta^A(\sigma, x)$ is defined. An event $\sigma \in \Sigma^A$ is said to be a strictly self-loop event of A if $(\forall x \in X^A)[\delta^A(\sigma, x)! \Rightarrow (\delta^A(\sigma, x) = x)]$. Such an event would never bring A from one state to a different state. For an automaton A , Σ_{loop}^A denotes the set of all its strictly self-loop events.

The definition of δ^A can be extended to $(\Sigma^A)^* \times X^A$ as follows:

$$\delta^A(\varepsilon, x) = x, \text{ and}$$

$$(\forall \sigma \in \Sigma^A)(\forall s \in (\Sigma^A)^*)\delta^A(s\sigma, x) = \delta^A(\sigma, \delta^A(s, x)).$$

The behavior of automaton A can then be described by the prefix-closed language $L(A)$ and the marked language $L_m(A)$. Formally,

$$L(A) = \{s \in (\Sigma^A)^* \mid \delta^A(s, x_0)!\},$$

$$L_m(A) = \{s \in L(A) \mid \delta^A(s, x_0) \in X_m^A\}.$$

A state $x \in X^A$ is reachable if $(\exists s \in (\Sigma^A)^*) \delta^A(s, x_0^A) = x$ and coreachable if $(\exists s \in (\Sigma^A)^*) \delta^A(s, x) \in X_m^A$. Automaton A is said to be reachable if all its states are reachable, and coreachable (or nonblocking) if all its states are coreachable and so $\overline{L_m(A)} = L(A)$. A is then said to be trim if it is both reachable and coreachable. If A is not reachable, then a reachable automaton, denoted by $Reach(A)$, can be computed by deleting from A every state that is not reachable. Thus, $Reach(A)$ generates the same prefix-closed and marked languages as A . If A is not trim, then a trim automaton, denoted by $Trim(A)$, can be computed by deleting from A every state that is either not reachable or not coreachable. Therefore, $Trim(A)$ has no unreachable states and no uncoreachable states, and generates the same marked language as A .

An automaton A can be represented graphically by an edge-labeled directed graph with states represented by nodes, a transition $\delta^A(\sigma, x) = y$ by an edge going from state x to state y with event label σ , the initial state by a node with an entering arrow, and a marker state by a node drawn as a double concentric circle.

On ‘equivalence’ of two automata A_1 and A_2 , we write $A_1 = A_2$ if their edge-labeled directed graphs are identical in structure (including marker states); and $A_1 \equiv A_2$ if the automata generate the same prefix-closed and marked languages. So $(A_1 = A_2)$ implies $(A_1 \equiv A_2)$ but the converse is not true in general.

2.2.3 Composition of Automata

Let A_i , $i \in \{1, 2\}$, be two automata. The composed behavior of A_1 and A_2 can be defined using models of interleaving events formalized as synchronous product and cartesian product, as defined in the following.

Synchronous Product

The synchronous product of A_1 and A_2 is an automaton $A = (X^A, \Sigma^A, \delta^A, x_0^A, X_m^A)$ synthesized with (i) $\Sigma^A = \Sigma^{A_1} \cup \Sigma^{A_2}$, (ii) $X^A = X^{A_1} \times X^{A_2}$, (iii) $X_m^A = X_m^{A_1} \times X_m^{A_2}$, (iv)

$x_0^A = (x_0^{A_1}, x_0^{A_2})$, and (v) $\delta^A = \delta^{A_1} \times \delta^{A_2}$ defined by

$$\delta^A(\sigma, (x_1, x_2)) = \begin{cases} (\delta^{A_1}(\sigma, x_1), \delta^{A_2}(\sigma, x_2)), & \text{if } \sigma \in \Sigma^{A_1} \cap \Sigma^{A_2} \text{ and} \\ & \delta^{A_1}(\sigma, x_1)! \text{ and } \delta^{A_2}(\sigma, x_2)!; \\ (\delta^{A_1}(\sigma, x_1), x_2), & \text{if } \delta^{A_1}(\sigma, x_1)! \text{ and } \sigma \notin \Sigma^{A_2}, \\ (x_1, \delta^{A_2}(\sigma, x_2)), & \text{if } \delta^{A_2}(\sigma, x_2)! \text{ and } \sigma \notin \Sigma^{A_1}; \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

Intuitively, the synchronous product of A_1 and A_2 models a system A , of A_1 and A_2 operating concurrently by interleaving events generated by A_1 and A_2 with synchronization on shared events $\sigma \in \Sigma^{A_1} \cap \Sigma^{A_2}$, such that

- events common to both the automata can occur only if each automata is in a state where such an event is defined;
- events that are not common to both the automata may occur as long as they occur in a sequential order along which they are defined by the respective transition functions of A_1 and A_2 .

It can then be shown [101] that:

$$\begin{aligned} L(A) &= L(A_1) \parallel L(A_2), \\ L_m(A) &= L_m(A_1) \parallel L_m(A_2). \end{aligned}$$

With a slight abuse of notation, the same notation $\parallel$ used for the synchronous product of languages is also used to denote the synchronous product of automata. Thus, if A is the synchronous product of A_1 and A_2 , we write $A = A_1 \parallel A_2$.

The synchronous product of $n \geq 2$ automata $A_1, A_2, \dots, A_n$, denoted by $\parallel_{i=1}^n A_i$, can be defined recursively using the associativity of $\parallel$ [101].

Cartesian Product

The cartesian product of A_1 and A_2 , denoted by $A_1 \sqcap A_2$, is an automaton

$A = (X^A, \Sigma^A, \delta^A, x_0^A, X_m^A)$ synthesized with (i) $\Sigma^A = \Sigma^{A_1} \cup \Sigma^{A_2}$, (ii) $X^A = X^{A_1} \times X^{A_2}$,

(iii) $X_m^A = X_m^{A_1} \times X_m^{A_2}$, (iv) $x_0^A = (x_0^{A_1}, x_0^{A_2})$, and (v) $\delta^A = \delta^{A_1} \times \delta^{A_2}$ defined by

$$\delta^A(\sigma, (x_1, x_2)) = \begin{cases} (\delta^{A_1}(\sigma, x_1), \delta^{A_2}(\sigma, x_2)), & \text{if } \sigma \in \Sigma^{A_1} \cap \Sigma^{A_2} \text{ and} \\ & \delta^{A_1}(\sigma, x_1)! \text{ and } \delta^{A_2}(\sigma, x_2)!; \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

$A_1 \sqcap A_2$ models a system of A_1 and A_2 operating concurrently by synchronization on shared events only. Unlike synchronous product, in the cartesian product automaton $A_1 \sqcap A_2$, there is no transition associated with the events that are not in the set $\Sigma^{A_1} \cap \Sigma^{A_2}$.

It can then be shown that [101]:

$$\begin{aligned} L(A_1 \sqcap A_2) &= L(A_1) \cap L(A_2), \\ L_m(A_1 \sqcap A_2) &= L_m(A_1) \cap L_m(A_2). \end{aligned}$$

Note that if $\Sigma^{A_1} = \Sigma^{A_2}$, then $A_1 \sqcap A_2 \equiv A_1 \parallel A_2$.

The cartesian product of $n \geq 2$ automata $A_1, A_2, \dots, A_n$, denoted by $\prod_{i=1}^n A_i$, can be defined recursively using the associativity of $\sqcap$ [12].

2.3 Supervisory Control of Discrete-Event Systems

We now review the basic discrete-event control problem formulated in the RW framework [66, 65], and a monolithic control synthesis approach [66, 65, 45] which aims to synthesize a single or centralized supervisor to control a DES to satisfy given specifications. This monolithic approach is often contrasted with the modular control synthesis approach (e.g., [104]), which aims to synthesis multiple supervisors to control a given DES. The modular control synthesis approach is reviewed in the Appendix.

2.3.1 Discrete-Event Control Problem

As mentioned in Chapter 1, a DES is a discrete state, event driven system. In the RW framework, a DES is modeled by an automaton $A = \{X^A, \Sigma^A, \delta^A, x_0^A, X_m^A\}$, with X^A represents the state space of the DES and Σ^A represents the set of individual events that can occur in the system. The external behavior of the DES is then described by the languages

generated by A . The language $L(A)$ encapsulates all the strings of events that the system is capable of generating, and the language $L_m(A)$, which is a distinguished subset of $L(A)$, often encapsulates sequences of events that, when executed by the DES, denote the achievement of some system goal or accomplishment of some system task. In practice, A is often constructed as the synchronous product of simpler automata, each of which represents a component of the system.

The discrete-event control problem is formulated in the RW framework under a basic premise that the behavior of DES A is not satisfactory and hence must be modified. In particular, some event sequences generated by the DES may violate a certain desired ordering and therefore should be prevented from happening. For example, these sequences may lead the DES to deadlock or livelock situations, or to the unfair use of shared resources. Thus some means of control are needed to modify the behavior of the (uncontrolled) DES such that only desired event sequences, i.e., those that do not lead to the violation of some pre-specified constraints, can ever be generated. Modifying the behavior of DES A is understood as restricting its generated languages to subsets of $L(A)$. Let another automaton C , which we refer to as a specification automaton, specify the desired constraint over the entire event set Σ^A . In other words, $L(C)$ encompasses all the desired event sequences that we wish to impose on DES A . Then the basic control problem can be informally described as modifying A , or restricting its generated languages, such that the modified DES only generates strings belonging to $L(C)$.

The setting of this basic DES control problem centers around the theoretic idea of discrete-event feedback control. In this setting, a control feature is that some events of the DES are controllable, i.e., their occurrences can be prevented by an external supervisor. To control the DES, a supervisor must be synthesized. This supervisor will, depending on its observation of the event sequences that have occurred thus far, appropriately disable (or prevent from happening) controllable events so that the specified constraint modeled by automaton C will never be violated (see Figure 2.1).

Note that in controlling a DES, uncontrollable events can never be disabled. There are many reasons why an event would be modeled as uncontrollable [12]: (i) It is inherently unpreventable, e.g., a fault event; (ii) it cannot be prevented due to hardware or actuation limitations; or (iii) it is modeled as uncontrollable by choice, e.g., when the event has high system priority or when disabling it is expensive.

Figure 2.1: The control synthesis approach: Given a DES A and a specification automaton C , the objective is to synthesize a supervisor to control DES A such that the constraint specified by C is never violated.

2.3.2 Desired Behavior - Specification Automata

In practice, a specification automaton C is normally constructed from control requirements expressed in natural language statements, such as to avoid a set of forbidden states, enforce a first come, first served policy for a shared resource or disallow the overflow or underflow of a buffer. Constructing such a specification automaton is not a trivial process and might require a lot of skills and knowledge of discrete-event modeling. The difficulty in constructing specification automata has been experienced and reported in many DES applications such as manufacturing [43, 68], transportation [81] or automobile robots [40]. Recently, some research efforts attempt to tackle the problem of constructing specification automata for DES's and provide designers with tools to help specify specification automata and clarify whether the specification automata indeed prescribe the intended control requirements.

In one research direction, a translation algorithm is proposed to automate the specification automaton construction process [79]. The proposed algorithm takes control requirements expressed in a class of temporal logic [67] and automatically translates them into the corresponding specification automata. The supporting argument is that control requirements can be more easily written down in temporal logic [67], which is an expressive and readable language; and more importantly, it is natural language-motivated [67], as writing a certain type of specification thought of in natural language as a temporal logic formula is

relatively easier.

In another research direction, the concept of automaton transparency [59] is introduced to capture the essence of the linguistic description of a specification automaton. Intuitively, in a transparent specification automaton, events that are irrelevant to the specification but can occur in the system are ‘hidden’ in self-loops; while events that are relevant to the specification are highlighted in diligent transitions (i.e., those connecting distinctly different states). The most (or maximally) transparent automaton should visually highlight only sequences of events from a specification-relevant event set of minimal cardinality. Conversely, it should hide events from a specification-irrelevant event set of maximal cardinality. Such maximal transparency could more readily highlight the linguistic description of the specification and, hence, help designers better clarify whether a specified specification automaton completely and correctly reflects the intended control requirement. With this motivation, an algorithm has been developed to maximize the transparency of a given specification automaton [59].

It is envisioned that the reviewed work on temporal logic specification translation [79] and specification transparency maximization [59] could lay the foundation for the future development of a ‘user friendly’ specification synthesis framework. However, in this thesis, we do not consider this specification problem and simply take for granted that specification automata are correct as specified.

2.3.3 Supervisor Synthesis

Given a DES $A = (X^A, \Sigma^A, \delta^A, x_0^A, X_m^A)$ to be controlled, A can be built by the synchronous product of simple automata, each of which models a system component.

To characterize the control and observation capability of a supervisor controlling A , the event set Σ^A partitioned into (i) the controllable event set Σ_c^A and the uncontrollable event set Σ_{uc}^A , and (ii) the observable event set Σ_o^A and the unobservable event set Σ_{uo}^A . In the control context, a supervisor which observes only events in Σ_o^A can modify the behavior of A by disabling only the events in Σ_c^A . Formally, a supervisor S is an automaton over Σ^A ($\Sigma^S = \Sigma^A$) that satisfies the following properties:

1. S is Σ_{uc}^A -enabling, namely, $(\forall s \in (\Sigma^A)^*)(\forall \sigma \in \Sigma_{uc}^A) [(s \in L(S) \cap L(A)) \text{ and } (s\sigma \in L(A))] \Rightarrow (s\sigma \in L(S) \cap L(A))$.

2. S is feasible, namely, $(\forall \sigma \in \Sigma_{uo}^A)[((\exists x \in X^S)\delta^S(\sigma, x)!) \Rightarrow \sigma \in \Sigma_{loop}^S]$.

Supervisor S tracks and controls the behavior of DES A . It changes state according to the events generated by A and enables (or allows to occur) at each of its state x only those events σ where $\delta^S(\sigma, x)!$. That S is Σ_{uc}^A -enabling guarantees that uncontrollable events in Σ_{uc}^A will never be prevented from happening. That S is feasible means that its state change can only be triggered by the occurrence of events in Σ_o^A , since these are the only events it can observe.

That S supervises A , effectively, generates the languages of $S \sqcap A$. The prefix-closed behavior of the controlled DES, $L(S \sqcap A)$, consists of those strings in $L(A)$ whose every event is enabled by S . The marked behavior $L_m(S \sqcap A)$ consists of those strings in $L(S \sqcap A)$ that are marked by both S and A . Supervisor S is said to be nonblocking (for DES A) if every string generated by $S \sqcap A$ can be completed to a marked string, i.e., $\overline{L_m(S \sqcap A)} = L(S \sqcap A)$.

Formally, a general problem statement of supervisory control may be given as follows.

Problem 2.1. *Supervisory Control Problem (SCP): Given a DES A and a specification automaton C , synthesize a nonblocking supervisor S , which controls only Σ_c^A and observes only Σ_o^A , such that $L_m(S \sqcap A) \subseteq L_m(A) \cap L_m(C)$.*

In addressing SCP, a fundamental theorem in supervisory control theory [45] states that there is a nonblocking supervisor S , which observes only Σ_o^A and controls only Σ_c^A , such that $L_m(S \sqcap A) = K$ if and only if

1. K is controllable w.r.t A and Σ_c^A ; and
2. K is observable w.r.t A and P_{Σ^A, Σ_o^A} .

The concepts of language controllability and observability, which are first respectively introduced in [66] and [45], are reviewed below.

Definition 2.1 ([66]). *A language $K \subseteq L_m(A)$ is said to be controllable w.r.t A and Σ_c^A (or just controllable if Σ_c^A is understood) if $\overline{K}\Sigma_{uc}^A \cap L(A) \subseteq \overline{K}$, where the notation $\overline{K}\Sigma_{uc}^A$ denotes the set of strings of the form $s\sigma$ for some $s \in \overline{K}$ and $\sigma \in \Sigma_{uc}^A$.*

In other words, K is controllable provided no $L(A)$ -string which is already a prefix of some string in K , that when followed by an uncontrollable event in Σ_{uc}^A would exit from $\overline{K}$. If K is uncontrollable, there always exists a sublanguage of K (which may be empty) that is controllable [66]. Moreover, it has been shown that the class of controllable languages contained in a given language K is partially ordered by inclusion and closed under arbitrary unions, and therefore contains an unique supremal element called the supremal controllable sublanguage of K with respect to A [66]. In essence, the supremal controllable sublanguage of K is equal to the union of all the controllable sublanguages of K . As a practical matter of implementation, given an automaton C , the procedure $Supcon(C, A)$ [103] has been developed to compute a nonblocking automaton S such that $L_m(S)$ is the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$. The procedure can be implemented to run with polynomial time complexity of $O(|X^C|^2 |X^A|^2)$ [103].

Definition 2.2 ([45]). *A language $K \subseteq L_m(A)$ is said to be observable w.r.t A and P_{Σ^A, Σ_o^A} (or just observable if Σ_o^A is understood) if*

$$(\forall s, s' \in (\Sigma^A)^*) P_{\Sigma^A, \Sigma_o^A}(s) = P_{\Sigma^A, \Sigma_o^A}(s') \implies$$

1. $(\forall \sigma \in \Sigma^A)((s\sigma \in \overline{K} \text{ and } s' \in \overline{K} \text{ and } s'\sigma \in L(A)) \Rightarrow s'\sigma \in \overline{K});$
2. $(s \in K \text{ and } s' \in \overline{K} \cap L_m(A)) \Rightarrow s' \in K.$

Intuitively, the above conditions ensure that Σ_o^A provides a sufficient view for a supervisor to determine a necessary control or marking action when it is controlling A to achieve K as the marked language of the controlled system. The observability property of a language K can be checked in polynomial time [91]. However, unlike controllability, observability is not preserved under union operation, i.e., the union of two observable languages is not guaranteed to be observable [45]. As a result, the supremal observable language contained in an unobservable language does not exist. Therefore, if a language K is unobservable then it may contain several (incomparable) maximal observable languages [45].

Under complete observation, i.e., $\Sigma_o^A = \Sigma^A$, an arbitrary sublanguage K of $L_m(A)$ is observable. Thus, automaton $S = Supcon(C, A)$ can be used as (the internal model for) a supervisor solution for SCP. Under supervision of S , the controlled system $S \sqcap A$ is nonblocking and generates the marked language $L_m(S \sqcap A) = L_m(S)$ - the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$. Thus, supervisor S is said to be minimally

interventive (or minimally restrictive) since it only disables a controllable event (in Σ_c^A) when absolutely necessary. In the RW framework, a minimally interventive supervisor is also referred to as an optimal supervisor.

The optimal supervisor $Supcon(C, A)$, however, can be larger in state size than it is necessary to achieve the same control actions. The reason is that $Supcon(C, A)$ has ‘embedded’ in it all the a priori transitional constraints embodied in the free behavior of the system A itself, as well as some auxiliary constraints [88, 95]. While finding a minimal state supervisor achieving the same control actions as $Supcon(C, A)$ is a NP-hard problem [88], a heuristic reduction procedure called *Supreduce*, which is of polynomial complexity, has been developed [88]. Given $Supcon(C, A)$ and A , the procedure computes and returns a reduced state size supervisor $S_{reduced}$ which is control equivalent to the given supervisor $Supcon(C, A)$, i.e., $S_{reduced} \sqcap A \equiv Supcon(C, A)$. The procedure also provides a lower bound on the state size of the minimal state supervisor, and the computed supervisor $S_{reduced}$ is actually minimal state if its size matches this bound.

Under partial observation, i.e., $\Sigma_o^A \subsetneq \Sigma^A$, $L_m(A) \cap L_m(C)$ may contain several (incomparable) maximal observable languages. Therefore SCP may not have an optimal solution (in terms of a minimally interventive supervisor). However, it has been shown that if every controllable event is also observable, i.e., $\Sigma_c^A \subseteq \Sigma_o^A$, then the supremal controllable and observable sublanguage of $L_m(A) \cap L_m(C)$ does exist and, consequently, SCP does have optimal solutions [12]. Moreover, if $\Sigma_c^A \subseteq \Sigma_o^A$, an optimal supervisor solution for SCP can be computed in polynomial time [12].

2.4 Multiagent Coordination as Control Synthesis

The mathematical connection between discrete-event control and coordination is first presented in [80, 82]. There, a multiagent planning approach to logical coordination synthesis that views a class of distributed agents as DEP’s is proposed. The coordination synthesis problem involves finding a coordination module (CM) for every agent, using which their coordinated interactions would never violate some specified inter-agent constraint. The authors first show explicitly that, though conceptually different, the well-researched problem of supervision in control science and the problem of distributed agent coordination planning in computer agents science are mathematically related. This basic result enables the application of the vast body of knowledge and associated synthesis tools already founded

in discrete-event control theory for automatic coordination synthesis of distributed agents. In this section, we briefly review the idea of designing or planning for multiagent coordination as control synthesis, which sets the basic research direction for developing a new distributed agent coordination framework in this thesis.

2.4.1 Multiagent Coordination Planning

The basic multiagent coordination planning problem considered is to modify a system of n interacting agents such that the modified system as a whole is nonblocking and conforms to some inter-agent constraint. The free behavior of each agent A_i is modeled as an automaton interpreted as a DEP. The inter-agent constraint is qualitative (non-numerical) and also specified by some automaton C over the events of all the agents. The basic coordination problem is to modify the multiagent system $A = \parallel_{i=1}^n A_i$ such that the modified system A' is nonblocking and only generates strings belonging to $L(C)$.

For each agent A_i , its event set is partitioned into the controllable event set $\Sigma_c^{A_i}$ and the uncontrollable event set $\Sigma_u^{A_i}$. Informally, a controllable event can be disabled (hence prevented from happening) by the owner agent in coordinating with others, while an uncontrollable event cannot be. Chapter 3 will provide a more detailed discussion in this respect. Consider another automaton S'_i , with $\Sigma^{A_i} \subseteq \Sigma^{S'_i}$. Then S'_i can be a CM of agent A_i provided S'_i is $\Sigma_{uc}^{A_i}$ -enabling, namely,

$$(\forall s \in (\Sigma^{S'_i})^*)(\forall \sigma \in \Sigma_{uc}^{A_i})[s \in L(S'_i \parallel A_i) \text{ and } P_{\Sigma^{S'_i}, \Sigma^{A_i}}(s)\sigma \in L(A_i)] \Rightarrow [s\sigma \in L(S'_i \parallel A_i)].$$

Informally, $\Sigma_{uc}^{A_i}$ -enabling means that when agent A_i is coordinating through S'_i (via $\parallel$), it can never disable its uncontrollable events.

A general problem statement of multiagent coordination planning may then be given as follows.

Problem 2.2. *Given a system $A = \parallel_{i=1}^n A_i$ of n agents and an inter-agent constraint C , find a CM S'_i for each agent A_i such that $L_m(\parallel_{i=1}^n (A_i \parallel S'_i)) \subseteq L_m(C)$.*

Interestingly, it turns out that the Control Problem 2.1 and Coordination Problem 2.2 are mathematically equivalent in the sense of Theorem 2.1 below. The theorem presents the result for two agents, but can be easily extended to multiple agents.

Theorem 2.1. *Given automata S , A_1 and A_2 , with $\Sigma^S = \Sigma^{A_1} \cup \Sigma^{A_2}$,*

$$S \sqcap (A_1 \parallel A_2) \equiv (A_1 \parallel S'_1) \parallel (S'_2 \parallel A_2)$$

and

$$\Sigma^{S'_1} \cap \Sigma^{S'_2} = (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2}),$$

where S'_i , $i \in \{1, 2\}$, is automaton S , but with all its strictly self-loop events not defined in agent A_i , i.e., $\sigma \in \Sigma_{loop}^S - \Sigma^{A_i}$, removed.

Proof. Since S and $A_1 \parallel A_2$ share the same event set Σ^S and $\parallel$ is associative and commutative, it follows that

$$\begin{aligned} S \sqcap (A_1 \parallel A_2) &\equiv S \parallel (A_1 \parallel A_2) \\ &\equiv S \parallel A_1 \parallel A_2 \\ &\equiv A_1 \parallel S \parallel A_2. \end{aligned}$$

And since S'_i is S but with all its strictly self-loop events in S not defined in agent A_i , i.e., $\sigma \in \Sigma_{loop}^S - \Sigma^{A_i}$, removed, it follows by the definition of the *synchronous* operator $\parallel$ that $S \equiv S'_1 \parallel S'_2$. Thus,

$$\begin{aligned} S \sqcap (A_1 \parallel A_2) &\equiv A_1 \parallel (S'_1 \parallel S'_2) \parallel A_2 \\ &\equiv (A_1 \parallel S'_1) \parallel (S'_2 \parallel A_2). \end{aligned}$$

By set-theoretic manipulations,

$$\begin{aligned} \Sigma^{S'_1} \cap \Sigma^{S'_2} &= \{\Sigma^S - (\Sigma_{loop}^S - \Sigma^{A_1})\} \cap \{\Sigma^S - (\Sigma_{loop}^S - \Sigma^{A_2})\} \\ &= \{(\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma_{loop}^S \cap \Sigma^{A_1})\} \cap \{(\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma_{loop}^S \cap \Sigma^{A_2})\} \\ &= (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S) \\ &= \{(\Sigma^S \cup \Sigma^{A_1} \cap \Sigma^{A_2}) - \Sigma_{loop}^S\} \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S) \\ &= (\Sigma^S - \Sigma_{loop}^S) \cup \{(\Sigma^{A_1} \cap \Sigma^{A_2} - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S)\} \\ &= (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2}). \end{aligned}$$

Hence the result. □

Theorem 2.1 may be interpreted as follows. Suppose automaton $A = A_1 \parallel A_2$ models

a system. Then the left-hand side (of Theorem 2.1) can be viewed as a supervisor S controlling the system A (of possibly interacting DEP's A_1 and A_2). The right-hand side can be viewed as distributed agents, each (with its free behavior) modeled by A_i , $i \in \{1, 2\}$, coordinating between themselves via their respective CM's S'_i that each is deemed to be 'equipped' with. This suggests that we can apply existing control synthesis methods to obtain a supremal controllable S (which is nonblocking), from which every S'_i can be obtained as implied in Theorem 2.1. It means we can obtain CM's for agents in a system A such that the coordinating behavior $S \parallel A$ is nonblocking and does not contradict the inter-agent constraint C . As $S'_1 \parallel S'_2 = S$, the coordination is minimally interventive, meaning that the coordinating agents will not unnecessarily disable their own controllable events since the admissible coordination space $S \parallel A$ (due to the supremal controllable and nonblocking automaton S of C) is feasibly the least constrained that still conforms to the specified inter-agent constraint C .

2.4.2 Coordination Planning Methodology

An important implication of Theorem 2.1 presented in the preceding section and the discussion that followed is that control synthesis can be adapted and applied as a new multiagent planning approach to coordination. And the approach can be carried out without 'reinventing the wheel' through a planning methodology. This approach allows automatic synthesis of minimally interventive CM's. The proposed methodology consists of the following main planning steps.

- **Step 1: Modeling**

Create as input all automaton models of the free agents A_i and their inter-agent constraint C .

- **Step 2: Control Synthesis**

- Compute a nonblocking maximal permissive supervisor $S = \text{Supcon}(C, A)$.
- Reduce the state size of S by, for example, applying procedure *Supreduce* [88].

- **Step 3: Coordination Synthesis**

- Compute Σ_{loop}^S as the set of strictly self-loop events of S .

- Following Theorem 2.1 to compute for each agent a CM S'_i from S by removing from S all strictly self-loop events not defined in agent A_i .

Note that one obvious coordination solution is $S'_i = S$. In this case, the agents communicate synchronously all of their (local) events to achieve conformance to inter-agent constraints. However, this solution may entail unnecessary events communicated between the agents. Theorem 2.1 has clearly revealed the possibility of a better coordination solution with regard to less synchronous communication. It is the basis for extending the control methodology (Steps 1 and 2) with Step 3, as in the coordination planning methodology presented above.

2.4.3 Illustrative Example

We now present an example to illustrate the use of the proposed planning methodology for coordination design of a multiagent system.

Figure 2.2: Multiagent coordination planning for a distributed train system.

The system under study is a different version of the train controller system presented in [93]. The distributed system [see Figure 2.2(a)] has two train agents, modeled by automata T_1 and T_2 as shown in Figures 2.2(b) and 2.2(c), but no central controller. The modeled train behaviors should be quite self-explanatory, with $\Sigma = \Sigma_c \cup \Sigma_u$, where we arbitrarily fix $\Sigma_c = \{1entered, 2entered\}$ and $\Sigma_u = \{1arrived, 1left, 2arrived, 2left\}$. The train agents, one eastbound (EB) and one westbound (WB), each occupies its own loop track. But at one point, both tracks pass through a tunnel. There is no room to accommodate both trains going past each other in the tunnel. Unlike in [93], there are no traffic lights at both ends of the tunnel. Both trains are equipped with a signaller, using which they can send signals to communicate with each other.

Given this scenario, one possible solution is to let the train agents coordinate between themselves so as to respect the following inter-agent constraint:

The train agents are allowed access to the tunnel on a first arrival first access (FAFA) basis, and such that they are never both in the tunnel at the same time.

The textual description of the desired constraint can be formalized by an automaton C as shown in Figure 2.2(d). How this can be done is simply taken for granted here. We refer the reader to the paper [79] on how such an automaton may be more easily determined with the aid of a specification translator. In fact, the reader may convince himself that the automaton C represents the desired constraint. The problem then becomes that of formally synthesizing CM's for the train agents. This can be systematically done using the automated planning methodology proposed in the previous section.

The result after completing Step 2 is a simplified supervisor S which, for this system, is found to be minimal state (by *Supreduce*). It is shown in Figure 2.3. Following Step 3, we obtain $\Sigma_{loop}^S = \{1entered, 2entered\}$, using which the event $\sigma \in \Sigma_{loop}^S - \Sigma^{T_i}$, $i \in \{1, 2\}$ is projected or masked out from S to obtain the CM S'_i for train agent T_i , as shown in Figure 2.4. To elaborate, using these CM's means: Agent T_1 must inform agent T_2 whenever any of its events, $1arrived$ and $1left$, occurs, but need not do so when event $1entered$ occurs; and agent T_2 reciprocates in turn.

To meet the specified inter-agent constraint C , a clear advantage of our approach is that it offers unintuitive design insights, informing us the events (namely, $\Sigma_{sync}^S \cap \Sigma^{T_i} = \{iarrived, ileft\}$) that agent T_i , $i \in \{1, 2\}$, would need to communicate to the other agent T_j , $i \neq j \in \{1, 2\}$, via synchronous messaging, and those (namely, $(\Sigma_{loop}^S - \Sigma^{T_j}) =$

Figure 2.3: A simplified supervisor S .

$\{ientered\}$) that it need not. In our opinion, without the formal synthesis methodology, even for this simple train example, it would have been quite difficult to determine which events can be in Σ_{loop}^S that need not be coordinated between the train agents.

Figure 2.4: Coordination modules for the distributed train agents.

2.5 Chapter Summary

This chapter has reviewed important concepts and results of the languages and automata theory and the supervisory control theory of DES's that we will adapt to our work in later chapters. The mathematical connection between discrete-event control and coordination has also been discussed. This connection is exploited in the theoretical and algorithmic development of a new multiagent coordination framework presented in this thesis.

Chapter 3

FUNDAMENTAL AGENT COORDINATION PROBLEM

3.1 Chapter Overview

This chapter studies the first and the most fundamental problem of discrete-event multi-agent coordination. We introduce a formal, domain independent framework to address the problem of synthesizing coordination modules (CM's) or built-in strategies for distributed discrete-event agents coordinating to satisfy a pre-specified global inter-agent constraint.

In our framework, a coordinating agent is modeled as a DEP and is represented by a (deterministic) automaton [36]. From an agent planning viewpoint, an agent automaton represents the free (or unconstrained) but fixed-by-design behavior of the corresponding agent, and is formulated independently by an agent designer to encompass all possible local ways for the agent to achieve its local goal (or complete its own design task). To accomplish its goal, an agent may execute one of possibly many event sequences that bring its automaton from the initial state to a marker state. The sequences that the agent actually executes would depend on contingencies arising in the course of its operation.

A system of multiple discrete-event agents freely interacting can then be represented by a model of interleaving events formalized as the synchronous product [12, 101] of all individual agent automata. The agents would need to coordinate among themselves if, due to system needs or limitations, some of the event sequences generated by the product automaton are undesirable and must be prevented from happening. The coordination goal is

then to specify for the agents, an inter-agent constraint stipulating the set of desirable event sequences that can be generated during agents' interaction to accomplish their local goals. The problem of interest can then be defined as synthesizing, for each agent, a CM specifying, at each event execution step, the next events to execute or inhibit during multiagent interaction, so that none of the undesirable event sequences are ever executed. A CM is also represented as an automaton.

In synthesizing CM's for distributed agents, we focus on CM solutions that are minimally interventive with the agents' local plans. In essence, using minimally interventive CM's, coordinating agents only disable (or prevent from execution) their own events provided the execution of these events can lead to eventual violation of the constraints. In other words, such CM's enable coordinating agents to have maximal autonomy in selecting event sequences to accomplish their local goals. Such CM's are therefore of significant importance to "intelligent" agents capable of opportunistically selecting and executing the most appropriate event sequences.

Our formulated problem and proposed synthesis solution can be viewed as a multiagent planning approach [37, 24] to coordination. Previous work [80, 82] reviewed in Chapter 2 has shown that although conceptually different, discrete-event multiagent coordination and supervisory control [66, 65] share the same mathematical foundation. A major implication of significant interest is that the vast body of knowledge and associated synthesis tools from supervisory control [66, 65, 12, 101] can be adapted and applied for the automatic synthesis of CM's for coordinating agents. This has motivated a method using control synthesis for multiagent planning [82]. By first considering multiagent systems as DES's to be controlled and inter-agent constraints as control specifications, the method proposes to solve the coordination problem by synthesizing a global supervisor to control the system to satisfy the specification, and then using it to construct a CM for every agent. Examples have been given to demonstrate the effectiveness the proposed method [80, 82]. However, there is one inefficiency drawback: Every CM is constructed almost similarly to the supervisor, which results in their nearly identical structure. So, although conceptually illuminating, the method does not consider that the inter-agent constraint may impose different restrictions on different agents. As a result, some agents may have to interact and communicate more than it is necessary.

In this chapter, we develop a novel approach that allows CM's to be individually synthesized for each agent. As a result, an agent can expect a simpler CM if it is not tightly

restricted by the inter-agent constraint. In doing so, we first introduce the concept of a coordinable language and show it is the necessary and sufficient existence condition of CM's for distributed agents to achieve conformance to a pre-specified inter-agent constraint. We then propose a synthesis algorithm for computing CM's if they do exist. Importantly, the synthesized CM's are not only minimally interventive with the agents' local plans, but also ensure that communication between the agents is made only when necessary. Each of the CM's can also be efficiently implemented in terms of memory requirements as their state size may be greatly reduced, though not necessarily minimized, for each agent model. Such CM's are said to be near-optimal.

The rest of the chapter is organized as follows. In Section 3.2, we formalize our fundamental discrete-event coordination problem. We then present a procedure for the synthesis of near-optimal CM's in Section 3.4, along with an illustrative design synthesis example in Section 3.5. A discussion with related work in Section 3.6 highlights the importance of our discrete-event framework, and distinguishes our coordination problem from various control problems. Finally, Section 3.7 concludes this chapter.

3.2 Discrete-Event Agents and Coordination

3.2.1 Discrete-Event Agents and Constraints

Consider a multiagent system of $n \geq 2$ agents modeled by the respective automata $A_i = (X^{A_i}, \Sigma^{A_i}, \delta^{A_i}, x_0^{A_i}, X_m^{A_i})$, $1 \leq i \leq n$, where $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $i \neq j$. From an agent planning viewpoint, automaton A_i is viewed as the local plan of an agent, referred to as agent A_i (or just A_i), encompassing all possible local ways to achieve the agent's local goal. Note that, since each agent is assumed to formulate its local plan independently, the total absence of shared events among coordinating agents, i.e., $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $i \neq j$, is not uncommon.

If the agents operate independently, then the synchronous product $A = \prod_{i=1}^n A_i$ which encompasses every possible way of interleaving event sequences of the n automata, can be used to describe the system evolution. The agents A_i ($1 \leq i \leq n$) would need to coordinate between themselves if, due to system needs or limitations, the execution of some event sequences in $L(A)$ is undesirable and should be prevented. If a language $K \subseteq L(A)$ encompasses all the desirable event sequences, then at an appropriate level of abstraction,

the coordination problem can be defined as modifying the system in a certain way so that none of those sequences in $L(A) - K$ can ever be generated.

3.2.2 Uncontrollable and Controllable Events in an Agent Model

The event set Σ^{A_i} of agent A_i is partitioned into the controllable set $\Sigma_c^{A_i}$ and the uncontrollable set $\Sigma_{uc}^{A_i}$. Controllable events are those that can be disabled (or prevented from happening) by the owner agents when coordinating with other agents, while uncontrollable events are those that cannot be.

Interpreted from the coordination viewpoint, which slightly differently from the context of supervisory control (see Chapter 2), an event that is uncontrollable may or may not be prevented by the owner agent in the agent's process of pursuing its local goal: This usually depends on the engineered dynamics inside the agent's local state. One can in fact think of an uncontrollable event as one that is predetermined to be inherently autonomous, i.e., it can occur solely at the free will of an agent that owns it. Such a free will is exerted by the underlying local state dynamics of the agent. For this reason, an uncontrollable must not be prevented from happening by the owner agent for the sole purpose of coordinating with other agents.

Following the discrete-event modeling of a multiagent system, a system modeler has to pre-specify each event as either controllable or uncontrollable. As a rule, an event is pre-specified as uncontrollable if disabling the event is expensive or impossible. Machine breakdown is a hard example of an uncontrollable event. It can occur at the free will of the machine agent; here of course, the free will is exerted by the possibly unrestrained ageing dynamics of the machine agent. Customer arrival at a banking ATM is a softer example of an uncontrollable event; it is inhibitatable by an agent (the customer) that owns it. But in coordination, not only is prohibiting such an event judged as unnecessary, it is also often an expensive operation, hence pre-specified as uncontrollable.

An event that is controllable can only be inhibited by an agent that owns it when coordinating with other agents. While a disabled event is certainly prevented from occurring, whether or not an enabled event can be executed depends on the agent's underlying mechanism. A system modeler can freely pre-specify an event to be controllable as he sees fit, provided that some mechanism can be implemented for the agent to actually prohibit or execute the event as necessary. An example of an event that may be pre-specified as

controllable is a traffic green light turned on.

In the rest of this chapter, let $\Sigma_c^A = \bigcup_{i=1}^n \Sigma_c^{A_i}$ and $\Sigma_{uc}^A = \bigcup_{i=1}^n \Sigma_{uc}^{A_i}$.

3.2.3 Coordination Modules and Coordinated Behavior

In enabling distributed agents to coordinate, each agent A_i is equipped with a CM modeled by an automaton S_i defined as follows.

Definition 3.1. A CM for agent A_i ($1 \leq i \leq n$) is an automaton S_i with the following properties:

1. $\Sigma^{S_i} = \Sigma^{A_i} \cup \bigcup_{1 \leq j \leq n, j \neq i} \text{ComSet}(S_i, A_j)$, where $\text{ComSet}(S_i, A_j) \subseteq \Sigma^{A_j}$, ($1 \leq j \leq n, j \neq i$). Σ^{S_i} is called the coordination event set for agent A_i , and $\text{ComSet}(S_i, A_j)$ is the subset of events that agent A_j needs to communicate to A_i to synchronize S_i .
2. S_i is $\Sigma_{uc}^{A_i}$ -enabling, namely, $(\forall s \in (\Sigma^{S_i})^*)(\forall \sigma \in \Sigma_{uc}^{A_i}) [s \in L(S_i \parallel A_i) \text{ and } P_{\Sigma^{S_i}, \Sigma^{A_i}}(s)\sigma \in L(A_i)] \Rightarrow [s\sigma \in L(S_i \parallel A_i)]$.
3. $\forall 1 \leq i, j \leq n, i \neq j$, S_i and S_j are cooperative, namely, $(\forall s \in (\Sigma^A)^*)(\forall \sigma \in \text{ComSet}(S_i, A_j)) [P_{\Sigma^A, \Sigma^{A_j}}(s)\sigma \in L(A_j) \text{ and } P_{\Sigma^A, \Sigma^{S_j}}(s)\sigma \in L(S_j)] \Rightarrow [P_{\Sigma^A, \Sigma^{S_i}}(s)\sigma \in L(S_i)]$.

CM S_i is used by agent A_i to implement its local coordination process. Σ^{S_i} represents the observation capability of A_i : A string $s \in L(A)$ is perceived by A_i as $P_{\Sigma^A, \Sigma^{S_i}}(s)$. An agent can observe all its events, hence $\Sigma^{A_i} \subseteq \Sigma^{S_i}$.

Figure 3.1: Two agents A_1 and A_2 coordinating through their respective CM's S_1 and S_2 : Following the execution of a string $s \in L(A_1 \parallel A_2)$, agent A_i updates the state of its CM S_i to $x_i = \delta^{S_i}(P_{\Sigma^A, \Sigma^{S_i}}(s), x_0^{S_i})$. Only every event $\sigma_i \in \Sigma^{A_i}$ that is defined at x_i (i.e., $\delta^{S_i}(\sigma_i, x_i)!$) is enabled and so can be executed next by A_i . The result is that the system behavior is restricted to a sublanguage of $L(A_1 \parallel A_2)$.

The coordination between discrete-event agents through their respective CM's is depicted in Figure 3.1 and explained as follows. Following the execution of a string $s \in L(A)$, agent A_i , $1 \leq i \leq n$, due to partial observation, perceives only $s_i = P_{\Sigma^A, \Sigma^{S_i}}(s)$ and updates the state of CM S_i to $x_i = \delta^{S_i}(s_i, x_0^{S_i})$. Only every event $\sigma_i \in \Sigma^{A_i}$ that is defined at x_i (i.e., $\delta^{S_i}(\sigma_i, x_i)!$) is then enabled (allowed to be executed next) by A_i . That S_i is $\Sigma_{uc}^{A_i}$ -enabling guarantees that uncontrollable events are always enabled (hence are never prevented from being executed). With $i \neq j$, that S_i and S_j are cooperative ensures that whenever an event $\sigma \in ComSet(S_i, A_j)$ is executed and communicated by agent A_j , A_i can update its CM accordingly. The result is that the system behavior is restricted to a sublanguage of $L(A)$. By Definition 3.1, the events in the set $\bigcup_{1 \leq j \leq n, j \neq i} ComSet(S_i, A_j)$ or equivalently $(\Sigma^{S_i} - \Sigma^{A_i})$ are those to be communicated to agent A_i by the other agents in the system during coordination. It then follows that the events in the set $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i})$, called the system communication set, are those to be communicated among the agents during coordination.

Let $A_i^{S_i}$ denote agent A_i coordinating with the other agents using its (local) CM S_i , and $\parallel_{i=1}^n A_i^{S_i}$ denote the system of n agents A_i coordinating among themselves through their respective CM's. From the foregoing discussions, the behavior of the coordinated system can be defined as follows.

Definition 3.2 (Coordinated Behavior). *The behavior of the coordinated system $\parallel_{i=1}^n A_i^{S_i}$ is characterized by two languages $L(\parallel_{i=1}^n A_i^{S_i})$ and $L_m(\parallel_{i=1}^n A_i^{S_i})$ defined as follows.*

1. *Prefix-closed coordinated behavior $L(\parallel_{i=1}^n A_i^{S_i})$*

$$(a) \ \varepsilon \in L(\parallel_{i=1}^n A_i^{S_i}),$$

$$(b) \text{ if } s \in L(\parallel_{i=1}^n A_i^{S_i}) \text{ and } \sigma \in \Sigma^{A_i} \text{ and } s\sigma \in L(A) \text{ and } P_{\Sigma^A, \Sigma^{S_i}}(s)\sigma \in L(S_i) \text{ then}$$

$$s\sigma \in L(\parallel_{i=1}^n A_i^{S_i}),$$

$$(c) \text{ no other strings belong to } L(\parallel_{i=1}^n A_i^{S_i}).$$

2. *Marked coordinated behavior $L_m(\parallel_{i=1}^n A_i^{S_i})$*

$(\forall s \in L(A)) s \in L_m(\parallel_{i=1}^n A_i^{S_i})$ if and only if $[s \in L(\parallel_{i=1}^n A_i^{S_i})$ and $s \in L_m(A)$ and $(\forall 1 \leq i \leq n) P_{\Sigma^A, \Sigma^{S_i}}(s) \in L_m(S_i)]$.

In Definition 3.2, $L_m(\parallel_{i=1}^n A_i^{S_i})$ consists of every string $s \in L(\parallel_{i=1}^n A_i^{S_i}) \cap L_m(A)$ whose projection $P_{\Sigma^A, \Sigma^{S_i}}(s)$ is marked by the respective CM S_i . CM set $\{S_i \mid 1 \leq i \leq n\}$ is then said to be nonblocking if every string generated during coordination can be completed to a marked string, i.e., $L_m(\parallel_{i=1}^n A_i^{S_i}) = L(\parallel_{i=1}^n A_i^{S_i})$.

It can be easily shown that the prefix-closed and marked behaviors of the coordinated system can be equivalently represented by the respective languages generated by the synchronous product $\parallel_{i=1}^n (A_i \parallel S_i)$. It follows that $A_i^{S_i}$ can be modeled as $A_i \parallel S_i$, and the coordinated system as a whole can be represented by $\parallel_{i=1}^n (A_i \parallel S_i)$.

Definition 3.3 (Coordinable Language). *Let $\Sigma_{com} \subseteq \Sigma^A$. A language $K \subseteq L_m(A)$ is coordinable w.r.t A and Σ_{com} if*

1. K is controllable w.r.t A and $\Sigma_c^A = \bigcup_{i=1}^n \Sigma_c^{A_i}$, and
2. K is observable w.r.t A and $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}$, $1 \leq i \leq n$.

According to Definition 3.3, K is coordinable with respect to A and Σ_{com} if

1. K is controllable, meaning that if each agent coordinates properly (by appropriately enabling and disabling its own controllable events), then the coordinated system behavior will conform to K , i.e., $L_m(\parallel_{i=1}^n A_i^{S_i}) = K$, and
2. K is observable by each individual agent A_i equipped with observation capability $\Sigma^{A_i} \cup \Sigma_{com}$, meaning that A_i has sufficient information for determining its coordinating actions (that ensure the conformance of the coordinated system behavior to K).

Intuitively, if the above two conditions are satisfied, then a set of CM's, one for each agent, can be synthesized such that the overall system behavior conforms to K and the system communication event set is Σ_{com} . This is formally stated in Theorem 3.1. The proof of this fundamental theorem requires a procedure called *CM* which computes a CM

Procedure $CM(S, \Sigma^{CM_i})$ **begin**

- 1 Let $\pi : X_p \rightarrow 2^{X^S} - \{\emptyset\}$ be a bijective mapping;
- 2 Compute automaton $S'_i = (X_p, \Sigma^{CM_i}, \delta^{S'_i}, x_0^{S'_i}, X_m^{S'_i})$ where:
 - $x_0^{S'_i} \in X_p$ with $\pi(x_0^{S'_i}) = \{\delta^S(s, x_0^S) \mid P_{\Sigma^S, \Sigma^{CM_i}}(s) = \varepsilon\}$;
 - $X_m^{S'_i} = \{x_p \in X_p \mid (\exists s \in L_m(S)) \delta^S(s, x_0^S) \in \pi(x_p)\}$;
 - $(\forall \sigma \in \Sigma^{CM_i})(\forall x_p \in X_p) [\delta^{S'_i}(\sigma, x_p)! \Leftrightarrow (\exists s\sigma \in L(S)) \delta^S(s, x_0^S) \in \pi(x_p)]$;

When defined, $\delta^{S'_i}(\sigma, x_p) = x'_p$ with
 $\pi(x'_p) = \{\delta^S(s', x) \mid x \in \pi(x_p), P_{\Sigma^S, \Sigma^{CM_i}}(s') = \sigma\}$;
- 3 Return $S_i = Trim(S'_i)$;

end

S_i for an agent A_i , given automaton S and event set $\Sigma^{CM_i} \subseteq \Sigma^S$ for $\Sigma^{S_i} = \Sigma^{CM_i}$, with $L(S_i) = P_{\Sigma^S, \Sigma^{CM_i}}(L(S))$ and $L_m(S_i) = P_{\Sigma^S, \Sigma^{CM_i}}(L_m(S))$.

Computation-wise, procedure CM is the same as the well-known procedure [12] used in supervisory control to compute the natural projection (see Chapter 2) of an automaton onto an event set.

Theorem 3.1. *Given $n \geq 2$ agent automata A_i , $1 \leq i \leq n$, with $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $i \neq j$. Let $A = \parallel_{i=1}^n A_i$, $\emptyset \neq K \subseteq L_m(A)$ and $\Sigma_{com} \subseteq \Sigma^A$. Then, there exists a CM set $\{S_i \mid 1 \leq i \leq n\}$, where S_i is for A_i , such that $L_m(\parallel_{i=1}^n A_i^{S_i}) = K$, $L(\parallel_{i=1}^n A_i^{S_i}) = \overline{K}$ and $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \Sigma_{com}$, if and only if K is coordinable w.r.t A and Σ_{com} .*

Proof. For economy of notation, let P_i denote $P_{\Sigma^A, \Sigma^{S_i}}$ for $1 \leq i \leq n$.

(If) Suppose that K is coordinable w.r.t A and Σ_{com} , namely K is controllable w.r.t A and $\Sigma_c^A = \bigcup_{i=1}^n \Sigma_c^{A_i}$, and for all $1 \leq i \leq n$, K is observable w.r.t $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}$. We present a constructive proof that computes a CM set $\{S_i \mid 1 \leq i \leq n\}$ where S_i is for A_i , such that $L_m(\parallel_{i=1}^n A_i^{S_i}) = K$, $L(\parallel_{i=1}^n A_i^{S_i}) = \overline{K}$ and $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \Sigma_{com}$.

Let S be a trim automaton with $L_m(S) = K$; and for each $1 \leq i \leq n$, let $S_i = CM(S, \Sigma^{CM_i})$, where $\Sigma^{CM_i} = \Sigma^{A_i} \cup \Sigma_{com}$. Following, since K is controllable w.r.t A and Σ_c^A , it is easy to verify that the automata S_i 's meet all the necessary requirements of CM's, i.e., S_i is $\Sigma_{uc}^{A_i}$ -enabling and S_i and S_j are cooperative for $i \neq j$.

Since the event sets of the agents are pair-wised disjoint and $\Sigma_{com} \subseteq \Sigma^A$, it is obvious that $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \bigcup_{i=1}^n ((\Sigma^{A_i} \cup \Sigma_{com}) - \Sigma^{A_i}) = \Sigma_{com}$. To show that our construction works, it remains to show that $L_m(\|_{i=1}^n A_i^{S_i}) = K$ and $L(\|_{i=1}^n A_i^{S_i}) = \overline{K}$.

To begin with, note that since $S_i = CM(S, \Sigma^{CM_i})$, we have $\Sigma^{S_i} = \Sigma^{A_i} \cup \Sigma_{com}$, $L_m(S_i) = P_i(L_m(S))$ and $L(S_i) = P_i(L(S))$. It follows that $\forall 1 \leq i \leq n$, K is observable w.r.t A and P_i , and $(\forall s \in (\Sigma^A)^*)$,

$$P_i(s) \in L(S_i) \Leftrightarrow (\exists s' \in \overline{K}) P_i(s') = P_i(s) \quad (3.1)$$

$$P_i(s) \in L_m(S_i) \Leftrightarrow (\exists s' \in K) P_i(s') = P_i(s) \quad (3.2)$$

• *Proof of $L(\|_{i=1}^n A_i^{S_i}) = \overline{K}$*

- Since for $1 \leq i \leq n$, $L(S_i) = P_i(L(S))$, $P_i^{-1}(L(S_i)) = P_i^{-1}(P_i(L(S))) \supseteq L(S)$. Therefore,

$$\begin{aligned} L(\|_{i=1}^n A_i^{S_i}) &= L[\|_{i=1}^n (A_i \parallel S_i)] \\ &= L[(\|_{i=1}^n S_i) \parallel (\|_{i=1}^n A_i)] \\ &= L[(\|_{i=1}^n S_i) \parallel A] \\ &= \bigcap_{i=1}^n P_i^{-1}(L(S_i)) \cap L(A) \\ &\supseteq L(S) \cap L(A) = \overline{K}. \end{aligned}$$

- We show the other inclusion $L(\|_{i=1}^n A_i^{S_i}) \subseteq \overline{K}$ by induction on the length of strings, as follows.

* *Base:* It is obvious that $\varepsilon \in L(\|_{i=1}^n A_i^{S_i}) \cap \overline{K}$.

* *Inductive Hypothesis:* Assume that $(\forall s \in (\Sigma^A)^*)$, $|s| = m$ for some $m \geq 0$, $s \in L(\|_{i=1}^n A_i^{S_i}) \Rightarrow s \in \overline{K}$.

Now we must show that $(\forall \sigma \in \Sigma^A)$ and $(\forall s \in (\Sigma^A)^*)$, $|s| = m$, $s\sigma \in L(\|_{i=1}^n A_i^{S_i}) \Rightarrow s\sigma \in \overline{K}$. This can be done as follows.

- Without loss of generality, suppose $\sigma \in \Sigma^{A_i}$ for some $1 \leq i \leq n$.
- By Definition 3.2, $s\sigma \in L(\|_{i=1}^n A_i^{S_i}) \Rightarrow P_i(s)\sigma \in L(S_i)$.
- Since $\sigma \in \Sigma^{A_i}$, by (3.1), $P_i(s)\sigma \in L(S_i) \Rightarrow ((\exists s'\sigma \in \overline{K}) P_i(s') = P_i(s\sigma))$.

- By inductive hypothesis, $s \in L(\|_{i=1}^n A_i^{S_i}) \Rightarrow s \in \overline{K}$.
- Hence, since K is observable w.r.t A and P_i , by Condition 1) of Definition 2.2, the conditions $P_i(s) = P_i(s')$, $s \in \overline{K}$, $s\sigma \in L(A)$ and $s'\sigma \in \overline{K}$ together imply $s\sigma \in \overline{K}$.
- * Hence the inclusion $L(\|_{i=1}^n A_i^{S_i}) \subseteq \overline{K}$.
- *Proof of $L_m(\|_{i=1}^n A_i^{S_i}) = K$*
 - Since for $1 \leq i \leq n$, $L_m(S_i) = P_i(L_m(S))$, $P_i^{-1}(L_m(S_i)) = P_i^{-1}(P_i(L_m(S))) \supseteq L_m(S)$. Therefore,

$$\begin{aligned}
L_m(\|_{i=1}^n A_i^{S_i}) &= L_m[\|_{i=1}^n (A_i \parallel S_i)] \\
&= L_m[(\|_{i=1}^n S_i) \parallel (\|_{i=1}^n A_i)] \\
&= L_m[(\|_{i=1}^n S_i) \parallel A] \\
&= \bigcap_{i=1}^n P_i^{-1}(L_m(S_i)) \cap L_m(A) \\
&\supseteq L_m(S) \cap L_m(A) = K.
\end{aligned}$$

- We now show the other inclusion $L_m(\|_{i=1}^n A_i^{S_i}) \subseteq K$. Let $s \in L_m(\|_{i=1}^n A_i^{S_i})$, we show $s \in K$, as follows.
 - * By Definition 3.2, $s \in L_m(\|_{i=1}^n A_i^{S_i}) \Rightarrow [s \in L(\|_{i=1}^n A_i^{S_i}) \text{ and } s \in L_m(A) \text{ and } (\forall 1 \leq i \leq n) P_i(s) \in L_m(S_i)]$.
 - * Since $L(\|_{i=1}^n A_i^{S_i}) = \overline{K}$, $s \in L_m(\|_{i=1}^n A_i^{S_i}) \Rightarrow [s \in \overline{K} \cap L_m(A) \text{ and } (\forall 1 \leq i \leq n) P_i(s) \in L_m(S_i)]$.
 - * By (3.2), $(\forall 1 \leq i \leq n) [P_i(s) \in L_m(S_i) \Rightarrow [(\exists s_i \in K) P_i(s_i) = P_i(s)]]$.
 - * Hence, since $\forall 1 \leq i \leq n$, K is observable w.r.t A and P_i , by Condition 2) of Definition 2.2, the conditions $s \in \overline{K} \cap L_m(A)$, $P_i(s) = P_i(s_i)$ and $s_i \in K$ together imply $s \in K$.
 - * Hence the inclusion $L_m(\|_{i=1}^n A_i^{S_i}) \subseteq K$.

(Only If) Suppose that there exists a CM set $\{S_i \mid 1 \leq i \leq n\}$, where S_i is for A_i , such that $L_m(\|_{i=1}^n A_i^{S_i}) = K$, $L(\|_{i=1}^n A_i^{S_i}) = \overline{K}$ and $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \Sigma_{com}$. We show that K is coordinable w.r.t A and Σ_{com} .

By Definition 3.3, to show that K is coordinable w.r.t A and Σ_{com} , we have to show that

1. K is controllable w.r.t A and $\Sigma_c^A = \bigcup_{i=1}^n \Sigma_c^{A_i}$, and
2. K is observable w.r.t A and $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}$, $1 \leq i \leq n$.

• *Proof of Condition 1 of Coordinability*

To prove the controllability of K , let $s \in \overline{K}$ and $\sigma \in \Sigma_{uc}^{A_i}$ for some $1 \leq i \leq n$ such that $s\sigma \in L(A)$. By Definition 2.1, we have to show that $s\sigma \in \overline{K}$. This can be done as follows.

- To begin with, since $\overline{K} = L(\|_{i=1}^n A_i^{S_i})$, by Definition 3.2, $s \in \overline{K} \Rightarrow s \in L(\|_{i=1}^n A_i^{S_i}) \Rightarrow P_i(s) \in L(S_i)$.
- Next, since $s\sigma \in L(A)$ and $\sigma \in \Sigma_{uc}^{A_i}$, we have $P_{\Sigma^A, \Sigma^{A_i}}(s)\sigma \in L(A_i)$. Therefore, since CM S_i is $\Sigma_{uc}^{A_i}$ -enabling, $[P_{\Sigma^A, \Sigma^{A_i}}(s)\sigma \in L(A_i) \text{ and } \sigma \in \Sigma_{uc}^{A_i}] \Rightarrow P_i(s)\sigma \in L(S_i)$.
- Thus, by Definition 3.2, the conditions $s \in L(\|_{i=1}^n A_i^{S_i})$, $s\sigma \in L(A)$ and $P_i(s)\sigma \in L(S_i)$ together imply $s\sigma \in L(\|_{i=1}^n A_i^{S_i})$ or $s\sigma \in \overline{K}$. Hence the controllability condition.

• *Proof of Condition 2 of Coordinability*

To prove the observability of K , let $s, s' \in (\Sigma^A)^*$, $\sigma \in \Sigma^A$ such that $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s')$ for some $1 \leq i \leq n$. By Definition 2.2, we have to show that:

- (i) $[s\sigma \in \overline{K} \text{ and } s' \in \overline{K} \text{ and } s'\sigma \in L(A)] \Rightarrow s'\sigma \in \overline{K}$, and
- (ii) $[s \in K \text{ and } s' \in \overline{K} \cap L_m(A)] \Rightarrow s' \in K$.

To begin with, we first note that since $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \Sigma_{com}$, ($\forall 1 \leq i \leq n$) $\Sigma^{A_i} \cup \Sigma_{com} \supseteq \Sigma^{S_i}$. It follows that $P_{\Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^{A_i} \cup \Sigma_{com}}(s')$ implies ($\forall 1 \leq i \leq n$) $P_i(s) = P_i(s')$.

– *Proof of Condition (i) of Observability*

Suppose that $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s')$, $s\sigma \in \overline{K}$, $s' \in \overline{K}$ and $s'\sigma \in L(A)$. We show that $s'\sigma \in \overline{K}$, as follows.

- * Since $\overline{K} = L(\|_{i=1}^n A_i^{S_i})$, $[s\sigma \in \overline{K} \text{ and } s' \in \overline{K} \text{ and } s'\sigma \in L(A)] \Rightarrow [s\sigma \in L(\|_{i=1}^n A_i^{S_i}) \text{ and } s' \in L(\|_{i=1}^n A_i^{S_i}) \text{ and } s'\sigma \in L(A)]$.
- * By Definition 3.2, $[s\sigma \in L(\|_{i=1}^n A_i^{S_i}) \text{ and } \sigma \in \Sigma^{A_i}] \Rightarrow P_i(s)\sigma \in L(S_i)$.
- * Since $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s')$, $P_i(s) = P_i(s')$ and therefore, $P_i(s)\sigma \in L(S_i) \Rightarrow P_i(s')\sigma \in L(S_i)$.
- * By Definition 3.2, the conditions $\sigma \in \Sigma^{A_i}$, $s' \in L(\|_{i=1}^n A_i^{S_i})$ and $P_i(s')\sigma \in L(S_i)$ together imply $s'\sigma \in L(\|_{i=1}^n A_i^{S_i})$, or $s'\sigma \in \overline{K}$. Hence Condition i) of observability.

– *Proof of Condition (ii) of Observability*

Suppose $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s')$, $s \in K$ and $s' \in \overline{K} \cap L_m(A)$.

We show that $s' \in K$, as follows.

- * Since $L(\|_{i=1}^n A_i^{S_i}) = \overline{K}$ and $L_m(\|_{i=1}^n A_i^{S_i}) = K$, $[s \in K \text{ and } s' \in \overline{K} \cap L_m(A)] \Rightarrow [s \in L_m(\|_{i=1}^n A_i^{S_i}) \text{ and } s' \in L(\|_{i=1}^n A_i^{S_i}) \cap L_m(A)]$.
- * By Definition 3.2, $s \in L_m(\|_{i=1}^n A_i^{S_i}) \Rightarrow (\forall 1 \leq i \leq n) P_i(s) \in L_m(S_i)$.
- * Since $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s) = P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma_{com}}(s')$, $(\forall 1 \leq i \leq n) P_i(s) = P_i(s')$ and therefore, $(\forall 1 \leq i \leq n) P_i(s) \in L_m(S_i) \Rightarrow (\forall 1 \leq i \leq n) P_i(s') \in L_m(S_i)$.
- * Hence, by Definition 3.2, the conditions $s' \in L(\|_{i=1}^n A_i^{S_i})$, $s' \in L_m(A)$ and $(\forall 1 \leq i \leq n) P_i(s') \in L_m(S_i)$ together imply $s' \in L_m(\|_{i=1}^n A_i^{S_i})$ or $s' \in K$. Hence Condition ii) of observability.

□

3.3 Problem Statement and Solution Properties

A fundamental agent coordination problem may now be stated as follows.

Problem 3.1 (Fundamental Agent Coordination Problem (MCP)). *Given multiagent system $A = \|_{i=1}^{i=n} A_i$ of $n \geq 2$ interacting agents A_i and an inter-agent constraint automaton C with $\Sigma^C = \Sigma^A$, construct a set of nonblocking CM's $\{S_i \mid 1 \leq i \leq n\}$, with S_i for agent A_i , such that $L_m(\|_{i=1}^{i=n} A_i^{S_i}) = L_m(\text{Supcon}(C, A))$ and $L(\|_{i=1}^{i=n} A_i^{S_i}) = L(\text{Supcon}(C, A))$.*

In the context of MCP, $L_m(C)$ specifies the desired behavior, i.e., it embodies all the desirable event sequences that one wishes to impose on the system A . A CM set $\{S_i \mid 1 \leq i \leq n\}$ that satisfies the conditions stated in MCP is said to be minimally interventive since, using these CM's, each agent A_i does not unnecessarily disable its controllable events, unless not doing so could lead eventually to the violation of the inter-agent constraint C .

When solving MCP, it is desirable to synthesize optimal CM's, i.e., minimally interventive CM's with the following additional properties:

1) *Minimal Communication*: The cardinality of the event set to be communicated to each agent is minimal. Such a property is desirable when the underlying infrastructure has limited capability or the communication cost is high.

2) *Efficient Implementation*: Each CM S_i is of minimal state size (among all minimally interventive CM's for agent A_i satisfying the minimal communication property), and so can be efficiently implemented in terms of memory requirements.

Remark 3.1. *That the cardinality of the event set to be communicated to each agent is minimal does not necessary imply that the actual number of events that the agents communicate among themselves during coordination will be minimal. It is possible that some events in a minimal cardinality communication set are executed and communicated more frequently than the events in some larger communication set, resulting in a larger number of events communicated during coordination. However, in order for our coordination framework to be widely applicable, the underlying mechanism with which a coordinating agent selects an enabled event to execute is assumed not modeled. It follows that quantitative information such as the frequencies of event occurrences are not known a priori, and we therefore postulate that the problem of minimizing inter-agent communication can be logically addressed as that of minimizing the cardinality of the event set to be communicated to each agent.*

As will be explained, by Definition 3.3 and Theorem 3.1, the fundamental coordination problem can be solved by re-interpreting and utilizing control synthesis methods developed for SCP (Problem 2.1).

Definition 3.4 (Minimal Inter-Agent Communication Set). *Let $L \subseteq L_m(A)$. A subset Σ' of Σ^A is a minimal (cardinality) inter-agent communication set (of A for L) if*

1. L is observable w.r.t A , $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma'}$ for all $1 \leq i \leq n$.

2. $(\forall \Sigma'' \subseteq \Sigma^A)$ if L is observable w.r.t A and $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma''}$ for all $1 \leq i \leq n$ then $|\Sigma'| \leq |\Sigma''|$.

A minimal (cardinality) inter-agent communication set is denoted by $MinSysComSet(L, A)$.

By Definition 3.3 and Theorem 3.1, when the agents are coordinating to achieve conformance to an inter-agent constraint prescribed by the language L , $MinSysComSet(L, A)$ in Definition 3.4 is a set of events that must be communicated among the agents during coordination. Computing such an event set is mathematically equivalent to a variant of the minimal (cardinality) sensor-selection problem [32, 69]. The original sensor-selection problem [32] is addressed in the control context, on the premise that event observation incurs sensor installation cost. Given the language L for system A , the problem is finding a minimal cardinality event set $\Sigma' \subseteq \Sigma^A$ whose observation ensures the observability of L w.r.t A and $P_{\Sigma^A, \Sigma'}$. Formally, the statement of the sensor-selection problem [32] can be given as follows.

Problem 3.2 (Minimal sensor-selection problem [32]). *Given a DES A and an automaton H with $L_m(H) = L \subseteq L_m(A)$, find an event set $\Sigma' \subseteq \Sigma^A$ (of minimal cardinality) that satisfies the following conditions: (1) L is observable w.r.t A and $P_{\Sigma^A, \Sigma'}$, and (2) $(\forall \Sigma'' \subseteq \Sigma^A)(L \text{ is observable w.r.t } A \text{ and } P_{\Sigma^A, \Sigma''} \Rightarrow |\Sigma'| \leq |\Sigma''|)$.*

The minimal sensor-selection problem has been shown to be NP-hard [112]. Its current state-of-the-art solutions include an exponential time exact algorithm [32] and polynomial time approximate algorithms [69]. Besides not guaranteeing a minimal cardinality solution set, the latter are also only applicable under some assumptions. $MinSysComSet(L, A)$ could be computed by adapting the former [32].

Remark 3.2. *Given L and A , the original algorithm [32] considers all subsets of Σ^A and selects from them a minimal cardinality event set Σ' for which the observability of L w.r.t A and $P_{\Sigma^A, \Sigma'}$ holds. In modifying the algorithm for $MinSysComSet(L, A)$, we consider all subsets of Σ^A and select from them a minimal cardinality event set Σ' for which the observability of L w.r.t A and $P_{\Sigma^A, \Sigma^{A_i} \cup \Sigma'}$ holds for all $1 \leq i \leq n$.*

By Definition 3.4, a language $K \subseteq L_m(A)$ is always observable w.r.t A and $P_{\Sigma^A, \Sigma^{A_i} \cup MinSysComSet(K, A)}$ for all $1 \leq i \leq n$. If K is also controllable, then by Definition 3.3, it is coordinable w.r.t A and $MinSysComSet(K, A)$. It follows that, to guarantee

minimal communication among $n \geq 2$ agents coordinating to achieve K , by Theorem 3.1 and its constructive proof, CM's S_i can be computed such that $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \text{MinSysComSet}(K, A)$, and done with $\Sigma^{S_i} = \Sigma^{A_i} \cup \text{MinSysComSet}(K, A)$.

Remark 3.3. *For a coordinable $K \subseteq L_m(A)$ w.r.t A and $\text{MinSysComSet}(K, A)$, there might exist some other CM's S_j , $1 \leq j \leq n$, with $\bigcup_{i=1}^n (\Sigma^{S_i} - \Sigma^{A_i}) = \text{MinSysComSet}(K, A)$, for which Σ^{S_j} is a strict subset of $\Sigma^{A_j} \cup \text{MinSysComSet}(K, A)$. Such CM's might further reduce the communication needs for some individual sending agents. However, as explained above, CM's S_i with $\Sigma^{S_i} = \Sigma^{A_i} \cup \text{MinSysComSet}(K, A)$ for all $1 \leq i \leq n$ do guarantee minimal communication at the system level.*

Finally, from the control viewpoint, consider A_i , when self-looped at each state with events in $\Sigma^{S_i} - \Sigma^{A_i}$, as a plant to be supervised. Then each CM S_i can be thought of as a supervisor controlling the plant to achieve the controlled behavior $S_i \parallel A_i$. The problem of minimizing the state size of CM S_i for agent A_i is therefore mathematically related to the minimal supervisor problem [95, 88]. The original minimal supervisor problem [88], which is formally stated below, is proposed in the control context to address the economy of implementation in terms of memory requirements for the supervisor.

Problem 3.3 (Minimal supervisor problem [88]). *Given a DES A and a supervisor S for A with $\Sigma^S = \Sigma^A$, synthesize a minimal state size supervisor automaton S_{min} that is control equivalent to S , i.e., a supervisor automaton that satisfies the following conditions: (1) $L(S_{min}) \cap L(A) = L(S) \cap L(A)$ and $L_m(S_{min}) \cap L_m(A) = L_m(S) \cap L_m(A)$, and (2) $(\forall S')(L(S') \cap L(A) = L(S) \cap L(A) \text{ and } L_m(S') \cap L_m(A) = L_m(S) \cap L_m(A)) \Rightarrow |X^{S_{min}}| \leq |X^{S'}|$.*

The minimal supervisor problem has been shown to be NP-hard [88]. Thus, the computational complexity of an algorithm addressing Problem 3.3 is expected to be exponential. To mitigate the computational hardness of synthesizing a minimal state size supervisor S_{min} for Problem 3.3, a heuristic (polynomial time) reduction procedure called *Supreduce* [88] is proposed. The procedure *Supreduce*(S, A) synthesizes and returns a reduced state size supervisor $S_{reduced}$ which is control equivalent to the given supervisor S for DES A . Formally, the synthesized supervisor $S_{reduced}$ is an automaton with $L(S_{reduced}) \cap L(A) = L(S) \cap L(A)$, $L_m(S_{reduced}) \cap L_m(A) = L_m(S) \cap L_m(A)$ and ideally should have $|X^{S_{reduced}}| \ll |X^S|$. Numerical experimentation has shown that *Supreduce* can

often return a supervisor $Supreduce$ that has significant state size reduction for a moderate size supervisor S [88].

Thus, procedure $Supreduce$ [88] could be modified as a new procedure called $CMreduce$, in attempting to address the efficient implementation of CM's S_i , $1 \leq i \leq n$. The result is that, given S_i and A_i , $CMreduce(S_i, A_i)$ can often return a greatly state-size reduced CM automaton for agent A_i achieving the same behavior of $S_i \parallel A_i$.

Procedure $CMreduce(S_i, A_i)$

begin

- 1 Add every event in $(\Sigma^{S_i} - \Sigma^{A_i})$ as a self-loop transition at every state of A_i . Let the resulting automaton be A'_i ;
- 2 Return $Supreduce(S_i, A'_i)$;

end

To highlight, procedure $CMreduce$ is different from the original procedure $Supreduce$ [88] in that it is for $S_i \parallel A_i$ with $\Sigma^{S_i} \supseteq \Sigma^{A_i}$, instead of that with $\Sigma^{S_i} = \Sigma^{A_i}$.

3.4 Coordination Module Synthesis

An important implication of the preceding discussions is that the discrete-event techniques of control and sensor selection can be adapted and applied to synthesize CM's for distributed agents.

Algorithm 1 details the planning steps for coordination design of $n \geq 2$ agents. Given n agents A_i , $1 \leq i \leq n$, and inter-agent constraint C , it constructs a CM set $\{S_i \mid 1 \leq i \leq n\}$, where S_i is for A_i , which is nonblocking and minimally interventive, i.e., $L_m(\|_{i=1}^n A_i^{S_i})$ is equal to the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$. Moreover, the event set to be communicated among the agents is minimal among all the CM sets that achieve the same coordinated behavior. Furthermore, the state size of each CM S_i is often significantly reduced by the $CMreduce$ procedure. Whereas the two properties of minimal intervention and communication are guaranteed, the CM's returned by the algorithm have a relatively small state size that is not necessarily minimal; hence these CM's can only be said to be *near-optimal*. The correctness of the proposed algorithm is immediate based on the previous discussions. The planning steps presented may utilize TCT [102], a freely available software for DES synthesis.

Algorithm 1: Coordination Module Synthesis

Input: $n \geq 2$ agents $A_1, A_2, \dots, A_n$ with $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $1 \leq i \neq j \leq n$, and constraint C where $\Sigma^C = \bigcup_{i=1}^n \Sigma^{A_i}$

Output: A near-optimal nonblocking CM set $\{S_i \mid 1 \leq i \leq n\}$, where S_i is for A_i , such that $L_m(\|_{i=1}^n A_i^{S_i}) \subseteq L_m(\|_{i=1}^n A_i) \cap L_m(C)$

begin

- Step 1** Compute automaton A and controllable set Σ_c^A
 $A \leftarrow \|_{i=1}^n A_i; \Sigma_c^A \leftarrow \bigcup_{i=1}^n \Sigma_c^{A_i};$
- Step 2** Compute a nonblocking supervisor S
 $S \leftarrow \text{Supcon}(C, A);$
- Step 3** Compute coordination event sets $\Sigma^{CM_i}, 1 \leq i \leq n$
 $\Sigma^{CM_i} \leftarrow \Sigma^{A_i} \cup \text{MinSysComSet}(L_m(S), A);$
- Step 4** Compute CM's $S_i, 1 \leq i \leq n$
 $S_i \leftarrow \text{CM}(S, \Sigma^{CM_i});$
- Step 5** Reduce state size of the CM's $S_i, 1 \leq i \leq n$
 $S_i \leftarrow \text{CMreduce}(S_i, A_i);$
- Step 6** Return CM set $\{S_i \mid 1 \leq i \leq n\};$

end

Note that every event in $\text{MinSysComSet}(L_m(S), A)$ will need to be communicated by one agent and received by at least one other agent A_j . However, in cases where an event in $\text{MinSysComSet}(L_m(S), A)$ appears as a self-loop transition at every state of agent A_j 's CM, it need not be communicated to agent A_j . As a matter of practical interest, every such event should be removed from agent A_j 's CM to obtain a new CM S_j as in Remark 3.3 for actual implementation.

In the worst case, Algorithm 1 has exponential time complexity because all the following procedures suffer from exponential complexity: the synchronous product of A , the projection of S onto the event subset $\Sigma^{CM_i} \subseteq \Sigma^S$ in Procedure CM , and the construction of the minimal inter-agent communication set in Procedure MinSysComSet . This exponential complexity is not surprising since in our MCP, the multiagent system $A = \|_{i=1}^n A_i$, the inter-agent constraint C and each CM S_i , when self-looped at each state with events in $\Sigma^A - \Sigma^{S_i}$, share the same mathematical models as an n -component modular DES plant, a control specification and a supervisor controlling the plant, respectively; and the problem of synthesizing multiple supervisors for such a modular DES subject to a specification has been shown to be NP-hard [30].

When necessary, heuristic algorithms or data structures should be designed for the synthesis of large systems. It is envisioned that existing work on complexity reduction in discrete-event systems could be utilized to address this design issue. For example, binary decision diagrams and state tree structures [48] have proven to be efficient encoding techniques for reducing the complexity of synthesizing the product A and the automaton S . To mitigate the complexity of Procedure CM , each event subset Σ^{CM_i} could be enlarged such that $P_{\Sigma^S, \Sigma^{CM_i}}$ becomes a natural observer of $L_m(S)$ [99], before applying Procedure CM to compute CM S_i from S and Σ^{CM_i} . It has been shown that if $P_{\Sigma^S, \Sigma^{CM_i}}$ is a natural observer of $L_m(S)$, then computing a projection of S on Σ^{CM_i} only requires polynomial time w.r.t the state size of S [99]. However, one should note that enlarging Σ^{CM_i} would increase the number of events to be communicated among coordinating agents. To mitigate the complexity of Procedure *MinSysComSet*, the polynomial time approximate algorithms for sensor selection [69] may be adapted and applied to compute a small but not necessarily minimal inter-agent coordination event set. However, this approach also compromises the minimal inter-agent communication property guaranteed by Algorithm 1. Thus, one might have to trade-off between the computational complexity of synthesizing CM's and the number of events to be communicated among coordinating agents. A deeper investigation of this issue is an interesting topic that is beyond the scope of this thesis.

3.5 Illustrative Example

We now present an example to illustrate the use of Algorithm 1 and the design improvement it offers over the original approach [80, 82]. The system under study is a manufacturing system consisting of two agents A_1 and A_2 connected in tandem and separated by a one-slot buffer BUF [Figure 3.2(a)]. The working environment considered requires autonomous and “active” agents capable of coordinating between themselves to achieve the goal of satisfying a given inter-agent constraint. These agents would need to be individually equipped with CM's or “built-in” strategies for achieving the goal. In a conventional manufacturing environment, we might be able to treat each agent as a “passive” process to be controlled, and develop external supervisors - with each over the system of passive agents - using a decentralized control framework [46, 76]. However, as will be discussed in the next section, treating agents as passive processes to be controlled is not always possible or the most appropriate, and may have some limitations in distributed system modeling.

In all subsequent figures for the example, an automaton G is represented by an edge-labeled directed graph with a state represented by a node, and a transition $\delta^G(\sigma, x) = x'$ by a directed edge from state x to x' labeled with the symbol $\sigma \in \Sigma^G$ of an event whose occurrence it represents. The σ -labeled edge is drawn as a directed line with an optional tick ($\circ - | \gg \circ$) if the event $\sigma \in \Sigma^G$ is controllable. The initial state is represented by a node with an entering arrow, and a marker state by a node drawn as a double concentric circle.

Initially, the buffer is empty. A_1 is a producer that produces workpieces continually, one piece at a time, and places them into the buffer BUF. In order to do so, A_1 has to produce a workpiece, go to the buffer place, and place the workpiece into the buffer. According to its local plan, A_1 can either produce a workpiece first or go to the buffer place first [Figure 3.2(b)]. A_2 is a consumer that consumes workpieces continually, one piece at a time, from the buffer BUF. To do so, A_2 first needs to go to the buffer place. Upon arriving, it then takes a workpiece from BUF and returns to its initial state for a new consumption cycle [Figure 3.2(c)]. In this example, we arbitrarily fix $\Sigma_c^A = \{1produced, 1arrived, 2arrived, 2taken\}$ and $\Sigma_{uc}^A = \{1placed\}$.

The inter-agent constraint is stated as follows: *Producer agent must produce a workpiece first before going to the buffer place. Moreover, the buffer must never overflow or underflow.*

Intuitively, the constraint requires *1produced* to be executed first, and *1produced* and *1arrived* must be executed alternately thereafter; and similarly for *1placed* and *2taken*. Clearly, the constraint imposes more restrictions on the plan of A_1 than that of A_2 . Hence, one would expect CM S_1 to be more complex than CM S_2 . The textual description of the constraint can be formulated by automaton C [Figure 3.2(d)]. How this can be done is simply taken for granted here. Such an automaton may be more easily prescribed with the aid of a high-level specification translator [79].

Nonblocking CM pair (S_1, S_2) with $L_m(A_1^{S_1} \parallel A_2^{S_2}) \subseteq L_m(A) \cap L_m(C)$ may now be synthesized using Algorithm 1. Automaton $A = A_1 \parallel A_2$ representing the system is first computed in Step 1. In Step 2, automaton $S = SupCon(C, A)$, whose marked language $L_m(S)$ is the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$, is constructed (see Figure 3.3). After Step 3, we obtain the event sets: $MinSysComSet(L_m(S), A) = \{1placed, 2taken\}$, $\Sigma^{CM_1} = \{1produced, 1arrived, 1placed, 2taken\}$ and $\Sigma^{CM_2} = \{2arrived, 2taken, 1placed\}$; using which, CM's $S_i = CM(S, \Sigma^{CM_i})$, $i \in \{1, 2\}$, are computed in Step 4 (see Figure 3.4). Finally, the state size of these CM's is

Figure 3.2: Coordination planning for a manufacturing system.

reduced by procedure *CMreduce* in Step 5, and the synthesis CM solution is returned in Step 6 (see Figure 3.5). To elaborate, using these CM's means: A_1 must inform A_2 whenever it places a workpiece into the buffer, and A_2 reciprocates in turn whenever it takes a workpiece from the buffer. For this example, the CM's S_1 and S_2 returned by Algorithm 1 are verified to be optimal, i.e., each is of minimal state size among all the minimally interventive CM's that entail minimal communication between the agents.

In [80, 82], a different approach to the synthesis of CM's is presented. There, a supervisor S' is first computed by applying procedure *Supreduce* to automaton S [see Figure 3.6(a)]. Following, CM's S'_1 and S'_2 , with S'_i for A_i , are computed by removing from S' all its strictly self-loop events that are not defined in agent models A_1 and A_2 , respectively [see Figures 3.6(b) and 3.6(c)]. The supervisor S' returned might be minimally reactive as is the case in this example, i.e., S' has the least number of states and the largest number of strictly self-loop events with respect to inter-agent constraint C and system A [82].

Figure 3.3: Automaton S with $L_m(S)$ - the supremal controllable sublanguage of $L_m(A) \cap L_m(C)$.

(a) CM S_1 for producer agent A_1

(b) CM S_2 for consumer agent A_2

Figure 3.4: Coordination modules for the manufacturing agents before applying the state-size reduction procedure $CMreduce$.

With this method, a minimally reactive S' is the best outcome though this is not guaranteed in general. Synthesizing from such an S' produces rather state-efficient CM's S'_i with a reduced number of events (in $\Sigma^A - \Sigma_{loop}^{S'}$) to be communicated between the agents.

However, as this example clearly illustrates, Algorithm 1 offers an improvement in coordination design synthesis over the original method [80, 82], even with the latter producing a minimally reactive S' for CM design. Firstly, the system communication set $\{1placed, 2taken\}$ between the agents (using CM's S_1 and S_2) is a subset of that when using CM's S'_1 and S'_2 , which is $\{1produced, 1arrived, 1placed, 2taken\}$, thanks to the *MinSysComSet* procedure. Secondly, CM S_2 has a smaller state size than CM S'_2 , and thus has a lower memory requirement. Achieving this higher level of efficiency in implementation is the result of individually reducing the CM's state size using procedure *CMreduce* in Step 5 of Algorithm 1, instead of reducing them 'together' using procedure *Supreduce*, as in the original method. In essence, the individual state-size reduction

Figure 3.5: Coordination modules for the manufacturing agents.

of CM's exploits the fact that an inter-agent constraint may impose different restrictions on different agent models, and thus the less restricted agent should expect a simpler CM.

3.6 Discrete-Event Control versus Coordination

The proof of the necessary condition of Theorem 3.1 has utilized established mathematical results for the existence of a supervisor in discrete-event control theory [45]. However, multiagent coordination and supervisory control are conceptually different problems. As explained in [80, 82], the latter entails enablement or disablement of events in a DES by external supervisors, while the former entails interaction and communication among agents in the DES through their local CM's. Unlike event-feedback control, coordinating interaction necessarily involves synchronous message passing, whereby the agents exchange messages in the form of executed events to maintain conformance to a given inter-agent constraint. Below, we further distinguish our coordination design problem from related problems in the control literature.

Lin and Wonham [46] and Rudie and Wonham [76] study different versions of the decentralized discrete-event control problem which shares the same mathematical foundation with but is different from our problem of multiagent coordination, as will be explained later in this section. The decentralized control problem is to synthesize multiple supervisors, each with different observation and control capabilities, that jointly control a DES to achieve conformance to a given global control specification. The observation and control capabilities of individual (decentralized) supervisors are predetermined respectively as

Figure 3.6: Non-optimal coordination modules for the manufacturing agents.

subsets of observable and controllable events of the DES, and can be different for each supervisor.

In one problem version, the global control specification is specified in terms of a set of local specifications, each specification is a language over the union set of a supervisor's observable and controllable event subsets [46]. Sufficient conditions for the existence of a set of supervisors as well as synthesis algorithms for this problem have been proposed [46].

In another more general problem version, the global control specification is given as a language over the controlled DES's event set, which is not necessarily decomposable into a set of local specifications for individual supervisors as in the earlier discussed [46]. Lin [44] studies and provides existence conditions and synthesis algorithms for this problem under the assumption that the controllable event subset of a decentralized supervisor is the same as its observable event subset. However, this assumption is rather restrictive and may be not applicable for practical control systems.

Without any restrictive assumption, Rudie and Wonham [76] study the general problem

version, and establish an important condition specifying that a set of decentralized supervisors exists if and only if the global specification language satisfies the two properties of controllability and co-observability [76]. It has been shown that the latter property reduces to the observability of the specification language with respect to each of the supervisors' observable event subset, if the supervisors' controllable event subsets are pair-wise disjoint [12, 74]. Thus, for the case of two supervisors controlling a DES $A = A_1 \parallel A_2$ with $\Sigma^{A_1} \cap \Sigma^{A_2} = \emptyset$, and with each supervisor i observing what is equivalent to the event set $\Sigma^{A_i} \cup \Sigma_{com}$ and controlling event set $\Sigma_c^{A_i}$, a controllable and co-observable language is mathematically equivalent to a coordinable language given in Definition 3.3.

When the co-observability of the global specification language is not satisfied, decentralized supervisors may communicate observable events among themselves to establish co-observability [98, 2, 94, 70]. Among the first research efforts, Wong and Schuppen [98] present a necessary and sufficient condition for solving the decentralized control problem with communication by formulating a refinement relation between observation and control. Barrett and Lafortune [2], following the same approach, further develop a general decentralized control with communication framework. Schuppen [94] hypothesizes that the general problem of synthesizing communicating decentralized supervisors is likely to be undecidable. Rohloff and Schuppen [70] discuss the relationship between the minimal communication and minimal sensor selection problems, and propose several heuristic algorithms for approximating the minimal communication event sets between decentralized supervisors.

Like [70], we also study the problem of minimal communication, but in a new discrete-event multiagent coordination setting. In our multiagent coordination framework, the sets of observable events constitute the system communication set Σ_{com} , which is a union of local event subsets of the sending agents that need to be determined for each receiving agent. Unlike decentralized control and supervisory control in general, the observable events for a receiving agent (or events to be communicated to the agent when they occur) are not pre-determined but computed with the aim of minimizing communication, and therefore can be different for a different inter-agent constraint. Applying our synthesis algorithm for two agents, the local event subset $\Sigma_{com} \cap \Sigma^{A_i}$ of the sending agent A_i can be determined and minimized for the receiving agent A_j , thereby minimizing communication between the coordinating agents. Minimizing communication is a problem of significant importance for application domains in which communication bandwidth is a scarce resource.

Significantly, what distinguishes our multiagent coordination problem from the decentralized control problem (with or without communication) reviewed above is that, the former arises in application domains such as robotic agents where distributed agent autonomy is a key consideration, whereas the latter arises in domains where it is not and the DEP's are not active agent models but (can be treated as) passive system components to be controlled. From this perspective, the supervisory control framework presents a "supervisor-subordinate" architecture where a plant (which can consist of multiple components) is monitored and controlled by a single supervisor or a set of decentralized supervisors. In contrast, our multiagent framework presents a "peer-to-peer" architecture where distributed agents operate independently but cooperatively to achieve conformance to some system level constraint. Unlike a component to be controlled in the supervisory control framework, which acts passively under the direction of external supervisors, an agent in our framework acts actively following its own defined plan and its "built-in" coordination strategy (represented by its CM). In any case, when addressing a decentralized control problem, one might need to decide on the number of decentralized supervisors and decompose the event set into controllable and observable subsets for these supervisors which are non-issues in multiagent coordination planning. In the decentralized control framework, it is as yet unclear how such event set decomposition and the reduction of (decentralized) supervisor state can be effectively solved.

Recently, Rudie, Lin and Lafortune [73, 72] consider a problem where one control agent (or supervisor) communicates with another agent for information so as to distinguish the states of its automaton, or recognize the set of transitions pre-specified as essential, for control decision-making or diagnosis. Since communication may be costly, a strategy to reduce communication between agents is developed. Like theirs, we also seek to reduce communication between agents, but consider a different problem where the agents coordinate by interacting and communicating for information so as to cooperatively satisfy an inter-agent constraint.

Finally, in a recent independent and emerging work [8], a different problem called supervisor localization is presented. For a DES A consisting of $n \geq 2$ interacting local components A_i , $1 \leq i \leq n$, with pair-wise disjoint event sets, the localization problem focuses on decomposing (or localizing) a global supervisor S of A into a set of local supervisors $\{S_i \mid 1 \leq i \leq n\}$, with S_i controlling A_i , while preserving the control behavior of S over A . Although communication minimization is not explicitly considered in the

supervisor localization solution, the problem can be shown to be equivalent to our multiagent coordination problem [80, 82], i.e., Problem 2. However, unlike the supervisor localization framework [8], our multiagent framework clearly distinguishes the related but different concepts of control and coordination by the Cartesian and synchronous product operators [80], respectively. In distinguishing control and coordination, the mathematical equivalence between coordination of localized supervisors and of agents is established and discussed in [82, Corollary 1]. More importantly, in our opinion, this conceptual difference brings into sharper focus the essence of our new coordination problem, namely, designing built-in CM's - not supervisors - for autonomous agents, and leads us to not prejudging that the only means of CM synthesis is by first constructing supervisors for a multiagent system. Finally, we note that the intent of our framework is to naturally model active agents coordinating through their CM's, whereas that of the framework [8] is apparently to model passive agents being controlled by their interacting localized supervisors

3.7 Chapter Summary

This chapter has formulated the fundamental discrete-event multiagent agent coordination problem, and presented a novel approach for discrete-event coordination synthesis. Discussion with related work has highlighted the importance of our problem and solution approach.

A fundamental insight unearthed in this chapter is that minimal sensor selection and minimal agent communication actually share the same algorithmic solution although they are conceptually different domains. Importantly, this insight, together with earlier work [80, 82] establishing the mathematical connection between supervisory control and multiagent coordination, leads us to adapting and applying the DES algorithmic foundation to develop a new algorithm (Algorithm 1) to synthesize CM's for coordinating agents, without reinventing the wheel. As guaranteed by an established theoretical result (Theorem 3.1), the synthesis algorithm (Algorithm 1) computes and returns near-optimal CM's which are minimally interventive and entail minimal communication between the agents. Moreover, they are state-reduced individually for each agent, leading to design improvement over the original method [80, 82] as demonstrated by an example.

Chapter 4

NETWORKED AGENT COORDINATION PROBLEM - PART I: CONFLICT RESOLUTION

4.1 Chapter Overview

In Chapter 3, we formulated and addressed the fundamental coordination problem for multiple agents. In this and the next chapter, we generalize the problem to a distributed-constraint network of agents. In a multiagent system, multiple inter-agent constraints can be distributed such that each constraint is pre-specified for a subgroup of agents. These agent subgroups can be overlapping, meaning that an agent can be coordinating on different inter-agent constraints with different agents in the system. As we will elaborate more later on, conflicts can arise between agents in overlapping groups, and if these agents cannot properly resolve them, they can fail to reach their design goals. This presents a challenging design problem of networked coordination which is commonly encountered in large-scale distributed systems.

The networked coordination problem under study can be described as follows. Given an agent group $\{A_i \mid 1 \leq i \leq n\}$ of size $n \geq 2$, and an inter-agent constraint set $\{C_j \mid 1 \leq j \leq m\}$ of size $m \geq 1$, where each constraint restricts a subgroup of the agents, synthesize, for each agent A_i , a set of CM's so that the agents coordinating as a whole can satisfy every constraint C_j . In our discrete-event coordination framework where agents

and constraints are modeled by automata, a constraint C_j is said to restrict an agent A_i , or equivalently, C_j is a relevant constraint for A_i , if the event sets of C_j and A_i are not disjoint. Given this problem, one immediate solution approach is to compute a global constraint $C = C_1 \parallel C_2 \parallel \dots \parallel C_m$, and simply apply the synthesis algorithm presented in Chapter 3 to construct a CM for every agent A_i to satisfy C . However, recall from Chapter 3 that the synthesis algorithm requires the computation of the synchronous product of all agent and specification models, i.e., $A_1 \parallel \dots \parallel A_n \parallel C$. This simple approach, therefore, suffers from a major drawback: The associated computational complexity could be prohibitively expensive due to the infamous state explosion problem [21] arising from the computation of automaton synchronous product. For a relatively large number of agents and constraints, we would therefore need to alleviate the computation of the synchronous product, and cannot directly apply the synthesis algorithm developed in Chapter 3.

In this and the next chapter, we propose a compositional approach to the synthesis of CM's for the networked coordination problem. Our approach consists of two main steps as follows. In the first step, we construct for each agent a set of $\parallel$ -connected local CM's, one for each of the agent's relevant constraints. The advantage of constructing local CM's is that we can avoid having to compute the product of all agent and constraint models, thereby mitigating the problem of state explosion. However, as an example in this chapter will show, local CM's do not generally constitute a correct solution for the networked coordination problem. The reason is that using local CM's to coordinate, agents may conflict with one another, causing them to get into a situation of blocking where some agents cannot reach their design goals. To fix this, in the second step, we generate a conflict resolution plan, and follow this plan to design additional deconflicting CM's for individual agents and interpose them between their local CM's. Deconflicting CM's are used to resolve the conflicts between agents due to their using local CM's synthesized for different inter-agent constraints. Local and deconflicting CM's can then be shown to constitute a correct solution to the networked coordination problem. Importantly, our deconflicting algorithms are based on projections of agent models onto some pre-determined event sets, and this often enables designing deconflicting CM's in an efficient manner.

We begin the formal development of our compositional synthesis approach by proposing a new specification formalism called the Distributed Constraint Specification Network (DCSN) to organize a system of agents and their distributed constraints. A DCSN describes the networking constraint relationships among agents by associating every agent

with its relevant constraints, thereby specifying “who needs to coordinate with whom over what constraints”. Graphically depicted, a DCSN is a specification tool which is designer comprehensible in that it can clearly show the networking of constraints among agents for a networked coordination problem. For an intuitive example, the reader might want to skip ahead to Figure 4.3 for a graphically depicted DCSN. Importantly, as will be shown in the next chapter, a DCSN can be converted to a relational model indicating potential conflicts among agents in different subnets that must be resolved. This relational model is the basis for generating a plan for the conflict resolution.

Within a DCSN, a constraint and the subgroup of agents that it is relevant for form a basic constraint subnet, and the union of $r \geq 2$ basic subnets constitutes a r -constraint subnet. Our compositional synthesis approach for a given DCSN can then be described as follows.

- **Step 1 Basic Subnet Synthesis**

Synthesize for every agent a set of $\parallel$ -connected local CM's, one for each of the agent's relevant constraints. This step is performed by applying the algorithm developed in Chapter 3 to every basic constraint subnet of the DCSN, i.e., every subnet containing one inter-agent constraint.

- **Step 2 Subnet Composition**

- **Step 2.1 Conflict Resolution Plan Generation**

Generate a conflict resolution plan for the DCSN. This plan is a sequence of subnet composition operations. Each operation entails designing deconflicting CM's for the agents of the subnets concerned, so as to ensure nonblockingness, and hence correctness, when the subnets are composed together.

- **Step 2.2 Conflict Resolution Plan Execution**

Compose subnets with conflict resolution by following a precedence order of subnet composition operations in the plan generated in Step 2.1. This is to completely deconflict the local CM's synthesized in Step 1 to ensure nonblockingness of the whole DCSN.

An algorithm for Step 1 has already been proposed in Chapter 3. In the main development of this chapter, we shall explain and illustrate the problem of conflicts among agents

in overlapping subnets. The conflict resolution approach that follows lays an algorithmic foundation for Step 2.2. The theory and algorithms for Step 2.1, of generating a conflict resolution plan to correctly and completely deconflict local CM's as the subnets are successively composed together to form the DCSN, are the subject of the next chapter.

The rest of this chapter is organized as follows. Section 4.2 reviews the essential properties of language projections that can be used to efficiently design deconflicting CM's for coordinating agents. Section 4.3 introduces and defines the DCSN as a formalism for organizing a system of agents and their distributed constraints. Throughout this chapter, we will use a simple manufacturing line example to illustrate our theoretical development. The example system is described in Section 4.4. Next, Section 4.5 formalizes our networked coordination problem. In Section 4.6, we apply the synthesis algorithm developed in Chapter 3 to construct local CM's for every basic constraint subnet of a given DCSN. In Section 4.7, we discuss the issue of conflicts among constraint subnets, and propose our conflict resolution approach. Finally, Section 4.8 concludes this chapter.

4.2 Preliminaries

In this chapter, we will use small letters such as n, m, k, r to denote integers. For an integer $n \geq 1$, the symbol I_n denotes the index set $\{1, 2, \dots, n\}$.

4.2.1 Natural Projection

Let Σ be an event set and $\Sigma_o \subseteq \Sigma$. Recall from Chapter 2 that P_{Σ, Σ_o} denotes the (natural) projection from Σ^* to Σ_o^* , which is a map defined as follows:

$$P_{\Sigma, \Sigma_o}(\varepsilon) = \varepsilon,$$

and $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)$,

$$P_{\Sigma, \Sigma_o}(s\sigma) = \begin{cases} P_{\Sigma, \Sigma_o}(s)\sigma, & \text{if } \sigma \in \Sigma_o; \\ P_{\Sigma, \Sigma_o}(s), & \text{otherwise.} \end{cases}$$

In words, P_{Σ, Σ_o} erases from a string $s \in \Sigma^*$ every event that is not in Σ_o . The definition of a projection can be extended to languages as follows: For $L \subseteq \Sigma^*$, $P_{\Sigma, \Sigma_o}(L) =$

$\{P_{\Sigma, \Sigma_o}(s) \in \Sigma_o^* \mid s \in L\}$, and for $L_o \in \Sigma_o^*$, $P_{\Sigma, \Sigma_o}^{-1}(L_o) = \{s \in \Sigma^* \mid P_{\Sigma, \Sigma_o}(s) \in L_o\}$.

For notational simplicity, whenever $\Sigma_o \not\subseteq \Sigma$, we shall use P_{Σ, Σ_o} instead of the notationally correct $P_{\Sigma, \Sigma \cap \Sigma_o}$ to denote the natural projection from Σ^* to $(\Sigma \cap \Sigma_o)^*$.

Lemma 4.1 ([101]). *If $L \subseteq \Sigma^*$ is a prefix-closed language, then so is $P_{\Sigma, \Sigma_o}(L)$. And, for an arbitrary language $L \in \Sigma^*$, $\overline{P_{\Sigma, \Sigma_o}(L)} = P_{\Sigma, \Sigma_o}(\overline{L})$.*

Let A be an automaton and $\Sigma_o \subseteq \Sigma^A$. Then, by Lemma 4.1, $P_{\Sigma, \Sigma_o}(L(A))$ is a prefix-closed language, and if A is trim, i.e., $L(A) = \overline{L_m(A)}$, then $P_{\Sigma, \Sigma_o}(L(A)) = \overline{P_{\Sigma, \Sigma_o}(L_m(A))}$. With a slight abuse of notation, we will use $P_{\Sigma^A, \Sigma_o}(A)$ to denote the projected image of A onto Σ_o , which is an automaton that generates $P_{\Sigma, \Sigma_o}(L_m(A))$ and $P_{\Sigma, \Sigma_o}(L(A))$ as the marked and prefix-closed languages, respectively.

The following property of distributivity of projections over the language synchronous product [87] will be used later in this chapter.

Proposition 4.1 ([87]). *Given $n \geq 2$ languages $L_i \in \Sigma_i^*$, $i \in I_n$. Let $\Sigma = \bigcup_{i \in I_n} \Sigma_i$, $\Sigma_o \subseteq \Sigma$ and $\Sigma_s = \bigcup_{j, k \in I_n, j \neq k} \Sigma_j \cap \Sigma_k$. Then if $\Sigma_s \subseteq \Sigma_o$, the following holds:*

$$P_{\Sigma, \Sigma_o}(\parallel_{i \in I_n} L_i) = \parallel_{i \in I_n} P_{\Sigma_i, \Sigma_o}(L_i).$$

In Proposition 4.1, Σ_s is the set of shared events of all event sets Σ_i 's, $i \in I_n$. The proposition simply states that if $\Sigma_s \subseteq \Sigma_o$, the projection of the product language $L_1 \parallel L_2 \parallel \dots \parallel L_n$ onto Σ_o is equal to the product of the projections of the component languages. Proposition 4.1 offers an economical way of computing the projection of the product of a set of languages, without having to compute the product language itself [87, 26, 101].

Proposition 4.2 is the automata-theoretic version of Proposition 4.1.

Proposition 4.2. *Given $n \geq 2$ automata A_i 's, $i \in I_n$. Let $A = A_1 \parallel A_2 \dots \parallel A_n$, $\Sigma_o \subseteq \Sigma^A$ and $\Sigma_s = \bigcup_{j, k \in I_n, j \neq k} \Sigma^{A_j} \cap \Sigma^{A_k}$. Then if $\Sigma_s \subseteq \Sigma_o$, the following holds:*

$$P_{\Sigma^A, \Sigma_o}(A) \equiv \parallel_{i \in I_n} P_{\Sigma^{A_i}, \Sigma_o}(A_i).$$

Proof. Immediate from Proposition 4.1 and Lemma 4.1. □

4.2.2 Natural Observer

For an automaton A , $P_{\Sigma^A, \Sigma_o}(A)$ is often used as an abstracted model of A . Intuitively, if A is interpreted as a system model and $\Sigma_o \subseteq \Sigma^A$ as a set of events whose occurrence is of interest to some outsider observing A , then the abstraction $P_{\Sigma^A, \Sigma_o}(A)$ hides every event that corresponds to operational details of A which are of no interest to the observer, and hence presents the observer with a relevant high-level model of A .

Abstractions have been extensively used in the literature to deal with the state explosion problem in control synthesis [27, 33, 78], to design high-level specifications in hierarchical control [100, 17], and to mitigate the complexity of nonconflict test among supervisors in modular control [56, 57]. In all of these developments, the abstracted models are in some sense consistent with, but simpler than the original models. A key property for abstraction consistency often used in the literature is natural observer, which is defined as follows.

Definition 4.1 ([99]). *Let $L \subseteq \Sigma^*$ be a language and $\Sigma_o \subseteq \Sigma$. Then the projection P_{Σ, Σ_o} is said to be a L -observer if the following condition is satisfied: $\forall t \in P_{\Sigma, \Sigma_o}(L), s \in \bar{L}$, if $P_{\Sigma, \Sigma_o}(s)$ is a prefix of t then $\exists u \in \Sigma^*$ such that $su \in L$ and $P_{\Sigma, \Sigma_o}(su) = t$.*

In words, Definition 4.1 asserts that whenever $P_{\Sigma, \Sigma_o}(s)$ can be extended to a string in $P_{\Sigma, \Sigma_o}(L)$ by catenating to it a string $u_o \in \Sigma_o^*$, the underlying string s can also be extended to a string in L by catenating to it a string $u \in \Sigma^*$ with $P_{\Sigma, \Sigma_o}(u) = u_o$. This definition of a natural observer adapts the same concept from [100], where it is originally defined for general causal reporter maps. Informally, if P_{Σ, Σ_o} is a $L_m(A)$ -observer for some automaton A , Definition 4.1 says that what is expected in the abstract model $P_{\Sigma, \Sigma_o}(A)$ is also realizable in the original model A .

Importantly, a projection which has the property of a natural observer also provides abstraction simplicity in terms of automaton state size reduction in general, and this renders the efficient computation of abstract models in practice [99]. It is known that in the worst case, the complexity of computing $P_{\Sigma^A, \Sigma_o}(A)$ is exponential in the state size of automaton A , and the state size of $P_{\Sigma^A, \Sigma_o}(A)$ is of exponential order with respect to that of A [99]. However, if P_{Σ^A, Σ_o} is a $L_m(A)$ -observer, $P_{\Sigma^A, \Sigma_o}(A)$ can be computed in polynomial time and has a state size not greater than that of A [99]. If P_{Σ, Σ_o} is not a $L_m(A)$ -observer, it can be refined to become one by enlarging Σ_o accordingly; this can be done using a polynomial time algorithm [28].

Proposition 4.3 ([56]). *Given $n \geq 2$ languages $L_i \in \Sigma_i^*$, $i \in I_n$. Let $\Sigma = \bigcup_{i \in I_n} \Sigma_i$, $\Sigma_o \subseteq \Sigma$ and $\Sigma_s = \bigcup_{j,k \in I_n, j \neq k} \Sigma_j \cap \Sigma_k$. Then if $\Sigma_s \subseteq \Sigma_o$ and $(\forall i \in I_n) P_{\Sigma_i, \Sigma_o}$ is a L_i -observer, P_{Σ, Σ_o} is a $(\parallel_{i \in I_n} L_i)$ -observer.*

By Proposition 4.1, under the stated conditions of Proposition 4.3, the projection of the product language $\parallel_{i \in I_n} L_i$ onto Σ_o is equal to the product of the projections of the component languages L_i 's. Thus, Proposition 4.3 implies that the product of natural L_i -observers is a natural observer of the product language $\parallel_{i \in I_n} L_i$.

4.2.3 Output Control Consistency

Another property of natural projections that is often used in tandem with the observer property is output control consistency (OCC).

Definition 4.2 ([28]). *Let $L \subseteq \Sigma^*$ be a prefix-closed language and Σ_o and Σ_{uc} be two event subsets of Σ , interpreted respectively as observable and uncontrollable event sets. Then the projection P_{Σ, Σ_o} is said to be output control consistent (OCC) for L if $\forall s \in L$ of the form $s = s' \sigma_1 \dots \sigma_k$, $k \geq 1$, where s' is either ε or terminates with an event in Σ_o , $\sigma_1, \dots, \sigma_{k-1} \in \Sigma - \Sigma_o$ and $\sigma_k \in \Sigma_o$, the following holds: If $\sigma_k \in \Sigma_{uc}$ then $(\forall 1 \leq i \leq k-1) \sigma_i \in \Sigma_{uc}$.*

By Definition 4.2, along every $s \in L$, between an observable but uncontrollable event and its nearest “upstream” observable event (or otherwise the empty string) is a path of uncontrollable events. Thus, if L is interpreted as (the behavior of) an underlying system model and $P_{\Sigma, \Sigma_o}(L)$ as (the behavior of) the system abstracted model, then, that P_{Σ, Σ_o} is OCC for L characterizes the fact that every uncontrollable event in the abstracted model can never be disabled and hence prevented from occurring by disabling controllable events in the underlying model, whereas every controllable event in the abstracted model can be. The abstracted model output $P_{\Sigma, \Sigma_o}(L)$ is in this sense “control consistent” with the original model L . Computationally, if P_{Σ, Σ_o} is not OCC for L , the projection can be refined to be so in polynomial time [113], by adding to Σ_o all the nearest “upstream” observable events of every event $\sigma \in \Sigma_o \cap \Sigma_{uc}$.

Interestingly, projections with both OCC and observer properties have been shown to play an important role in the development of efficient algorithms for synthesizing modular

discrete-event supervisors [99, 56, 28, 104, 19]. A detailed review of this modular control problem is presented in the Appendix.

In concluding this section, we state a useful theoretical result proved in [28]. It is abstracted in a more generic form herewith to support the subsequent development of an efficient method of constructing deconflicting CM's for coordinating agents.

Proposition 4.4 ([28]). *Given $n \geq 2$ automata G_i , $i \in I_n$, with $\Sigma^{G_i} \cap \Sigma^{G_j} = \emptyset$ for $i \neq j$, and two automata E and F with $\Sigma^E, \Sigma^F \subseteq \bigcup_{i \in I_n} \Sigma^{G_i}$. Let $G_E = \bigparallel_{\Sigma^{G_i} \cap \Sigma^E \neq \emptyset} G_i$ and $G_F = \bigparallel_{\Sigma^{G_i} \cap \Sigma^F \neq \emptyset} G_i$, $S_E = \text{Supcon}(E, G_E)$, $S_F = \text{Supcon}(F, G_F)$ and $\Sigma_o \supseteq \Sigma^E \cap \Sigma^F$. Then, if P_{Σ^E, Σ_o} is a $L_m(S_E)$ -observer and P_{Σ^F, Σ_o} is a $L_m(S_F)$ -observer, and for all $i \in I_n$, $P_{\Sigma^{G_i}, \Sigma_o}$ are OCC for $L(G_i)$, we have*

$$\text{Supcon}(G_o, P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F)) \parallel S_E \parallel S_F \parallel G \equiv \text{Supcon}(E \parallel F, G),$$

where G_o is a one-state automaton that generates and marks Σ_o^* and $G = G_1 \parallel G_2 \parallel \dots \parallel G_n$.

4.3 Distributed Constraint Specification Network (DCSN)

Let $\mathcal{A} = \{A_i \mid i \in I_n\}$ be a set of $n \geq 2$ nonblocking automata modeling a set of n discrete-event agents, with $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $i \neq j$. The event set Σ^{A_i} (of agent A_i) is partitioned into the controllable event set $\Sigma_c^{A_i}$ and the uncontrollable event set $\Sigma_{uc}^{A_i}$.

Let $A = A_1 \parallel A_2 \parallel \dots \parallel A_n$ model a system of n agents in $\mathcal{A}$ freely interacting, with $\Sigma_c^A = \bigcup_{i \in I_n} \Sigma_c^{A_i}$ and $\Sigma_{uc}^A = \bigcup_{i \in I_n} \Sigma_{uc}^{A_i}$. Let $J \subseteq I_n$. Then, an inter-agent constraint for a group of agents $\mathcal{A}_J = \{A_j \mid j \in J\}$ can be prescribed by an automaton C_J such that $(\forall j \in J) \Sigma^{C_J} \cap \Sigma^{A_j} \neq \emptyset$. The language $L_m(C_J)$ is interpreted as the set of desirable event sequences that one wishes to impose on the group of agents $\mathcal{A}_J$. In other words, constraint C_J specifies that the agents in $\mathcal{A}_J$ must coordinate among themselves so that none of those event sequences in $L_m(A_J) - L_m(A_J \parallel C_J)$ will ever be generated during their interaction, where $A_J = \bigparallel_{j \in J} A_j$.

In distributed multiagent systems, there are often multiple distributed inter-agent constraints, each restricting a group of interacting agents. To specify the relevance relationships

of distributed constraints among these agents, we define a formalism called the distributed constraint specification network (DCSN). The DCSN allows a human designer to organize and interconnect the agents and their distributed constraints in a networking structure that, in our opinion, comprehensibly shows “who needs to coordinate with whom over what constraints”.

Definition 4.3. Let $n \geq 2$, $m \geq 1$. A distributed constraint specification network (DCSN) $\mathcal{N}$ is a tuple $(\mathcal{A}, \mathcal{C})$, where $\mathcal{A} = \{A_i \mid i \in I_n\}$ is an agent set of size n and $\mathcal{C} = \{C_{J_k}^k \mid k \in I_m, J_k \subseteq I_n\}$ is an inter-agent constraint set of size m , such that $(\forall C_{J_k}^k \in \mathcal{C})(\forall i \in J_k) \Sigma^{A_i} \cap \Sigma^{C_{J_k}^k} \neq \emptyset$.

Each $C_{J_k}^k \in \mathcal{C}$ in a DCSN $\mathcal{N}$ is said to be a relevant constraint for agents in the group $\mathcal{A}_{J_k} = \{A_i \mid i \in J_k\}$. Without loss of generality, assume henceforth that $\bigcup_{k \in I_m} J_k = I_n$, i.e., every agent in $\mathcal{A}$ is in $\mathcal{A}_{J_k}$ for some k , and so every agent needs to coordinate. Then a DCSN can be redefined as $\mathcal{N} = \{(J_k, C_{J_k}^k) \mid k \in I_m, J_k \subseteq I_n\}$.

Definition 4.4. An element $\mathcal{N}_1^k = (J_k, C_{J_k}^k)$ of $\mathcal{N}$ is called a basic subnet of $\mathcal{N}$; and a non-empty $\mathcal{N}_r^{\mathcal{S}_r} \subseteq \mathcal{N}$ consisting of $r = |\mathcal{S}_r| \geq 1$ basic subnets is called a r -constraint subnet of $\mathcal{N}$ with constraint subset $\{C_{J_k}^k \mid k \in \mathcal{S}_r\}$. Where the constraint subset is arbitrary, a r -constraint subnet is simply denoted by $\mathcal{N}_r$.

By Definition 4.4, a subnet of a DCSN is also a DCSN.

Intuitively, a DCSN is a formalism that represents interconnections among agents and constraints, associating every agent with its relevant inter-agent constraints. Under the interconnections, an inter-agent constraint induces a group of agents that it is relevant for. It is then clear that the agents in the agent group need to coordinate to satisfy the constraint.

A DCSN can be graphically represented by an undirected hyper-graph with agents represented by rectangular nodes, and each constraint relevant for an agent group by an oval hyper-edge with arcs connecting it to all the agents in the group. Through its graphical representation which is intuitively clear and easy to understand, a DCSN is designer comprehensible for modeling the inter-agent constraint relationships among agents, as the example in the next section will demonstrate.

Figure 4.1: A manufacturing transfer line system.

4.4 A Manufacturing Transfer Line Example

Throughout this chapter, we shall use a simple manufacturing transfer line example [Figure 4.1(a)] to illustrate our theoretical development. The system under study consists of three agents A_1 , A_2 and A_3 [Figures 4.1(b)–4.1(d)], and four constraints $E_{\{1,2\}}^1$, $E_{\{1,2\}}^2$, $B_{\{1,3\}}^3$ and $B_{\{2,3\}}^4$ [Figures 4.2(a)–4.2(d)], organized into a DCSN (Figure 4.3).

The system works as follows. A_1 and A_2 are producer agents that continually follow a production plan: Acquire manufacturing equipment E_1 and E_2 in either order, produce a workpiece, return the equipment to their initial location, move to the buffers' location, place the finished workpiece into the respective buffer, and finally return to the initial state for a new production cycle. A_3 is a delivery agent that continually takes a work piece from either buffer B_1 or B_2 , processes, and delivers it to customers. We arbitrarily fix $\Sigma_{uc}^A = \{1produce, 1return, 1place, 2produce, 2return, 2place, 3process, 3deliver\}$.

The four constraints $E_{\{1,2\}}^1$, $E_{\{1,2\}}^2$, $B_{\{1,3\}}^3$ and $B_{\{2,3\}}^4$ are formulated to respectively ensure mutual exclusion of equipment use, and no overflow or underflow of buffers.

The DCSN is composed of four basic subnets $\mathcal{N}_1^1 = (\{1, 2\}, E_{\{1,2\}}^1)$, $\mathcal{N}_1^2 = (\{1, 2\}, E_{\{1,2\}}^2)$, $\mathcal{N}_1^3 = (\{1, 3\}, B_{\{1,3\}}^3)$ and $\mathcal{N}_1^4 = (\{2, 3\}, B_{\{2,3\}}^4)$. When depicted graphically, a nice feature of DCSN is that the constraint inter-connections between agents are explicitly shown for comprehensibility of design. For instance, in Figure 4.3, it is clear that A_1 would need to

Figure 4.2: Constraints for a manufacturing transfer line system.

Figure 4.3: A DCSN for a manufacturing transfer line system.

coordinate with A_2 for $E^1_{\{1,2\}}$ and $E^2_{\{1,2\}}$, and with A_3 for $B^3_{\{1,3\}}$.

4.5 Networked Coordination Problem Statement

Given a DCSN $\mathcal{N} = (\mathcal{A}, \mathcal{C})$, we propose to equip each agent in $\mathcal{A}$ with a set of CM's so that they can coordinate to satisfy every constraint in $\mathcal{C}$. Let $\mathcal{CM}_i = \{S_i^h \mid i \in I_{r_i}\}$ denote the set of $r_i \geq 1$ CM's that are equipped to agent A_i . In the subsequence development of this and the next chapter, we will show how the set $\mathcal{CM}_i$'s can be synthesized.

Through their CM's, the agents coordinate as follows. Following the execution of a string $s \in L(A)$, A_i updates the state of every CM $S_i^h \in \mathcal{CM}_i$ to $x_i^h = \delta^{S_i^h}(P_{\Sigma^A, \Sigma^{S_i^h}}(s), x_0^{S_i^h})$. A_i then enables (allows to execute) only events $\sigma_i \in \Sigma^{A_i}$ that is defined at every current state of its CM's. In other words, A_i enables an event $\sigma_i \in \Sigma^{A_i}$ only if $(\forall S_i^h \in \mathcal{CM}_i) \delta^{S_i^h}(x_i^h)!$. The result is that the system behavior is restricted to a sublanguage of $L(A)$.

For each CM S_i^h , $\Sigma^{S_i^h}$ represents the set of events that A_i needs to observe in order to correctly update the state of S_i^h when interacting with the other agents. The event set $(\Sigma^{S_i^h} - \Sigma^{A_i})$, which cannot be observed locally by A_i , must be communicated to A_i by

other agents. In other words, an $A_j, j \neq i$, must inform to A_i whenever it executes an event in $\bigcup_{S_i^h \in \mathcal{CM}_i} [(\Sigma^{S_i^h} - \Sigma^{A_i}) \cap \Sigma^{A_j}]$.

Let $\mathcal{CM} = \{\mathcal{CM}_i \mid i \in I_n\}$ and $CM_i = \parallel_{S_i^h \in \mathcal{CM}_i} S_i^h$, for $i \in I_n$. From the foregoing discussions, the system of n agents in $\mathcal{A}$ coordinating through their respective CM's can be represented by $A^{CM} = \parallel_{i \in I_n} (A_i \parallel CM_i)$. The CM's are then said to be nonblocking if every string generated during the agents' interaction can be completed to a marked string, i.e., $\overline{L_m(A^{CM})} = L(A^{CM})$.

We can now formally state our discrete-event networked coordination problem.

Problem 4.1. *Given a DCSN $\mathcal{N} = (\mathcal{A}, \mathcal{C})$ of n agents and m inter-agent constraints, let $A = \parallel_{i \in I_n} A_i$ and $C = \parallel_{k \in I_m} C_{J_k}^k$, where $A_i \in \mathcal{A}$ and $C_{J_k}^k \in \mathcal{C}$. Synthesize a set $\mathcal{CM} = \{\mathcal{CM}_i \mid i \in I_n\}$, where $\mathcal{CM}_i$ is a set of CM's for agent A_i , such that $A^{CM} \equiv \text{Supcon}(C, A)$, i.e., the resulting coordinated system is nonblocking and satisfies every constraint in $\mathcal{C}$ in a minimally restrictive manner.*

$L_m(C)$ specifies the desired behavior, embodying all the event sequences that one wishes to impose on the system A . A set $\mathcal{CM}$ of CM's is then said to satisfy (every constraint in) $\mathcal{C}$ if $L_m(A^{CM}) \subseteq L_m(C)$. It can be easily shown that $L_m(A^{CM})$ is controllable with respect to A and Σ_{uc}^A . Thus, for a set $\mathcal{CM}$ of CM's satisfying $\mathcal{C}$, $L_m(A^{CM}) \subseteq L_m(\text{Supcon}(C, A))$. A CM set $\mathcal{CM}$ is then said to satisfy $\mathcal{C}$ in a *minimally interventive* manner if $A^{CM} \equiv \text{Supcon}(C, A)$, implying that using such CM's, each agent A_i would not unnecessarily disable its controllable events, unless not doing so could lead eventually to the violation of some inter-agent constraint in $\mathcal{C}$.

4.6 Basic Subnet Synthesis

In addressing Problem 4.1, we will follow the two steps of our compositional synthesis approach outlined in the introduction (Section 4.1). In this section, filling in Step 1, we demonstrate how the synthesis algorithm developed in Chapter 3 can be used to construct local CM's for coordinating agents.

Given a DCSN $\mathcal{N} = (\mathcal{A}, \mathcal{C})$ of n agents and m inter-agent constraints, we consider the problem of synthesizing CM's for some basic subnet $\mathcal{N}_1^k = (J_k, C_{J_k}^k)$ of $\mathcal{N}$, $k \in I_m$.

To fix notation, let $A_{J_k} = \parallel_{i \in J_k} A_i$ and $SUP^k = Supcon(C_{J_k}^k, A_{J_k})$. We are interested in synthesizing, for each agent A_i in the subnet, a CM S_i^k such that $\parallel_{i \in J_k} (A_i \parallel S_i^k) \equiv SUP^k$.

As discussed earlier, the synthesis algorithm developed in Chapter 3 can be used to synthesize local CM's for the agents in a basic subnet. For this purpose, the pseudo-code of the synthesis algorithm is modified as a procedure called *CMBasicSubnet* for basic subnet synthesis, as shown in Figure 4.4.

Figure 4.4: Procedure *CMBasicSubnet* for synthesizing a CM S_i^k for every agent A_i in $\mathcal{N}_1^k = (J_k, C_{J_k}^k)$ such that $(A_1 \parallel S_1^k) \parallel \dots \parallel (A_n \parallel S_n^k) \equiv SUP^k$.

Recall that $MinSysComSet(L_m(SUP^k), A_{J_k})$ computes and returns a minimal cardinality communication event set that the agents A_i 's in the subnets must communicate among themselves, CM constructs for each agent $A_i, i \in J_k$, a CM S_i^k from SUP^k and $\Sigma_{mincom,i}^k$, and $CMreduce$ is a CM reduced procedure, which can often return a greatly state-size reduced CM automaton for agent A_i , achieving the same behavior of $A_i \parallel S_i^k$.

Example 4.1. To illustrate the use of Procedure *CMBasicSubnet*, we apply it to the manufacturing transfer line example and synthesize CM's for agents A_1 and A_2 to cooperatively satisfy $E_{\{1,2\}}^1$.

By Step 1 of *CMBasicSubnet*, we first compute $SUP^1 = Supcon(E_{\{1,2\}}^1, A_1 \parallel A_2)$, which has 40 states and 82 transitions. Next, by Step 2, the minimal communication sets for A_1 and A_2 are computed: $\Sigma_{mincom,1}^1 = \{2take1, 2return\}$ and $\Sigma_{mincom,2}^1 = \{1take1, 1return\}$. Following Step 3, CM's $S_i^1, i \in \{1, 2\}$, are computed by applying Procedure *CM* on SUP^1 and $\Sigma^{A_i} \cup \Sigma_{mincom,i}^1$. Each of these CM's has 11 states and 19 transitions. Finally, in Step 4, $CMreduce$ is applied to reduce the state size of S_1^1 and S_2^1 ,

Figure 4.5: CM's of A_1 and A_2 for $E_{\{1,2\}}^1$.

arriving at the state-reduced CM's, each with 2 states and 11 transitions (see Figure 4.5). To elaborate, using these CM's means: A_1 must inform A_2 whenever it takes or returns the equipment E_1 , and A_2 reciprocates in turn.

Similarly, the CM's S_1^2 and S_2^2 synthesized using *CMBasicSubnet* for agents A_1 and A_2 to cooperatively satisfy $E_{\{1,2\}}^2$ are given in Figure 4.6.

Figure 4.6: CM's of A_1 and A_2 for $E_{\{1,2\}}^2$.

4.7 Composing Subnets with Conflict Resolution

4.7.1 Composing Basic Subnets

We now consider how two basic subnets can be composed together to obtain a solution for the resultant two-constraint subnet.

Given $\mathcal{N}_2^{\{h,k\}} = \{(J_h, C_{J_h}^h), (J_k, C_{J_k}^k)\}$, let $SUP^{\{h,k\}} = Supcon(C_{J_h}^h \parallel C_{J_k}^k, A_{J_h} \parallel A_{J_k})$. We are interested in synthesizing, for each agent A_i , a set of CM's $\mathcal{CM}_i$ such that $\parallel_{i \in J_h \cup J_k} (A_i \parallel \mathcal{CM}_i) \equiv SUP^{\{h,k\}}$. Without loss of generality, we assume $J_h \cap J_k \neq \emptyset$. Otherwise, the two basic subnets contain no common agents and would only need to be synthesized individually.

Filling in Step 2.2 of our compositional synthesis approach outlined in Section 4.1, we present in this section an algorithm to compose the CM solutions of $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ to obtain a solution for $\mathcal{N}_2^{\{h,k\}}$. As discussed earlier, an alternative synthesis approach is to reorganize $\mathcal{N}_2^{\{h,k\}}$ into a new subnet consisting of one constraint $C_{J_h}^h \parallel C_{J_k}^k$ for the agent group $\{A_i \mid i \in J_h \cup J_k\}$. The solution for this reorganized basic subnet can then be obtained by applying *CMBasicSubnet*. However, as already discussed in Section 4.1, this approach has a major drawback: It suffers from exponential complexity of computing the product of all agents $\{A_i \mid i \in J_h \cup J_k\}$ and constraints $C_{J_h}^h$ and $C_{J_k}^k$. For a large number of agents, this computation may become prohibitively expensive.

Our compositional approach entails designing deconflicting CM's for the agents concerned to resolve any conflict between $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. The need for additional deconflicting CM's will be clear from the following example.

Example 4.2. For $\mathcal{N}_2^{\{1,2\}} = \{\mathcal{N}_1^1, \mathcal{N}_1^2\}$, we apply *CMBasicSubnet* to compute CM's of agents A_1 and A_2 for $\mathcal{N}_1^1 = (\{1, 2\}, E_{\{1,2\}}^1)$ and $\mathcal{N}_1^2 = (\{1, 2\}, E_{\{1,2\}}^2)$. The CM's S_1^1 and S_2^1 for $\mathcal{N}_1^1$, and S_1^2 and S_2^2 for $\mathcal{N}_1^2$, are shown in Figures 4.5 and 4.6. However, using only these CM's does not guarantee that A_1 and A_2 will interact correctly for the subnet $\mathcal{N}_2^{\{1,2\}}$. In fact, the system of A_1 and A_2 interacting using these CM's contains blocking states. For instance, the event sequence $1take1 - 2take2$, which is allowed to be executed by the CM's, leads to the blocking situation of each agent holding one equipment and waiting forever to acquire the equipment held by the other agent.

Thus, the local CM's individually constructed for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ do not generally constitute a correct solution for $\mathcal{N}_2^{\{h,k\}}$. The reason is that in general, $SUP^k \parallel SUP^h \neq$

$SUP^{\{h,k\}}$; and whenever this happens, the system of agents coordinating using only their CM's constructed for the individual basic subnets will contain blocking states. This fact is formalized in Definition 4.5 and Proposition 4.5 below.

Definition 4.5. *Two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are said to be (synchronously) nonconflicting if $SUP^h \parallel SUP^k$ is nonblocking. Otherwise, they are said to be conflicting.*

Proposition 4.5. *If $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting then the local CM's synthesized for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ constitute a correct CM solution for $\mathcal{N}_2^{\{h,k\}}$. Otherwise, the system of agents coordinating using only the local CM's constructed for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ will contain blocking states.*

Proof. Given four automata G_1, G_2, E and F , the following theoretical result has been established in [19]: If $Supcon(E, G_1) \parallel Supcon(F, G_2)$ is nonblocking then $Supcon(E, G_1) \parallel Supcon(F, G_2) \equiv Supcon(E \parallel F, G_1 \parallel G_2)$.

Therefore, since $SUP^h = Supcon(C_{J_h}^h, A_{J_h})$ and $SUP^k = Supcon(C_{J_k}^k, A_{J_k})$, if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, namely $SUP^h \parallel SUP^k$ is nonblocking (by Definition 4.5), then

$$SUP^h \parallel SUP^k \equiv SUP^{\{h,k\}}.$$

For each agent A_i in $\mathcal{N}_1^h$, let S_i^h be its local CM synthesized by applying *CMBasicSubnet* for $\mathcal{N}_1^h$. Then,

$$\parallel_{i \in J_h} (A_i \parallel S_i^h) \equiv SUP^h.$$

Similarly, for each agent A_j in $\mathcal{N}_1^k$, let S_j^k be its local CM synthesized by applying *CMBasicSubnet* for $\mathcal{N}_1^k$, then

$$\parallel_{j \in J_k} (A_j \parallel S_j^k) \equiv SUP^k.$$

Therefore,

$$[\parallel_{i \in J_h} (A_i \parallel S_i^h)] \parallel [\parallel_{j \in J_k} (A_j \parallel S_j^k)] \equiv SUP^h \parallel SUP^k.$$

It follows that if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting then

$$\left[\parallel_{i \in J_h} (A_i \parallel S_i^h) \right] \parallel \left[\parallel_{j \in J_k} (A_j \parallel S_j^k) \right] \equiv SUP^{\{h,k\}}.$$

In other words, if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, using the local CM's synthesized for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$, the agents in these two basic subnets can coordinate to achieve $SUP^{\{h,k\}}$. Thus, the local CM's synthesized for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ constitute a correct CM solution for $\mathcal{N}_2^{\{h,k\}}$.

Otherwise, if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are conflicting, $SUP^h \parallel SUP^k$ is a blocking automaton, namely, the system of agents coordinating using only the local CM's constructed for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ will contain blocking states.

Hence the proposition. $\square$

By Proposition 4.5, whenever $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, to obtain a CM solution for $\mathcal{N}_2^{\{h,k\}}$, we only need to synthesize the two basic subnets individually, i.e., without having to design additional deconflicting CM's. This motivates the development of a procedure for testing the nonconflict of $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. By Definition 4.5, the simplest way of doing so is to directly compute $SUP^h \parallel SUP^k$ and check whether or not it is a nonblocking automaton. However, this approach is computationally inefficient since it can be shown to have the same complexity order as that of computing the product of all agents and constraints.

In what follows, Lemma 4.2 leads us to a more efficient approach to testing the nonconflict of $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$.

Lemma 4.2. *Let $\Sigma_{CR}^{\{h,k\}} \supseteq \bigcup_{i \in J_k \cap J_h} \Sigma^{A_i}$ and define P_{CR}^h and P_{CR}^k as projections from $\bigcup_{i \in J_h} \Sigma^{A_i}$ and $\bigcup_{i \in J_k} \Sigma^{A_i}$ to $\Sigma_{CR}^{\{h,k\}}$, respectively. Then, if P_{CR}^h is a $L_m(SUP^h)$ -observer and P_{CR}^k is a $L_m(SUP^k)$ -observer, two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting if and only if $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$ is a nonblocking automaton.*

Proof. We have $SUP^h = \text{Supcon}(C_{J_h}^h, A_{J_h})$, $SUP^k = \text{Supcon}(C_{J_k}^k, A_{J_k})$ and $\Sigma_{CR}^{\{h,k\}} \supseteq \bigcup_{i \in J_k \cap J_h} \Sigma^{A_i}$. Suppose P_{CR}^h is a $L_m(SUP^h)$ -observer and P_{CR}^k is a $L_m(SUP^k)$ -observer. Then abstracting a theoretical result proved in [56], it follows that $SUP^h \parallel SUP^k$ is nonblocking if and only if $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$ is nonblocking.

In other words, $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting if and only if $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$ is a nonblocking automaton. $\square$

Thus, under the stated sufficiency conditions in Lemma 4.2, testing the nonconflict of $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ can be reduced to checking whether or not $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$ is nonblocking. This way, we only need to first compute $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$ instead of $SUP^h \parallel SUP^k$, which results in a computationally cheaper nonconflict test for two reasons. The first is that such automata $P_{CR}^h(SUP^h)$ and $P_{CR}^k(SUP^k)$ can be individually computed in polynomial time [99], and the second is that their state sizes are often smaller than those of SUP^h and SUP^k , respectively.

Nevertheless, if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are conflicting (due to blocking), we need to design additional deconflicting CM's for the agents concerned to resolve the conflicts between $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. Definition 4.6 formalizes the notion of deconflicting CM's.

Definition 4.6. A set of CM's $\{S_i^{\{h,k\}} \mid i \in J_k \cup J_h\}$, where $S_i^{\{h,k\}}$ is for A_i , is said to be a set of deconflicting CM's for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ if
$$\left[\parallel_{i \in J_h \cup J_k} (A_i \parallel S_i^{\{h,k\}}) \right] \parallel SUP^h \parallel SUP^k \equiv SUP^{\{h,k\}}.$$

Together with the local CM's synthesized for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$, deconflicting CM's will constitute a correct solution for $\mathcal{N}_2^{\{h,k\}}$. Essentially, deconflicting CM's remove blocking states from $SUP^h \parallel SUP^k$ when used by the agents of subnet $\mathcal{N}_2^{\{h,k\}}$.

In designing deconflicting CM's for coordinating agents, our approach is to first synthesize an automaton as the basis for conflict resolution between two basic subnets, and then “localize” it to every agent as the agent's deconflicting CM if the agent shares some events with the conflict resolution (automaton). By the foregoing notation, Definition 4.7 formalizes the solution basis of our conflict resolution approach.

Definition 4.7. An automaton $CR^{\{h,k\}}$ is said to be a conflict resolution for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ if
$$[CR^{\{h,k\}} \parallel SUP^h \parallel SUP^k] \equiv SUP^{\{h,k\}}.$$

It can be shown that a conflict resolution for any two basic subnets always exists. Indeed, $CR^{\{h,k\}}$ can be simply computed as $Supcon(G, SUP_{J_h}^h \parallel SUP_{J_k}^k)$, where G is a one-state automaton that generates and marks $(\Sigma^{A_{J_h}} \cup \Sigma^{A_{J_k}})^*$. However, similar to the problem of testing the nonconflict of two basic subnets discussed previously, computing $CR^{\{h,k\}}$ as $Supcon(G, SUP_{J_h}^h \parallel SUP_{J_k}^k)$ has the same order of complexity as that of $\parallel_{i \in J_h \cup J_k} A_i$, which is inefficient.

In what follows, we present an efficient approach for computing a conflicting resolution for two basic subnets (Lemma 4.3), and based on which, propose a conflict resolution algorithm (Procedure *DeconflictBasicSubnet*, Figure 4.7).

Lemma 4.3. *Let $\Sigma_{CR}^{\{h,k\}} \supseteq \bigcup_{i \in J_h \cap J_k} \Sigma^{A_i}$ and define P_{CR}^h and P_{CR}^k as projections from $\bigcup_{i \in J_h} \Sigma^{A_i}$ and $\bigcup_{i \in J_k} \Sigma^{A_i}$ to $\Sigma_{CR}^{\{h,k\}}$, respectively. Then, if P_{CR}^h is a $L_m(SUP^h)$ -observer, P_{CR}^k is a $L_m(SUP^k)$ -observer, and $\forall i \in J_h \cup J_k$, $P_{\Sigma^{A_i}, \Sigma_{CR}^{\{h,k\}}}$ is OCC for $L(A_i)$, then $CR^{\{h,k\}} = Supcon[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)]$ is a conflict resolution for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$, where G is a one-state automaton that generates $(\Sigma_{CR}^{\{h,k\}})^*$ as both the prefix-closed and marked languages.*

Proof. By Proposition 4.4, since the event sets of the agents in $\mathcal{N}_2^{\{h,k\}}$ are pair-wise disjoint, $SUP^h = Supcon(C_{J_h}^h, A_{J_h})$, $SUP^k = Supcon(C_{J_k}^k, A_{J_k})$ and $\Sigma_{CR}^{\{h,k\}} \supseteq \bigcup_{i \in J_h \cap J_k} \Sigma^{A_i}$, it follows that if P_{CR}^h is a $L_m(SUP^h)$ -observer, P_{CR}^k is a $L_m(SUP^k)$ -observer and $(\forall i \in J_h \cup J_k) P_{\Sigma^{A_i}, \Sigma_{CR}^{\{h,k\}}}$ is OCC for $L(A_i)$, then

$$Supcon[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)] \parallel SUP^h \parallel SUP^k \equiv SUP^{\{h,k\}}$$

namely, $CR^{\{h,k\}} = Supcon[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)]$ is a conflict resolution for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. $\square$

Thus, $CR^{\{h,k\}}$ can be computed as $Supcon[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)]$ if all the conditions stated in Lemma 4.3 are satisfied. Importantly, following this approach to compute a conflict resolution, instead of $SUP^h \parallel SUP^k$, we only need to compute the product $P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)$. Since P_{CR}^i , $i \in \{h, k\}$, is a $L_m(SUP^i)$ -observer, the state size of $P_{CR}^i(SUP^i)$ is known to be often smaller than that of SUP^i .

By Lemma 4.3, a conflict resolution for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ can be computed as follows:

- (i) Initially, let $\Sigma_{J_h \cap J_k} = \bigcup_{i \in J_h \cap J_k} \Sigma^{A_i}$.
- (ii) Enlarge $\Sigma_{J_h \cap J_k}$ to $\Sigma_{CR}^{\{h,k\}}$ so that all the stated conditions in Lemma 4.3 are satisfied.

- (iii) Construct G as a one-state automaton with its only state being both an initial state and a marker state, and with every event in $\Sigma_{CR}^{\{h,k\}}$ self-looped at that state. Thus, G generates $(\Sigma_{CR}^{\{h,k\}})^*$ which is both its prefix-closed and marked languages.
- (iv) Compute $CR^{\{h,k\}} = \text{Supcon}[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)]$.

Note that the smaller the cardinality of the set $\Sigma_{CR}^{\{h,k\}}$ returned by Step (ii) is, the more economical the computation would be for Step (iv). The problem of finding a minimal cardinality event set $\Sigma_{CR}^{\{h,k\}}$ satisfying every condition in Lemma 4.3 has proven to be NP-hard [99]. However, a polynomial time algorithm exists to synthesize such an event set $\Sigma_{CR}^{\{h,k\}}$ of reasonably small size [26].

From the foregoing discussion, Procedure *DeconflictBasicSubnet* (Figure 4.7) is developed to design deconflicting CM's for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. It first checks if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting by applying Lemma 4.2 (Step 1). If they are, then no deconflicting CM is needed. Otherwise, Lemma 4.3 is applied to compute a conflict resolution $CR^{\{h,k\}}$ for the two subnets (Step 2). Next, in Step 3, the procedure determines whether or not an agent A_i needs to take part in resolving the conflict between the subnets, i.e., if $\Sigma^{CR^{\{h,k\}}} \cap \Sigma^{A_i} \neq \emptyset$. If so, it computes for A_i a deconflicting CM $S_i^{\{h,k\}}$. Note that such a deconflicting CM could simply be taken as $CR^{\{h,k\}}$. However, to achieve economy of implementation, it uses *CMreduce* to obtain a reduced CM $S_i^{\{h,k\}} = \text{CMreduce}(CR^{\{h,k\}}, A_i)$. In the worst case, $\Sigma_{CR}^{\{h,k\}} = \bigcup_{i \in J_h \cup J_k} \Sigma^{A_i}$ and *DeconflictBasicSubnet* has to compute the synchronous product of all agents and constraints in the two subnets. It therefore has exponential complexity. However, *DeconflictBasicSubnet* is often efficient in practice since $\Sigma_{CR}^{\{h,k\}}$ is often a strict subset of $\bigcup_{i \in J_h \cup J_k} \Sigma^{A_i}$.

Procedure *DeconflictBasicSubnet* ($\mathcal{N}_1^h, \mathcal{N}_1^k$)

Output: A deconflicting CM $S_i^{\{h,k\}}$ for agent A_i to resolve the conflict between $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$

begin

Step 1: Check if $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting:

- **Step 1a** Let $\Sigma_{CR}^{\{h,k\}} = \bigcup_{i \in J_h \cap J_k} \Sigma^{A_i}$;
- **Step 1b** Enlarge $\Sigma_{CR}^{\{h,k\}}$ so that P_{CR}^h becomes a $L_m(SUP_{J_h}^h)$ -observer and P_{CR}^k becomes a $L_m(SUP_{J_k}^k)$ -observer, where P_{CR}^h and P_{CR}^k are projections from $\bigcup_{i \in J_h} \Sigma^{A_i}$ and $\bigcup_{i \in J_k} \Sigma^{A_i}$ to $\Sigma_{CR}^{\{h,k\}}$, respectively;
- **Step 1c** If $\overline{L_m(P_{CR}^h(SUP^h))} \parallel \overline{L_m(P_{CR}^k(SUP^k))} = L(P_{CR}^h(SUP^h)) \parallel L(P_{CR}^k(SUP^k))$, i.e., $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, no deconflicting CM is needed. Otherwise, go to Step 2 to design deconflicting CM's for the agents;

Step 2: Compute a conflict resolution $CR^{\{h,k\}}$ for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$:

- **Step 2a** Enlarge $\Sigma_{CR}^{\{h,k\}}$ so that P_{CR}^h is a $L_m(SUP_{J_h}^h)$ -observer, P_{CR}^k is a $L_m(SUP_{J_k}^k)$ -observer, and $\forall i \in J_h \cup J_k$, $P_{\Sigma^{A_i}, \Sigma_{CR}^{\{h,k\}}}$ is OCC for $L(A_i)$;
- **Step 2b** Construct G as a one state automaton with its only state being both an initial state and a marker state, and with every event in $\Sigma_{CR}^{\{h,k\}}$ self-looped at that state;
- **Step 2c** Compute $CR^{\{h,k\}} = Supcon[G, P_{CR}^h(SUP^h) \parallel P_{CR}^k(SUP^k)]$;

Step 3: For each agent A_i in the subnet $\mathcal{N}_2^{\{h,k\}}$, if $\Sigma^{CR^{\{h,k\}}} \cap \Sigma^{A_i} \neq \emptyset$, compute for A_i a deconflicting CM $S_i^{\{h,k\}} = CMreduce(CR^{\{h,k\}}, A_i)$;

end

Figure 4.7: Procedure *DeconflictBasicSubnet* for resolving conflict between two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$.

Lemma 4.4. *For $i \in J_h \cup J_k$, let $S_i^{\{h,k\}}$ be the deconflicting CM computed for agent A_i in Step 3 of *DeconflictBasicSubnet*, or trivially a one-state automaton that generates and marks $(\Sigma^{A_i})^*$ if no deconflicting CM is needed for A_i , either because $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting or because $\Sigma^{CR^{\{h,k\}}} \cap \Sigma^{A_i} = \emptyset$. Then, $\parallel_{i \in J_h \cup J_k} (A_i \parallel S_i^{\{h,k\}}) \equiv CR^{\{h,k\}}$.*

Proof. If the two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, the lemma is trivially true. Otherwise, by Step 3 of *DeconflictBasicSubnet*, for every agent A_i with $\Sigma^{CR^{\{h,k\}}} \cap \Sigma^{A_i} \neq \emptyset$, we have $S_i^{\{h,k\}} = CMreduce(CR^{\{h,k\}}, A_i)$. Recall from Chapter 3 that *CMreduce* is a procedure that, given $CR^{\{h,k\}}$ and A_i , often returns a greatly state-size reduced CM automaton for agent A_i achieving the same behavior of $A_i \parallel CR^{\{h,k\}}$. It follows that

$$\parallel_{\Sigma^{CR^{\{h,k\}}} \cap \Sigma^{A_i} \neq \emptyset} (A_i \parallel S_i^{\{h,k\}}) \equiv CR^{\{h,k\}}.$$

For other agents that do not share events with $CR^{\{h,k\}}$, essentially no deconflicting CM is needed. Therefore,

$$\parallel_{i \in J_h \cup J_k} (A_i \parallel S_i^{\{h,k\}}) \equiv CR^{\{h,k\}}.$$

Hence the lemma. $\square$

Theorem 4.1 formally summarizes how we can compose (the solution CM's of) two basic subnets $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ to form (a CM solution set for) the two-constraint subnet $\mathcal{N}_2^{\{h,k\}}$.

Theorem 4.1. *For $i \in J_h \cup J_k$, let $\mathcal{CM}_i$ be the CM set for agent A_i computed as follows: (i) $\mathcal{CM}_i$ includes every CM computed for A_i when applying *CMBasicSubnet* for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$, and (ii) $\mathcal{CM}_i$ includes every deconflicting CM computed for A_i when applying *DeconflictBasicSubnet* to resolve the conflict that exists between $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$. Then*

$$\parallel_{i \in J_h \cup J_k} (A_i \parallel CM_i) \equiv SUP^{\{h,k\}}, \text{ where } CM_i \text{ is a synchronous product of all CM's in } \mathcal{CM}_i.$$

Proof. If $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ are nonconflicting, the theorem is trivially true. Otherwise, by Lemma 4.4, we have

$$\parallel_{i \in J_h \cup J_k} (A_i \parallel CM_i) \equiv (SUP^h \parallel SUP^k \parallel CR^{\{h,k\}}),$$

where $CR^{\{h,k\}}$ is a conflict resolution for $\mathcal{N}_1^h$ and $\mathcal{N}_1^k$ computed in Step 2 of *DeconflictBasicSubnet*.

By Definition 4.7, $(SUP^h \parallel SUP^k \parallel CR^{\{h,k\}}) \equiv SUP^{\{h,k\}}$. It follows that

$$\parallel_{i \in J_h \cup J_k} (A_i \parallel CM_i) \equiv SUP^{\{h,k\}}.$$

Hence the theorem. $\square$

4.7.2 Composing Arbitrary Subnets

With slight modifications, the theoretical results presented in the previous section can be generalized to composing two subnets $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$ of sizes $x, y \in I_m$, to form a larger $(x+y)$ -constraint subnet. In doing so, we follow the same compositional approach, i.e., we first synthesize the CM's for each individual subnet, and then design deconflicting CM's for the agents concerned to resolve any conflict between two subnets.

Given two subnets $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$, $x, y \in I_m$, let $SUP^{\mathcal{S}_x} = Supcon(\parallel_{k \in \mathcal{S}_x} C_{J_k}^k, \parallel_{k \in \mathcal{S}_x} A_{J_k})$, $SUP^{\mathcal{S}_y} = Supcon(\parallel_{k \in \mathcal{S}_y} C_{J_k}^k, \parallel_{k \in \mathcal{S}_y} A_{J_k})$, and $SUP^{\mathcal{S}_x \cup \mathcal{S}_y} = Supcon(\parallel_{k \in \mathcal{S}_x \cup \mathcal{S}_y} C_{J_k}^k, \parallel_{k \in \mathcal{S}_x \cup \mathcal{S}_y} A_{J_k})$. Then, generalizing Definition 4.7 as follows is Definition 4.8, which formalizes the basis for conflict resolution between two subnets $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$.

Definition 4.8. An automaton $CR^{\mathcal{S}_x \cup \mathcal{S}_y}$ is said to be a conflict resolution for $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$ if $(CR^{\mathcal{S}_x \cup \mathcal{S}_y} \parallel SUP^{\mathcal{S}_x} \parallel SUP^{\mathcal{S}_y}) \equiv SUP^{\mathcal{S}_x \cup \mathcal{S}_y}$.

Now, generalizing Lemma 4.3 is Lemma 4.5, which offers an efficient approach to compute a conflict resolution $CR^{\mathcal{S}_x \cup \mathcal{S}_y}$.

Lemma 4.5. For each agent A_i in subnet $\mathcal{N}_x^{\mathcal{S}_x}$, let $S_i^1, \dots, S_i^{r_i}$ be its r_i CM's with the respective event sets $\Sigma_i^1, \dots, \Sigma_i^{r_i}$, $r_i \geq 1$, such that using those CM's to coordinate, the agents can coordinate to satisfy every constraint in the subnet, i.e.,

$$\parallel_{A_i \in \mathcal{N}_x^{\mathcal{S}_x}} (A_i \parallel S_i^1 \parallel S_i^2 \dots \parallel S_i^{r_i}) \equiv SUP^{\mathcal{S}_x}.$$

The CM's $S_i^1, \dots, S_i^{r_i}$ consists of local CM's synthesized for A_i to satisfy every relevant constraint in every basic subnet that contains A_i and deconflicting CM's to resolve any conflict among those basic subnets. The CM's for the agents in subnet $\mathcal{N}_y^{\mathcal{S}_y}$ are defined similarly.

Let $\Sigma_{CR}^{S_x \cup S_y}$ be a superset of $\bigcup_{i \in J_k, k \in S_x \cap S_y} \Sigma^{A_i}$. Suppose all the following conditions are satisfied:

- $\Sigma_{CR}^{S_x \cup S_y} \supseteq \bigcap_{A_i \in \mathcal{N}_x^{S_x}} (\Sigma^{S_i^1} \cap \dots \cap \Sigma^{S_i^{r_i}})$.
- $\Sigma_{CR}^{S_x \cup S_y} \supseteq \bigcap_{A_i \in \mathcal{N}_y^{S_y}} (\Sigma^{S_i^1} \cap \dots \cap \Sigma^{S_i^{r_i}})$.
- $\forall A_i \in \mathcal{N}_x^{S_x} \cup \mathcal{N}_y^{S_y}, \forall 1 \leq j \leq r_i, P_{\Sigma_i^j, \Sigma_{CR}^{S_x \cup S_y}}$ is a $L_m(S_i^j)$ -observer.
- $\forall A_i \in \mathcal{N}_x^{S_x} \cup \mathcal{N}_y^{S_y}, P_{\Sigma^{A_i}, \Sigma_{CR}^{S_x \cup S_y}}$ is OCC for $L(A_i)$.

Then

$$CR^{S_x \cup S_y} = \text{Supcon}(G, \bigparallel_{A_i \in \mathcal{N}_x^{S_x} \cup \mathcal{N}_y^{S_y}} [P_{\Sigma_i^1, \Sigma_{CR}^{S_x \cup S_y}}(S_i^1) \parallel \dots \parallel P_{\Sigma_i^{r_i}, \Sigma_{CR}^{S_x \cup S_y}}(S_i^{r_i})])$$

is a conflict resolution for $\mathcal{N}_x^{S_x}$ and $\mathcal{N}_y^{S_y}$, where G is a one-state automaton that generates and marks $(\Sigma_{CR}^{S_x \cup S_y})^*$.

Proof. To begin with, for each $A_i \in \mathcal{N}_x^{S_x} \cup \mathcal{N}_y^{S_y}$ let $CM_i = S_i^1 \parallel S_i^2 \dots \parallel S_i^{r_i}$, and let $CM_X = \bigparallel_{A_i \in \mathcal{N}_x^{S_x}} CM_i$ and $CM_Y = \bigparallel_{A_i \in \mathcal{N}_y^{S_y}} CM_i$.

Then, since $\Sigma_{CR}^{S_x \cup S_y} \supseteq \bigcup_{i \in J_k, k \in S_x \cap S_y} \Sigma^{A_i}$, by Proposition 4.2, we have:

$$P_{\Sigma^{CM_X}, \Sigma_{CR}^{S_x \cup S_y}}(CM_X) \equiv \bigparallel_{A_i \in \mathcal{N}_x^{S_x}} [P_{\Sigma_i^1, \Sigma_{CR}^{S_x \cup S_y}}(S_i^1) \parallel \dots \parallel P_{\Sigma_i^{r_i}, \Sigma_{CR}^{S_x \cup S_y}}(S_i^{r_i})].$$

$$P_{\Sigma^{CM_Y}, \Sigma_{CR}^{S_x \cup S_y}}(CM_Y) \equiv \bigparallel_{A_i \in \mathcal{N}_y^{S_y}} [P_{\Sigma_i^1, \Sigma_{CR}^{S_x \cup S_y}}(S_i^1) \parallel \dots \parallel P_{\Sigma_i^{r_i}, \Sigma_{CR}^{S_x \cup S_y}}(S_i^{r_i})].$$

It follows that

$$\begin{aligned} CR^{S_x \cup S_y} &= \text{Supcon}(G, \bigparallel_{A_i \in \mathcal{N}_x^{S_x} \cup \mathcal{N}_y^{S_y}} [P_{\Sigma_i^1, \Sigma_{CR}^{S_x \cup S_y}}(S_i^1) \parallel \dots \parallel P_{\Sigma_i^{r_i}, \Sigma_{CR}^{S_x \cup S_y}}(S_i^{r_i})]) \\ &\equiv \text{Supcon}(G, P_{\Sigma^{CM_X}, \Sigma_{CR}^{S_x \cup S_y}}(CM_X) \parallel P_{\Sigma^{CM_Y}, \Sigma_{CR}^{S_x \cup S_y}}(CM_Y)). \end{aligned}$$

Next, since $\forall A_i \in \mathcal{N}_x^{\mathcal{S}_x}, \forall 1 \leq j \leq r_i, P_{\Sigma_i^j, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is a $L_m(S_i^j)$ -observer and $\Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y} \supseteq \bigcap_{A_i \in \mathcal{N}_x^{\mathcal{S}_x}} (\Sigma_i^{S_i^1} \cap \dots \cap \Sigma_i^{S_i^{r_i}})$, by Proposition 4.3, $P_{\Sigma^{CM_x}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is a $L_m(CM_x)$ -observer. Similarly, $P_{\Sigma^{CM_y}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is also a $L_m(CM_y)$ -observer.

Finally, since $P_{\Sigma^{CM_x}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is a $L_m(CM_x)$ -observer, $P_{\Sigma^{CM_y}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is a $L_m(CM_y)$ -observer and $\forall A_i \in \mathcal{N}_x^{\mathcal{S}_x} \cup \mathcal{N}_y^{\mathcal{S}_y}, P_{\Sigma^{A_i}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}$ is OCC for $L(A_i)$, by Proposition 4.4,

$$[Supcon(G, P_{\Sigma^{CM_x}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}(CM_x) \parallel P_{\Sigma^{CM_y}, \Sigma_{CR}^{\mathcal{S}_x \cup \mathcal{S}_y}}(CM_y))] \parallel SUP^{\mathcal{S}_x} \parallel SUP^{\mathcal{S}_y} \equiv SUP^{\mathcal{S}_x \cup \mathcal{S}_y}.$$

In other words, $CR^{\mathcal{S}_x \cup \mathcal{S}_y}$ is a conflict resolution for $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$.

Hence the lemma. $\square$

A procedure called *DeconflictSubnet*, which is almost identical to *DeconflictBasicSubnet*, can be used to design deconflicting CM's for every agent A_i in $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$. In essence, the procedure applies Lemma 4.5 to compute a conflict resolution $CR^{\mathcal{S}_x \cup \mathcal{S}_y}$ for $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$, and then localize it to every agent as the agent's deconflicting CM if the agent shares some events with the conflict resolution. Like *DeconflictBasicSubnet*, *DeconflictSubnet* has exponential complexity in the worst case but is often efficient in practice.

Theorem 4.2 summarizes how we can compose two subnets $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$ to form the subnet $\mathcal{N}_{x+y}^{\mathcal{S}_x \cup \mathcal{S}_y}$.

Theorem 4.2. For $i \in J_k, k \in \mathcal{S}_x \cup \mathcal{S}_y$, let CM_i be the CM set for agent A_i computed as follows: (i) CM_i includes every CM computed for A_i when applying Procedures *CMBasicSubet*, *DeconflictBasicSubnet* and *DeconflictSubnet* to synthesize CM's for the basic subnets and resolve the conflict that exists among the subnets of $\mathcal{N}_x^{\mathcal{S}_x}$ and of $\mathcal{N}_y^{\mathcal{S}_y}$, and (ii) CM_i includes every deconflicting CM computed for A_i when applying Procedure *DeconflictSubnet* to resolve the conflict that exists between $\mathcal{N}_x^{\mathcal{S}_x}$ and $\mathcal{N}_y^{\mathcal{S}_y}$. Then

$$\parallel_{i \in J_k, k \in \mathcal{S}_x \cup \mathcal{S}_y} (A_i \parallel CM_i) \equiv SUP^{\mathcal{S}_x \cup \mathcal{S}_y}.$$

Proof. Since, for $i \in J_k, k \in \mathcal{S}_x \cup \mathcal{S}_y, CM_i$ includes: (i) Every CM computed for A_i to satisfy its relevant constraints in $\mathcal{N}_{x+y}^{\mathcal{S}_x \cup \mathcal{S}_y}$, and (ii) every CM computed for A_i to resolve any

conflicts in $\mathcal{N}_{x+y}^{S_x \cup S_y}$ that A_i might involve, we have

$$\parallel_{i \in J_k, k \in S_x \cup S_y} (A_i \parallel CM_i) \equiv (CR^{S_x \cup S_y} \parallel SUP^{S_x} \parallel SUP^{S_y}),$$

where $CR^{S_x \cup S_y}$ is a conflict resolution for $\mathcal{N}_x^{S_x}$ and $\mathcal{N}_y^{S_y}$ computed by *DeconflictSubnet*. Therefore, by Definition 4.8, we have

$$\parallel_{i \in J_k, k \in S_x \cup S_y} (A_i \parallel CM_i) \equiv SUP^{S_x \cup S_y}.$$

Hence the theorem. □

Figure 4.8: A DCSN of 4 agents and 3 constraints. It is possible that in ensuring overall network nonblockingness, agents A_2 and A_4 might need to communicate with each other even though they are coordinating on different constraints with different agents.

In concluding this section, it is interesting to note that two agents in a DCSN might need to communicate with each other even though they are coordinating on different constraints with other agents. For example, in the DCSN in Figure 4.8, it is possible that when composing the two subnets $(\{1, 2\}, C^1_{1,2})$ and $((\{1, 3\}, C^2_{1,3}), (\{3, 4\}, C^3_{3,4}))$ using *DeconflictSubnet*, the computed conflict resolution automaton shares some events with both A_2 and A_4 . As a result, the deconflicting CM of A_2 will contain events of A_4 , and that of A_4 will contain events of A_2 . This means that the agents A_2 and A_4 will need to communicate between themselves to ensure nonblockingness of the DCSN, even though A_2 is coordinating on $C^1_{1,2}$ with A_1 and A_4 is coordinating on $C^3_{3,4}$ with A_3 .

Example 4.3. We now provide a solution for the manufacturing example using our compositional approach with conflict resolution.

Following Step 1 of our approach presented in Section 4.1, we use *CMBasicSubnet* to design three local CM's for each of the agents A_1 and A_2 , and two CM's for agent A_3 . Each of these local CM's corresponds to a relevant constraint of the agents.

In Step 2.1, we need to generate a conflict resolution plan to completely and correctly composing together the subnets of the DCSN presented in Figure 4.3. For a large DCSN, an algorithm that can automatically generate a conflict resolution plan is required. In the next chapter, we will provide such an algorithm. For this example, we simply assume that the following plan has been given: (i) compose $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$ to form $\mathcal{N}_2^{\{1,2\}}$, as well as $\mathcal{N}_1^3$ and $\mathcal{N}_1^4$ to form $\mathcal{N}_2^{\{3,4\}}$, using *DeconflictBasicSubnet* and (ii) compose $\mathcal{N}_2^{\{1,2\}}$ and $\mathcal{N}_2^{\{3,4\}}$ to form $\mathcal{N}$ using *DeconflictSubnet*.

In Step 2.2, we follow the conflict resolution plan generated in Step 1 to compose the subnets and design deconflicting CM's for the agents when necessary. For this example, we only need to design deconflicting CM's when composing $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$, since $\mathcal{N}_1^3$ and $\mathcal{N}_1^4$, as well as $\mathcal{N}_2^{\{1,2\}}$ and $\mathcal{N}_2^{\{3,4\}}$, are found to be nonconflicting.

The complete solution is shown in Figure 4.9.

Figure 4.9: Complete CM solution for a manufacturing transfer line system.

4.8 Chapter Summary

In this chapter, we have introduced and formulated a networked coordination problem and shown how it can be addressed based on a new formalism called the Distributed Constraint

Specification Network (DCSN) for organizing a system of agents and constraints. We have also discussed how subnets in a given DCSN might conflict with one another, and presented an efficient approach to resolve those conflicts.

To reiterate, a DCSN, when graphically represented, can explicitly show the interconnections among agents and their relevant constraints in a multiagent system. Importantly, besides furnishing a comprehensible specification, a DCSN can be systematically decomposed into constraint subnets for individual CM synthesis and conflict resolution, and provide opportunities to mitigate the complexity of synthesizing CM's for a network of coordinating agents. This contribution is detailed in the next chapter.

Our DCSN formalism is reminiscent of a process communication graph [92] that has been used in supervisory control research to organize components of a large DES. However, while our purpose of organizing agents and constraints into a DCSN is to efficiently synthesize and resolve any possible conflict between constraint subnets, a process communication graph [92] is used mainly to speed up the process of computing a supervisor for a large DES.

Our DCSN formalism is also reminiscent of the constraint network basis for solving the distributed constraint satisfaction problem (DCSP) [108]. However, while the DCSP framework provides the foundation for addressing the coordination problem of agents modeled by (a set of) variables with numerical domains, our work provides a different but complementary distributed constraint satisfaction foundation for agents modeled by DEP's.

Chapter 5

NETWORKED AGENT COORDINATION PROBLEM - PART II: REPRESENTATION AND GENERATION OF CONFLICT RESOLUTION PLANS

5.1 Chapter Overview

In this chapter, we continue with our study of the networked coordination problem. In Chapter 4, the problem has been formulated and a compositional synthesis approach has been proposed and overviewed. The proposed approach takes as input a DCSN describing the inter-connections among agents and their distributed constraints, and generates as output a set of CM's for each agent that satisfies every constraint in the DCSN. Recall that our compositional approach consists of the following steps.

- **Step 1 Basic Subnet Synthesis**

Synthesize for every agent a set of $\parallel$ -connected local CM's, one for each of the agent's relevant constraints. This step is performed by applying Procedure *CMBasicSubnet* (Figure 4.4) to every basic subnet of the DCSN.

- **Step 2 Subnet Composition**

- **Step 2.1 Conflict Resolution Plan Generation**

Generate a conflict resolution plan for the given DCSN.

- **Step 2.2 Conflict Resolution Plan Execution**

Compose subnets with conflict resolution by following a precedence order of subnet composition operations in the plan. This is to completely deconflict the local CM's synthesized in Step 1. Each subnet composition operation entails applying Procedure *DeconflictBasicSubnet* (Figure 4.7) or its straightforward generalization *DeconflictSubnet* (discussed in Section 4.7.2) to design additional deconflicting CM's for the agents concerned so as to ensure nonblockingness of the composed subnet.

The procedures *CMBasicSubnet*, *DeconflictBasicSubnet* and *DeconflictSubnet* lay an algorithmic foundation for Steps 1 and 2.2, and were already discussed in detail in Chapter 4. In this chapter, filling in Step 2.1 of our compositional synthesis approach, we present the theory and algorithms for representing and generating conflict resolution plans. We assume that a subnet composition is an operation on two subnets. While in theory, more than two subnets can be composed together to form a larger subnet, there is no evidence that it would be computationally more advantageous.

A conflict resolution plan for a given DCSN is a finite number of subnet composition operations, with ordering constraints between them. Such a plan may encompass several complete planning sequences, each of which is an ordered sequence of the subnet composition operations that satisfies all the ordering constraints. Executing a given plan means following one of its complete planning sequences to successively compose (the solution of) different pairs of subnets to form (the solution of) larger subnets, starting with all basic subnets “disconnected” from each other, and ending with all of them correctly composed to form the DCSN.

For a relatively large DCSN, there are often many conflict resolution plans and the choice of plans can affect the efficiency of the CM synthesis process. For example, one plan may allow more simultaneity in the execution of subnet composition operations than others, and this can often result in less total execution time when sufficient computing resources are available for concurrent computations. The main objective of this chapter is to develop

a general framework for the automatic generation of an optimal conflict resolution plan (based on some efficiency criterion) for a DCSN.

The main contributions of this chapter are as follows.

- First, we present a compact representation encompassing all conflict resolution plans for a DCSN using AND/OR graphs [20]. An AND/OR graph representation of conflict resolution plans provides the search space for automatic selection of an optimal plan. In an AND/OR graph plan, the nodes represent the DCSN's subnets, and each edge connects two subnets to a larger subnet, denoting the composition of the former to form the latter. A conflict resolution plan is then represented by a tree starting from the root node representing the complete DCSN, and terminating at the leaf nodes representing the basic subnets. While many other data structures can also be used to represent conflict resolution plans, they often present several limitations when compared to the AND/OR graph representation. To elaborate, it is possible to represent plans by an exhaustive enumeration of complete planning sequences. But since many complete planning sequences share common subsequences, enumerating them separately is space inefficient. Directed graphs can also be used to represent and encompass conflict resolution plans for a DCSN. However, as will be illustrated later in this chapter, when compared to directed graphs, AND/OR graphs are often more compact, and hence more efficient to work with. In addition, as will be shown, one benefit of the AND/OR graph representation is that it shows explicitly the possibility of executing subnet composition operations in parallel.
- Next, we develop an algorithm to generate an AND/OR graph representation of conflict resolution plans for a DCSN. The novel idea behind the algorithm is in converting a DCSN to an alternative representation called constraint relational network (CRN) that is amenable to efficient decomposition by applying cut-set theory for recursive plan generation. In essence, the CRN of a DCSN is a constraint relational model which graphically connects with an edge every pair of inter-agent constraints whose agent groups, as induced under the DCSN, overlap. Each overlap means that there is (at least) a common member agent of the two agent groups that may run into synchronization conflict between the two constraints when coordinating with the other member agents. In other words, an edge of the CRN indicates potential conflicts to be resolved between two basic subnets. The reverse of a composition of two po-

tentially conflicting subnets of a DCSN into a larger subnet is the decomposition of the larger subnet into the two component subnets, and can be shown to correspond to the removal of a cut-set [20] of the CRN. It follows that the AND/OR graph of conflict resolution plans can be generated by recursively enumerating all the cut-sets of the CRN's of a DCSN and its subnets. Importantly, the AND/OR graph generation algorithm can be easily extended to generate only conflict resolution plans which satisfy some problem-dependent requirements. In this direction, we explain how the algorithm can be extended to support planning for multiagent nonblocking reconfigurability

- Finally, we develop a heuristic search algorithm to select an optimal plan from an AND/OR graph of conflict resolution plans. The proposed algorithm is based on the general heuristic A^* search algorithm [77] and entails incorporating an admissible heuristic [55] to guarantee some criterion-based optimality of the selected plan. As a first effort, we present the design of an important heuristic for selecting a conflict resolution plan that meets the criterion of maximal simultaneity in the execution of subnet compositions, thereby effectively managing synthesis complexity and speeding up the process of automated coordination design.

The rest of this chapter is organized as follows. In Section 5.2, we show how an AND/OR graph can be used as a compact representation of conflict resolution plans for a given DCSN. Next, Section 5.3 introduces the concept of a CRN, and proposes an algorithm that generates an AND/OR graph of conflict resolution plans for a given DCSN by performing recursive decomposition on its CRN. A heuristic algorithm to search for an optimal conflict resolution plan over the AND/OR graph representation is then presented in Section 5.4. Finally, Section 5.5 concludes this chapter.

5.2 Representations of Conflict Resolution Plans

In this section, we show how the space of conflict resolution plans for a DCSN can be represented. In Section 5.2.1, we describe a representation of conflict resolution plans using directed graph. In Section 5.2.2, an alternative representation of conflict resolution plans using AND/OR graphs [53] is presented. We then show that, when compared to a directed

graph of conflict resolution plans, the equivalent AND/OR graph often has a significantly smaller number of nodes.

Similar to Chapter 4, in this chapter, we shall use small letters such as n, m, k, r to denote integers, and for an integer $n \geq 1$, the symbol I_n denotes the index set $\{1, 2, \dots, n\}$.

5.2.1 Directed Graph Representation of Conflict Resolution Plans

Let $\mathcal{N}$ be a DCSN of n agents and m inter-agent constraints, $n \geq 2, m \geq 1$. Recall from Definition 4.4 that for $\mathcal{S}_r \subseteq I_m$, $\mathcal{N}_r^{\mathcal{S}_r} \subseteq \mathcal{N}$ denotes a r -constraint subnet of $\mathcal{N}$, which consists of $r = |\mathcal{S}_r| \geq 1$ basic subnets $\{(J_k, C_{J_k}^k) \mid k \in \mathcal{S}_r\}$. When the set of basic subnets is arbitrary, a r -constraint subnet is simply denoted by $\mathcal{N}_r$.

Definition 5.1. A r -constraint subnet $\mathcal{N}_r$ of $\mathcal{N}$, $r \geq 1$, is said to be *constraint-connected* if the DCSN graph representing $\mathcal{N}_r$ is a connected graph¹.

In words, in a constraint-connected subnet, every agent is connected to every other agent either directly by a mutually relevant inter-agent constraint or indirectly through a path of intermediate constraints and agents. The implication is that every agent might need to coordinate with every other agent in the subnet, either to achieve conformance to mutually relevant inter-agent constraints or to ensure overall nonblockingness of the subnet. By Definition 5.1, a basic subnet is trivially constraint-connected.

If a DCSN $\mathcal{N}$ is not constraint-connected, it contains two or more smaller but constraint-connected DCSN's and their networked coordination problems can be independently solved. Without loss of generality, we shall henceforth assume that a DCSN $\mathcal{N}$ is constraint-connected.

Example 5.1. In illustrating our theoretical development throughout this chapter, we shall refer to Figure 5.1(a) for a specific DCSN $\mathcal{N}$ of 10 agents and 4 constraints. The DCSN $\mathcal{N}$ has four basic subnets: $\mathcal{N}_1^1 = (\{1, 2, 3, 4, 9\}, C_{\{1,2,3,4,9\}}^1)$, $\mathcal{N}_1^2 = (\{3, 4, 5, 6\}, C_{\{3,4,5,6\}}^2)$, $\mathcal{N}_1^3 = (\{6, 7, 8\}, C_{\{6,7,8\}}^3)$ and $\mathcal{N}_1^4 = (\{8, 9, 10\}, C_{\{8,9,10\}}^4)$. By Definition 5.1, this $\mathcal{N}$ is constraint-connected since its DCSN graph is connected [see Figure 5.1(a)].

Figures 5.1(b) and 5.1(c) show two subnets $\mathcal{N}_2^{\{1,2\}}$ and $\mathcal{N}_2^{\{1,4\}}$ of $\mathcal{N}$. The former is a constraint-connected subnet of $\mathcal{N}$ since its DCSN graph is connected. The latter is not since it has two subnets $\mathcal{N}_1^1$ and $\mathcal{N}_1^4$ which are disconnected as they have no agents in common.

¹A graph is said to be connected if there is a path from every node in the graph to every other node [20].

Figure 5.1: A specific DCSN of 10 agents and 4 constraints, and its subnets.

Thus a subnet of a constraint-connected DCSN may not necessary be constraint-connected itself.

Definition 5.2. Given a DCSN $\mathcal{N}$ consisting of m basic subnets $\mathcal{N}_1^1, \dots, \mathcal{N}_1^m$, a subnet-decomposition Φ is a set of subnets of $\mathcal{N}$ such that:

1. Every element subnet of Φ is constraint-connected .
2. Every basic subnet of $\mathcal{N}$ is contained in one of the elements of Φ .
3. There is no basic subnet of $\mathcal{N}$ that is contained in two different elements of Φ .

Thus a subnet-decomposition is simply a partition of (the set of all basic subnets of) $\mathcal{N}$. It follows that a conflict resolution plan for $\mathcal{N}$ is a sequence of transitions of subnet-decompositions, starting with the initial subnet-decomposition $\Phi_I = \{\mathcal{N}_1^1, \mathcal{N}_1^2, \dots, \mathcal{N}_1^m\}$ and ending with the final subnet-decomposition $\Phi_F = \{\mathcal{N}\}$. The initial subnet-decomposition

Φ_I characterizes the situation in which all the basic subnets are “disconnected” from each other, and the final subnet-decomposition Φ_F characterizes the situation where all the basic subnets are already composed together to form the complete DCSN $\mathcal{N}$. Each transition from one subnet-decomposition to another characterizes an operation of composing (the CM solutions) of subnets to form (a CM solution of) the larger subnets. A conflict resolution plan should only include transitions that correspond to the compositions of subnets that contain common agents, since subnets that contain no common agents are trivially non-conflicting. Note that $\Phi_F = \{\mathcal{N}\}$ is a subnet-decomposition since the given DCSN $\mathcal{N}$ is assumed to be constraint-connected.

Based on the above discussion, all possible sequences of subnet composition operations for a DCSN can be encompassed in a directed graph formally defined as follows.

Definition 5.3. *The directed graph of conflict resolution plans for a DCSN $\mathcal{N}$ is a tuple $P_{\mathcal{N}} = (C_{\mathcal{N}}, E_{\mathcal{N}})$, where*

1. $C_{\mathcal{N}}$ is the set of nodes of $P_{\mathcal{N}}$ and defined as $C_{\mathcal{N}} = \{\Phi \mid \Phi \text{ is a subnet-decomposition of } \mathcal{N}\}$.
2. $E_{\mathcal{N}}$ is the set of edges of $P_{\mathcal{N}}$ and defined as $E_{\mathcal{N}} = \{(\Phi, \Psi) \mid (\Phi, \Psi) \in C_{\mathcal{N}} \times C_{\mathcal{N}} \text{ and } \Psi \text{ is formed from } \Phi \text{ by combining any two of its element subnets that contain common agents}\}$.

Generating a conflict resolution plan for $\mathcal{N}$ is then equivalent to searching in $P_{\mathcal{N}}$ for a simple directed path from $\Phi_I \in C_{\mathcal{N}}$ to $\Phi_F \in C_{\mathcal{N}}$, which is a finite sequence of $p \geq 2$ nodes $n_1, n_2, \dots, n_p$, where

1. $(\forall 1 \leq i \leq p) n_i \in C_{\mathcal{N}}$.
2. $n_1 = \Phi_I, n_p = \Phi_F$.
3. $(\forall 1 \leq i, j \leq p) i \neq j \Rightarrow n_i \neq n_j$.
4. $(\forall 1 \leq i \leq p-1) (n_i, n_{i+1}) \in E_{\mathcal{N}}$.

Example 5.2. *In the DCSN in Figure 5.1(a), the following pairs of basic subnets contain common agents: $(\mathcal{N}_1^1, \mathcal{N}_1^2)$, $(\mathcal{N}_1^1, \mathcal{N}_1^3)$, $(\mathcal{N}_1^2, \mathcal{N}_1^4)$ and $(\mathcal{N}_1^3, \mathcal{N}_1^4)$. Thus, by Definition 5.1, besides $\mathcal{N}$ and the four basic subnets, the following subnets are constraint-connected: $\mathcal{N}_2^{\{1,2\}}$, $\mathcal{N}_2^{\{1,3\}}$, $\mathcal{N}_2^{\{2,4\}}$, $\mathcal{N}_2^{\{3,4\}}$, $\mathcal{N}_3^{\{1,2,3\}}$, $\mathcal{N}_3^{\{1,2,4\}}$, $\mathcal{N}_3^{\{1,3,4\}}$ and $\mathcal{N}_3^{\{2,3,4\}}$.*

Figure 5.2: The directed graph of conflict resolution plans for the DCSN in Figure 5.1(a).

Following Definition 5.3, Figure 5.2 shows the directed graph of conflict resolution plans for $\mathcal{N}$, which has 12 nodes representing 12 possible subnet-decompositions of $\mathcal{N}$. A complete planning sequence is then represented by a simple directed path from the node $\{\mathcal{N}_1^1, \mathcal{N}_1^2, \mathcal{N}_1^3, \mathcal{N}_1^4\}$ to the node $\{\mathcal{N}\}$. In Figure 5.2, there are 12 such directed paths, meaning that there are 12 complete planning sequences for $\mathcal{N}$. Four such directed paths are extracted from the directed graph in Figure 5.2 and displayed in Figure 5.3 for illustration.

(a) Directed path representing the first complete planning sequence

(b) Directed path representing the second complete planning sequence

(c) Directed path representing the third complete planning sequence

(d) Directed path representing the fourth complete planning sequence

Figure 5.3: Four complete planning sequences for the DCSN in Figure 5.1(a).

In following the sequence in Figure 5.3(a), we first compose $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$ to form $\mathcal{N}_2^{\{1,2\}}$,

then $\mathcal{N}_2^{\{1,2\}}$ and $\mathcal{N}_1^3$ to form $\mathcal{N}_3^{\{1,2,3\}}$, and finally $\mathcal{N}_3^{\{1,2,3\}}$ and $\mathcal{N}_1^4$ to form $\mathcal{N}$. The sequence in Figure 5.3(b) requires us to first compose $\mathcal{N}_1^2$ and $\mathcal{N}_1^4$ to form $\mathcal{N}_2^{\{2,4\}}$, then $\mathcal{N}_2^{\{2,4\}}$ and $\mathcal{N}_1^3$ to form $\mathcal{N}_3^{\{2,3,4\}}$, and finally $\mathcal{N}_3^{\{2,3,4\}}$ and $\mathcal{N}_1^1$ to form $\mathcal{N}$. The sequences of composition operations in Figures 5.3(c) and 5.3(d) can be similarly described.

5.2.2 AND/OR Graph Representation of Conflict Resolution Plans

We have shown that for a DCSN $\mathcal{N}$, the directed graph $P_{\mathcal{N}} = (C_{\mathcal{N}}, E_{\mathcal{N}})$ represents and encompasses conflict resolution plans. However, encompassing conflict resolution plans in $P_{\mathcal{N}}$ has a major drawback: The number of nodes $|C_{\mathcal{N}}|$, which is equal to the number of subnet-decompositions, may become very large when $\mathcal{N}$ contains many pairs of basic subnets that have common agents. In fact, if a DCSN with m basic subnets has the maximum number $\frac{1}{2} \times m \times (m - 1)$ of such distinct pairs, i.e., it has an exhaustive interconnection structure described as “strongly connected” that we shall formally define later, $|C_{\mathcal{N}}|$ is equal to the number of partitions of a set of m elements, which is 115, 975 for $m = 10$ and 1, 382, 958, 535 for $m = 15$.

Remark 5.1. *The number of partitions of a set of $m \geq 0$ elements, denoted by $\text{partition}(m)$, can be computed recursively as follows [22].*

1. $\text{partition}(0) = 1$, and
2. For $m \geq 1$, $\text{partition}(m) = \sum_{k=0}^{m-1} \binom{m}{k} \times \text{partition}(m - k - 1)$.

In this section, we present a more compact representation of conflict resolution plans using AND/OR graphs [53].

Observe that a conflict resolution planning sequence for a DCSN $\mathcal{N}$ is a reversal of a successive decomposition, starting with $\mathcal{N}$, of constraint-connected component subnets until only basic subnets remain. This suggests that the forward search problem of generating conflict resolution plans for a DCSN $\mathcal{N}$ can be addressed as a backward search problem of successively decomposing $\mathcal{N}$ into pairs of constraint-connected component subnets until only basic subnets are left. The space of all possible conflict resolution plans for $\mathcal{N}$ can therefore be generated by enumerating all possible ways of successively decomposing $\mathcal{N}$ this way.

Motivated by the foregoing discussion, we propose an alternative representation using AND/OR graphs [53] for the conflict resolution plans of a DCSN $\mathcal{N}$. The AND/OR graph representation is equivalent to the directed graph representation discussed, but requires fewer nodes in general and is therefore more compact. It forms the basis of a backward search algorithm for conflict resolution plan generation, and also simplifies the heuristic search for optimal plans.

Definition 5.4. *The AND/OR graph of conflict resolution plans for a DCSN $\mathcal{N}$ is a hypergraph $T_{\mathcal{N}} = (S_{\mathcal{N}}, H_{\mathcal{N}})$, where*

1. $S_{\mathcal{N}}$ is the set of nodes of $T_{\mathcal{N}}$ and defined as $S_{\mathcal{N}} = \{\mathcal{N}_r \subseteq \mathcal{N} \mid \mathcal{N}_r \text{ is constraint-connected}\}$.
2. $H_{\mathcal{N}}$ is the set of hyper-edges of $T_{\mathcal{N}}$ and defined as $H_{\mathcal{N}} = \{(\mathcal{N}_{r_1}, (\mathcal{N}_{r_2}, \mathcal{N}_{r_3})) \in S_{\mathcal{N}} \times (S_{\mathcal{N}} \times S_{\mathcal{N}}) \mid \mathcal{N}_{r_2} \cap \mathcal{N}_{r_3} \neq \emptyset \text{ and } \mathcal{N}_{r_1} = \mathcal{N}_{r_2} \cup \mathcal{N}_{r_3}\}$.

The nodes in the AND/OR graph $T_{\mathcal{N}}$ represent constraint-connected subnets of $\mathcal{N}$, and each of the hyper-edges is a pair $(\mathcal{N}_{r_1}, (\mathcal{N}_{r_2}, \mathcal{N}_{r_3}))$ denoting the decomposition of subnet $\mathcal{N}_{r_1}$ into two component subnets $\mathcal{N}_{r_2}$ and $\mathcal{N}_{r_3}$, or equivalently, the composition of $\mathcal{N}_{r_2}$ and $\mathcal{N}_{r_3}$ into $\mathcal{N}_{r_1}$. A hyper-edge points from a node representing a subnet to two nodes representing the component subnets. The node that represents the complete DCSN $\mathcal{N}$ is referred to as the root node and denoted by n_{root} , and the nodes representing basic subnets of $\mathcal{N}$ are referred to as the leaf nodes. The set of all leaf nodes of $T_{\mathcal{N}}$ is $\{\mathcal{N}_1 \subseteq \mathcal{N} \mid \mathcal{N}_1 \text{ is a basic subnet of } \mathcal{N}\}$, and is denoted by Θ_{leaf} .

In what follows, a conflict resolution plan for $\mathcal{N}$ is represented by a tree in $T_{\mathcal{N}}$ that starts at n_{root} and terminates at Θ_{leaf} . Formally, a tree $tree$ in the AND/OR graph $T_{\mathcal{N}} = (S_{\mathcal{N}}, H_{\mathcal{N}})$, starting at a node $n_I \in S_{\mathcal{N}}$ and terminating at a set of nodes $\Theta \subseteq S_{\mathcal{N}}$, can be described recursively as follows.

- If $n_I \in \Theta$, $tree$ contains only one node n_I and no edge, and we write $tree = (n_I)$.
- Otherwise, $tree$ contains the node n_I , an edge $h = (n_I, (n_1, n_2)) \in H_{\mathcal{N}}$, and the nodes and edges of two trees $tree_1$ and $tree_2$. Each tree $tree_i, i \in \{1, 2\}$, starts from one of n_I 's two successors, n_i , and terminates at some $\Theta_i \subseteq \Theta$, where Θ_1 and Θ_2 are disjoint and $\Theta_1 \cup \Theta_2 = \Theta$. In this case, we write $tree = (n_I, h, tree_1, tree_2)$.

The set of all trees starting from n_I and terminating at Θ is denoted by $Trees(n_I, \Theta)$. If $tree \in Trees(n_I, \Theta)$, n_I is called the root node of $tree$ and a node in Θ called a terminal

node of *tree*. Whenever the set of terminal nodes is arbitrary, the set of trees starting from a node n_I is simply denoted by $Trees(n_I, -)$, and the set of all trees of $T_{\mathcal{N}}$ is denoted by $Trees(-, -)$.

A tree in $Trees(n_{root}, \Theta_{leaf})$ is said to be complete. Formally then, a complete tree is a conflict resolution plan. Any tree in $T_{\mathcal{N}}$ whose root node is not n_{root} or whose leaf nodes are not all in Θ_{leaf} is called a non-complete tree. A non-complete tree is a subgraph² of one or more complete trees. A non-complete tree whose root node is n_{root} is called a partial tree. In what follows, a tree in $Trees(n_{root}, -)$ is a partial conflict resolution plan.

Example 5.3. Figure 5.4 shows the AND/OR graph of conflict resolution plans for the DCSN of Figure 5.1(a). Since the DCSN has 13 constraint-connected subnets, its AND/OR graph has 13 nodes, each of which represents a constraint-connected subnet. The root node represents the DCSN and there are four leaf nodes representing the four basic subnets $\mathcal{N}_1^1$, $\mathcal{N}_1^2$, $\mathcal{N}_1^3$, and $\mathcal{N}_1^4$. Three complete trees are extracted from the AND/OR graph and displayed in Figure 5.5 for illustration.

In Figure 5.5, the trees in Figures 5.5(a) and 5.5(b) contain the planning sequences in Figures 5.3(a) and 5.3(b), respectively, and the tree in Figure 5.5(c) contains the two planning sequences in Figures 5.3(c) and 5.3(d). In fact, each complete tree in $T_{\mathcal{N}}$ contains a subset of complete planning sequences in the directed graph $P_{\mathcal{N}}$.

In comparison to the directed graph of conflict resolution plans, encompassing conflict resolution plans using AND/OR graphs is often more advantageous in terms of memory storage. The reason is that, in general, $T_{\mathcal{N}}$ has a significantly smaller number of nodes than $P_{\mathcal{N}}$, especially when the number of basic subnets of $\mathcal{N}$ is relatively large. From Definitions 5.3 and 5.4, the number of nodes $|S_{\mathcal{N}}|$ of $T_{\mathcal{N}}$ is equal to the number of constraint-connected subnets of $\mathcal{N}$, while the number of nodes $|C_{\mathcal{N}}|$ of $P_{\mathcal{N}}$ is equal to the number of subnet-decompositions of $\mathcal{N}$. In the following, two opposite and extreme cases of “strongly connected” and “weakly connected” DCSN’s are presented to compare the number of nodes of $T_{\mathcal{N}}$ and $P_{\mathcal{N}}$.

A DCSN is said to be strongly connected if the agent group of every basic subnet overlaps with the agent group of every other basic subnet in the DCSN. It then follows that every subnet of a strongly connected DCSN is constraint-connected. Thus, if $\mathcal{N}$ is a strongly con-

²A graph $G = (V, E)$ is said to be a subgraph of another graph $G' = (V', E')$ if $V \subseteq V'$ and $E \subseteq E'$, namely, the nodes and edges of G are all contained in G' .

Figure 5.4: The AND/OR graph of conflict resolution plans for the DCSN of Figure 5.1(a).

nected DCSN with m basic subnets, $|S_{\mathcal{N}}|$ is equal to the number of nonempty subsets of a set of m elements, namely, $|S_{\mathcal{N}}| = 2^m - 1$; and $|C_{\mathcal{N}}|$ is equal to the number of partitions of a set of m elements (see Remark 5.1), which is often much larger than $2^m - 1$ as m increases. Table 5.1 presents a comparison between the number of nodes in $P_{\mathcal{N}}$ and $T_{\mathcal{N}}$ for strongly connected DCSN's.

Figure 5.5: Three AND/OR conflict resolution plans for the DCSN in Figure 5.1(a).

Table 5.1: Number of nodes in directed graph and AND/OR graph representations of conflict resolution plans for strongly connected DCSN's

m : Number of basic subnets in a strongly connected DCSN $\mathcal{N}$	$ C_{\mathcal{N}} = \text{partition}(m)$: Number of nodes in the directed graph $P_{\mathcal{N}}$	$ S_{\mathcal{N}} = 2^m - 1$: Number of nodes in the AND/OR graph $T_{\mathcal{N}}$
1	1	1
2	2	3
3	5	7
4	15	15
5	52	31
6	203	63
7	877	127
8	4140	255
9	21147	511
10	115975	1023

A DCSN $\mathcal{N}$ containing m basic subnets is said to be weakly connected if the agent group of the i th basic subnet overlaps only with the agent group of the $(i + 1)$ th basic subnet, for $1 \leq i \leq m - 1$. In this case, we have $|S_{\mathcal{N}}| = \sum_{r=1}^m r = \frac{1}{2} \times m \times (m + 1)$, since for $1 \leq r \leq m$, there are exactly $(m + 1 - r)$ subnets of r constraints which are constraint-connected. $|C_{\mathcal{N}}|$, on the other hand, is equal to the number of subnet connections, which is $2^{(m-1)}$. Table 5.2 presents a comparison between the number of nodes in $P_{\mathcal{N}}$ and $T_{\mathcal{N}}$ for weakly connected DCSN's.

Table 5.2: Number of nodes in the directed graph and AND/OR graph representations of conflict resolution plans for weakly connected DCSN's

m : Number of basic subnets in a weakly connected DCSN $\mathcal{N}$	$ C_{\mathcal{N}} = 2^{m-1}$: Number of nodes in the directed graph $P_{\mathcal{N}}$	$ S_{\mathcal{N}} = \frac{1}{2} \times m \times (m + 1)$: Number of nodes in the AND/OR graph $T_{\mathcal{N}}$
1	1	1
2	2	3
3	4	6
4	8	10
5	16	15
6	32	21
7	64	28
8	128	36
9	256	45
10	512	55

As shown in Tables 5.1 and 5.2, in the two opposite extremes of strongly connected and weakly connected DCSN's, the AND/OR graph representation has fewer nodes than the directed graph representation when the number of basic subnets exceeds 4, and the advantage becomes greater as this number further increases. This reduction is due to the fact that the nodes in the directed graph representation denote subnet-decompositions whereas the nodes in the AND/OR graph representation denote subnets which are constituents of subnet-decompositions. With no two nodes containing the same constituent subnet, a complete AND/OR tree is a partially ordered plan encompassing several directed graph planning sequences. In general, the extent of reduction in the number of nodes depends on the con-

nectivity structure among the basic subnets of the DCSN. As the two tables also show, the advantage of using AND/OR graphs is greater for strongly connected DCSN's. In fact, the more "connected" the basic subnets of a DCSN are, the more economical the AND/OR graph representation would be.

In addition to being more compact, another advantage of AND/OR graphs over directed graphs is that an AND/OR graph of conflict resolution plans shows explicitly the possibility of executing subnet composition operations in parallel. Formally, if a tree $stree = (n_s, h_s, stree_1, stree_2)$ is a subgraph of a conflict resolution plan $tree$, there is no ordering constraint between subnet composition operations represented by h_1 and h_2 , where h_1 is an arbitrary edge of $stree_1$ and h_2 is an arbitrary edge of $stree_2$. For example, in the tree in Figure 5.5(c), there is no ordering constraint between the compositions of $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$ and of $\mathcal{N}_1^3$ and $\mathcal{N}_1^4$, meaning that these operations can be performed in any sequential order or in parallel.

5.3 Generation of AND/OR Graph Representation of Conflict Resolution Plans

We now present an algorithm for generating the AND/OR graph representation of conflict resolution plans. Our algorithm takes as input a DCSN and generates as output the AND/OR graph representation of conflict resolution plans for the DCSN.

The basic idea of our algorithm is to first enumerate all possible decompositions of a DCSN $\mathcal{N}$ into two constraint-connected component subnets. Each such decomposition corresponds to an edge of the AND/OR graph $T_{\mathcal{N}}$ connecting the root node representing $\mathcal{N}$ to two nodes, with each representing a component subnet. The same decomposition process is then repeated for each of the component subnets, which are component DCSN's, until only basic subnets are left. Recursive decomposition lends itself to straightforward AND/OR graph construction of all conflict resolution plans.

To facilitate the systematic enumeration of all possible decompositions of a subnet in a DCSN, we first convert the DCSN to a constraint relational network (CRN). In essence, the CRN of a DCSN, formally defined in Definition 5.5 below, is a constraint relational model which explicitly relates every pair of inter-agent constraints whose induced agent groups overlap.

Definition 5.5. The constraint relational network (CRN) $\mathcal{CRN}_r$ of a r -constraint subnet $\mathcal{N}_r^{\mathcal{S}_r} = \{(J_k, C_{J_k}^k) \mid k \in \mathcal{S}_r\}$ is a tuple $(\mathcal{C}_r, \mathcal{R}_r)$, where $\mathcal{C}_r = \{C_{J_k}^k \mid k \in \mathcal{S}_r\}$ is the constraint set of size r in $\mathcal{N}_r$ and $\mathcal{R}_r \subseteq \mathcal{C}_r \times \mathcal{C}_r$ is a relation over $\mathcal{C}_r$, such that $(\forall C_{J_k}^k, C_{J_h}^h \in \mathcal{C}_r)[(C_{J_k}^k, C_{J_h}^h) \in \mathcal{R}_r \Leftrightarrow (J_k \cap J_h \neq \emptyset)]$.

By Definition 5.5, two constraints $C_{J_k}^k$ and $C_{J_h}^h$ are related if their induced agent groups are overlapping, i.e., $J_k \cap J_h \neq \emptyset$, meaning that there is at least one agent A_i , where $i \in J_k \cap J_h$, that belongs to both the basic subnets $(J_k, C_{J_k}^k)$ and $(J_h, C_{J_h}^h)$. In other words, A_i has to coordinate on $C_{J_k}^k$ with some agents, and on $C_{J_h}^h$ with some other agents. As already discussed in Chapter 4, conflicts between (the agents in) such a pair of subnets may arise, and hence, there is a need to check for and resolve any conflict when composing the subnets.

Graphically, a CRN can be represented by an undirected graph with constraints represented by nodes, and the relation between two agent-related constraints $C_{J_k}^k$ and $C_{J_h}^h$ by an edge that connects the corresponding two nodes and is labeled with the agent group overlap between the subnets $(J_k, C_{J_k}^k)$ and $(J_h, C_{J_h}^h)$.

Figure 5.6: The CRN of the DCSN in Figure 5.1(a).

Example 5.4. Figure 5.6 shows the CRN of the DCSN in Figure 5.1(a). The DCSN has four basic subnets $\mathcal{N}_1^1, \mathcal{N}_1^2, \mathcal{N}_1^3$ and $\mathcal{N}_1^4$, in which $\mathcal{N}_1^1$ has common agents with $\mathcal{N}_1^2$ and $\mathcal{N}_1^3$, and $\mathcal{N}_1^4$ has common agents with $\mathcal{N}_1^2$ and $\mathcal{N}_1^3$. As a result, the CRN has four nodes, each of which represents a constraint, and the node representing $C_{1,2,3,4,9}^1$ is connected to those representing $C_{3,4,5,6}^2$ and $C_{8,9,10}^3$, while the node representing $C_{6,7,8}^4$ connected to those representing $C_{3,4,5,6}^2$ and $C_{8,9,10}^3$. In Figure 5.6, an edge connecting two constraints is labeled with the agent group overlap between their induced agent groups: The edge connecting $C_{1,2,3,4,9}^1$ and $C_{3,4,5,6}^2$ is labeled with $\{A_3, A_4\}$, the edge connecting $C_{6,7,8}^4$ and $C_{3,4,5,6}^2$ is labeled with $\{A_6\}$, and so on.

Before proceeding further, we review the concept of a cut-set [20] of a connected graph and the basic approach to generate all cut-sets of the graph. This is presented in Remark 5.2 below.

Remark 5.2. *In a connected graph $G = (V, E)$, a cut-set [20] is a set of edges $E' \subseteq E$ such that the removal of E' from G disconnects G and the removal of any strict subset of E' does not disconnect G . Since a cut-set E' always “cuts” G into two parts, it may be conveniently represented as (V_1, V_2) , where V_1 and V_2 are the sets of vertices belonging to these two parts. Let T be a spanning tree of G . Then a “fundamental” cut-set of G is defined as a cut-set that contains exactly one branch of T . Defining the ring sum operation $\oplus$ of two arbitrary sets A and B as $A \oplus B = (A \cup B) - (A \cap B)$, it has been shown that any cut-set of G has the form $E_1 \oplus E_2 \oplus \dots \oplus E_z$ that is not a union of edge-disjoint cut-sets, where $z \geq 2$ is arbitrary and $E_1, \dots, E_z$ are different fundamental cut-sets of G . Thus, a formal approach to generate all cut-sets of G is to (i) construct a spanning tree, (ii) generate the set of fundamental cut-sets for the spanning tree, and then (iii) properly combine these fundamental cut-sets to get a new cut-set.*

Observe now that enumerating all possible decompositions of a subnet $\mathcal{N}_r$ into two constraint-connected subnets can be done by enumerating all possible cut-sets of its CRN $\mathcal{CRN}_r$. Specifically, consider a cut-set $(\mathcal{C}_x, \mathcal{C}_y)$ that decomposes $\mathcal{CRN}_r$ into two parts, where $\mathcal{C}_x$ and $\mathcal{C}_y$ are the two disjoint sets of vertices of $\mathcal{CRN}_r$ belonging to these two parts. Write $\mathcal{N}_x \sim \mathcal{C}_x$ and $\mathcal{N}_y \sim \mathcal{C}_y$ to denote respectively that $\mathcal{N}_x$ and $\mathcal{N}_y$ are the component subnets induced by $\mathcal{C}_x$ and $\mathcal{C}_y$, namely $\mathcal{N}_x = \{(J_k, C_{J_k}^k) \mid C_{J_k}^k \in \mathcal{C}_x\}$ and $\mathcal{N}_y = \{(J_k, C_{J_k}^k) \mid C_{J_k}^k \in \mathcal{C}_y\}$. Then $\mathcal{N}_x$ and $\mathcal{N}_y$ are two constraint-connected component subnets decomposed from $\mathcal{N}_r$. Conversely, any decomposition of $\mathcal{N}_r$ into two constraint-connected component subnets $\mathcal{N}_x$ and $\mathcal{N}_y$ corresponds to a cut-set $(\mathcal{C}_x, \mathcal{C}_y)$ of $\mathcal{CRN}_r$, with $\mathcal{N}_x \sim \mathcal{C}_x$ and $\mathcal{N}_y \sim \mathcal{C}_y$.

From the foregoing observation, Procedure *GenerateANDORGraph* (Figure 5.7) details the steps to generate an AND/OR graph representation of conflict resolution plans for a given DCSN $\mathcal{N}$. If $\mathcal{N}$ is a basic subnet, the procedure simply returns an empty AND/OR graph (Step 1), otherwise it converts $\mathcal{N}$ to the a CRN $\mathcal{CRN}$, and computes *CutSets* as the set of all cut-sets of $\mathcal{CRN}$ (Step 2). In Step 3, the procedure uses the cut-sets to recursively construct the AND/OR graph representation of conflict resolution plans.

Based on the foregoing discussion, *GenerateANDORGraph* is correct and complete in the sense that it correctly generates, for a DCSN $\mathcal{N}$, an AND/OR graph that completely

Procedure *GenerateANDORGraph* ($\mathcal{N}$)

Output: An AND/OR graph $T_{\mathcal{N}} = (S_{\mathcal{N}}, H_{\mathcal{N}})$ of conflict resolution plans for $\mathcal{N}$, initialized with $S_{\mathcal{N}} = \emptyset$ and $H_{\mathcal{N}} = \emptyset$

begin

Step 1: If $\mathcal{N}$ contains only one basic subnet then return; otherwise, convert $\mathcal{N}$ into a $\mathcal{CRN} = (\mathcal{C}, \mathcal{R})$;

Step 2: Compute *CutSets* as the set of all cut-sets of $\mathcal{CRN}$;

Step 3 while *CutSets* $\neq \emptyset$ **do**

Step 3a Remove a cut-set $(\mathcal{C}_x, \mathcal{C}_y)$ from *CutSets*. Let $\mathcal{N}_x \sim \mathcal{C}_x$ and $\mathcal{N}_y \sim \mathcal{C}_y$;

Step 3b Add nodes and an edge to T : $S_{\mathcal{N}} = S_{\mathcal{N}} \cup \{\mathcal{N}_x, \mathcal{N}_y, \mathcal{N}_x \cup \mathcal{N}_y\}$,
 $H_{\mathcal{N}} \cup \{(\mathcal{N}_x \cup \mathcal{N}_y, \mathcal{N}_x, \mathcal{N}_y)\}$;

Step 3c For $r \in \{x, y\}$, *GenerateANDORGraph*($\mathcal{N}_r$);

end

Figure 5.7: Procedure *GenerateANDORGraph* for generating an AND/OR graph of conflict resolution plans for a DCSN.

encompasses all possible conflict resolution plans for $\mathcal{N}$.

The amount of computation involved depends on the number of basic subnets of the input DCSN and its connectivity structure, which both affect the number of cut-sets of the CRN of $\mathcal{N}$ and that of each of the CRN's of the successively decomposed subnets. In general, the more basic subnets the input DCSN has, and the more “connected” these basic subnets are, the higher the amount of computation incurred for *GenerateANDORGraph*. In the worst case when a DCSN with m basic subnets is strongly connected, the number of nodes required for AND/OR graph plan generation is $(2^m - 1)$, and the complexity of the procedure can be shown to be $O(2^m)$, an exponential order in m . However, often in practice, agents are involved in only some and not all the basic subnets in the DCSN. In such cases, the connectivity structure is simpler and so the computational complexity of *GenerateANDORGraph* is usually much lower than $O(2^m)$. In fact, in the opposite extreme of a weakly connected DCSN with m basic subnets, it enjoys polynomial complexity of $O(m^2)$ since the number of nodes required for AND/OR graph plan generation is $\frac{1}{2} \times m \times (m + 1)$ nodes.

In practice, based on some criterion, the cut-sets may be subjected to some acceptance tests in Step 3a, and only accepted cut-sets are passed on to Steps 3b and 3c. Such tests can be developed to generate only conflict resolution plans which satisfy some problem-

dependent conditions. For example, a multiagent coordination system can be organized into a network of standalone (or atomic) subnets where no two standalone subnets share the same inter-agent constraint tasks. These subnets may have their agents periodically entering the network when instantiated (say, by some underlying agent management system) to coordinate on some inter-agent constraint tasks during runtime, and exiting the system network when uninstantiated. To support this situation, we need to, at the outset, ensure that agents in individual standalone subnets can always ensure nonblockingness of their coordination tasks when the system network needs to be reconfigured³ with the entry or exit of some standalone nets. This has significant implications in generating and selecting conflict resolution plans when not all standalone subnets are basic. Given a DCSN of standalone subnets, we would need an AND/OR graph plan that must include only decompositions that have these subnets in the subnet in one or the other child node, whenever each is a part of the bigger subnet in the parent node. Executing such plans forward can guarantee the nonblocking reconfigurability of the resultant multiagent network.

Example 5.5. *We explain how the AND/OR graph in Figure 5.4 can be generated by applying `GenerateANDORGraph` to decompose the CRN in Figure 5.6.*

*For economy of notation, let e_1 , e_2 , e_3 and e_4 denote the four edge labels $\{A_3, A_4\}$, $\{A_9\}$, $\{A_8\}$ and $\{A_6\}$ of the CRN in Figure 5.6, respectively. Following, note that a spanning tree of the CRN in Figure 5.6 is a tree containing three branches e_1 , e_2 and e_3 . By Remark 5.2, the fundamental cut-sets of the CRN derived from this spanning tree are (e_1, e_4) , (e_2, e_4) and (e_3, e_4) . Then by performing the ring sum operation $\oplus$ on those fundamental cut-sets, we obtain the following possible cut-sets of the CRN: (e_1, e_2) , (e_1, e_4) , (e_2, e_3) , (e_3, e_4) , (e_1, e_3) , and (e_2, e_4) . These cut-sets, which are computed and stored in *CutSets* during the first execution of Step 2 of `GenerateANDORGraph`, correspond to the decomposition of $\mathcal{N}$ into the following pairs of constraint-connected subnets: $(\mathcal{N}_1^1, \mathcal{N}_3^{\{2,3,4\}})$, $(\mathcal{N}_1^2, \mathcal{N}_3^{\{1,3,4\}})$, $(\mathcal{N}_1^3, \mathcal{N}_3^{\{1,2,4\}})$, $(\mathcal{N}_1^4, \mathcal{N}_3^{\{1,2,3\}})$, $(\mathcal{N}_2^{\{1,3\}}, \mathcal{N}_2^{\{2,4\}})$, and $(\mathcal{N}_2^{\{1,2\}}, \mathcal{N}_2^{\{3,4\}})$.*

Then, after the first execution of Steps 3a and 3b, `GenerateANDORGraph` constructs a subgraph of the AND/OR graph in Figure 5.4, which includes 13 nodes representing 13 subnets $\mathcal{N}$, $\mathcal{N}_1^1$, $\mathcal{N}_1^2$, $\mathcal{N}_1^3$, $\mathcal{N}_1^4$, $\mathcal{N}_2^{\{1,3\}}$, $\mathcal{N}_2^{\{2,4\}}$, $\mathcal{N}_2^{\{1,2\}}$, $\mathcal{N}_2^{\{3,4\}}$, $\mathcal{N}_3^{\{2,3,4\}}$, $\mathcal{N}_3^{\{1,3,4\}}$, $\mathcal{N}_3^{\{1,2,3\}}$ and $\mathcal{N}_3^{\{1,2,4\}}$, and 6 edges representing the above six decompositions of $\mathcal{N}$ [see Figure 5.8].

³In this context, reconfigurability means loading or unloading of agent and CM models as appropriate by some underlying agent management system.

Figure 5.8: A subgraph of the AND/OR graph in Figure 5.4. This subgraph is generated by the first call to *GenerateANDORGraph* to decompose the CRN in Figure 5.6.

*In subsequent recursive calls of *GenerateANDORGraph*, possible decompositions of the subnets $\mathcal{N}_2^{\{1,3\}}$, $\mathcal{N}_2^{\{2,4\}}$, $\mathcal{N}_2^{\{1,2\}}$, $\mathcal{N}_2^{\{3,4\}}$, $\mathcal{N}_3^{\{2,3,4\}}$, $\mathcal{N}_3^{\{1,3,4\}}$, $\mathcal{N}_3^{\{1,2,3\}}$ and $\mathcal{N}_3^{\{1,2,4\}}$ are enumerated in a similar manner, and the whole AND/OR graph of Figure 5.4 can eventually be generated.*

5.4 Selection of Optimal Conflict Resolution Plan

We have shown that for a given DCSN, an AND/OR graph can be generated to encompass all possible conflict resolution plans, and hence provide the search space for the selection of an optimal conflict resolution plan. To automate the design of a network of coordinating discrete-event agents, in this section, we develop an algorithm that performs heuristic search for an optimal plan over an AND/OR graph representation of conflict resolution plans.

5.4.1 General Heuristic Search for Optimal Conflict Resolution Plan

To select an optimal conflict resolution plan for a given DCSN $\mathcal{N}$, in addition to the ability to traverse the space of all possible conflict resolution plans provided by $T_{\mathcal{N}}$, there is a need for an optimization metric to access, or rank, the quality of individual plans.

Since a conflict resolution plan is a tree in $T_{\mathcal{N}}$ that starts from $n_{root} \in S_{\mathcal{N}}$ and terminates at $\Theta_{leaf} \subseteq S_{\mathcal{N}}$, an optimization metric for plan selection is simply a real function $F : Trees(n_{root}, \Theta_{leaf}) \rightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. We assume a minimization problem, and interpret a better conflict resolution plan as a plan with lower F -value. Thus if $tree_1, tree_2 \in Trees(n_{root}, \Theta_{leaf})$ with $F(tree_1) < F(tree_2)$, then the conflict resolution plan $tree_1$ is preferable to $tree_2$.

Although an optimal conflict resolution plan for a DCSN $\mathcal{N}$ may be found by enumerating all trees in $Trees(n_{root}, \Theta_{leaf})$ and evaluating them using an optimization metric, search techniques can be used to avoid this complete enumeration and improve the efficiency of the selection process. In what follows, we present such an algorithm to search for an optimal conflict resolution plan over $T_{\mathcal{N}}$. The algorithm is based on the general heuristic A^* search algorithm [77], and entails incorporating a heuristic defined on $Trees(n_{root}, -)$, the set of partial trees in $T_{\mathcal{N}}$.

Definition 5.6. A heuristic (for an optimization metric $F : Trees(n_{root}, \Theta_{leaf}) \rightarrow \mathbb{R}$) is a real function $H : Trees(n_{root}, -) \rightarrow \mathbb{R}$ such that $(\forall tree \in Trees(n_{root}, \Theta_{leaf})) H(tree) = F(tree)$.

Given a partial tree $ptree \in Trees(n_{root}, -)$, the heuristic value $H(ptree)$ shall be used in our algorithm as an estimation of the F -value of the best conflict resolution plan $tree \in Trees(n_{root}, \Theta_{leaf})$ that encompasses the partial plan $ptree$. Heuristic H is said to be admissible if the H -value of an arbitrary partial tree always underestimates the F -value of any complete tree encompassing it, as formalized in Definition 5.7.

Definition 5.7. A heuristic $H : Trees(n_{root}, -) \rightarrow \mathbb{R}$ is said to be admissible (for an optimization metric $F : Trees(n_{root}, \Theta_{leaf}) \rightarrow \mathbb{R}$) if, for an arbitrary partial $ptree \in Trees(n_{root}, -)$ and every complete tree $tree \in Trees(n_{root}, \Theta_{leaf})$ for which $ptree$ is a subgraph of, $H(ptree) \leq F(tree)$.

We can now formally present our plan selection algorithm. Given an admissible heuristic H for some optimization metric F , Procedure *HeuristicPlanSelection* details the

steps to select an optimal conflict resolution plan for a DCSN $\mathcal{N}$ from the AND/OR graph $T_{\mathcal{N}}$ (see Figure 5.9). The procedure returns a complete tree of $T_{\mathcal{N}}$ with the lowest F -value, and is thus an optimal conflict resolution plan for $\mathcal{N}$.

Procedure *HeuristicPlanSelection* ($T_{\mathcal{N}}, H$)

Input: AND/OR graph of conflict resolution plans $T_{\mathcal{N}} = (S_{\mathcal{N}}, H_{\mathcal{N}})$ for DCSN $\mathcal{N}$ and an admissible heuristic $H : \text{Trees}(n_{\text{root}}, -) \rightarrow \mathbb{R}$

Output: A tree in $\text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}})$ with the lowest F -value, which is an optimal conflict resolution plan for $\mathcal{N}$

begin

Step 1: Create a partial tree *ptree* which contains only the root node n_{root} ;

Step 2: Compute the heuristic value $H(\text{ptree})$ and put *ptree* into a queue Q ;

Step 3: **while** $Q \neq \emptyset$ **do**

Step 3a Extract from Q a tree with the lowest H -value and call it *ptree*;

Step 3b If *ptree* $\in \text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}})$, return it as a solution;

Step 3c Otherwise, select a terminal node n of *ptree* that is not in Θ_{leaf} ;

Step 3d **for each edge** $(n, (n_1, n_2)) \in H_{\mathcal{N}}$ **do**

Step 3d1 Create a new partial tree *ntree* whose nodes are those of *ptree* plus n_1 and n_2 , and whose edges are those of *ptree* plus $(n, (n_1, n_2))$;

Step 3d2 Compute $H(\text{ntree})$ and put *ntree* into Q ;

end

Figure 5.9: Procedure *HeuristicPlanSelection* for selecting an optimal conflict resolution plan for a DCSN $\mathcal{N}$.

HeuristicPlanSelection maintains a priority queue Q that contains partial trees of $T_{\mathcal{N}}$, ranked by their heuristic H -value. In Steps 1 and 2, a partial tree that contains only the root node n_{root} is created and put into Q . Each time through the while loop of Step 3, a tree with the lowest H -value is extracted from Q (Step 3a), and is returned as a solution if it is a complete tree (Step 3b), or otherwise expanded (Steps 3c and 3d). The expanded trees are then put into Q for further examination (Step 3d2). The correctness of *HeuristicPlanSelection* is formally stated in Theorem 5.1.

Theorem 5.1. *If H is an admissible heuristic for F , then *HeuristicPlanSelection* returns a complete tree in $\text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}})$ with the lowest F -value.*

Proof. Let the lowest F -value be F^* . By contradiction, assume that *HeuristicPlanSelection* returns *tree* $\in \text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}})$ with $F(\text{tree}) > F^*$. Since *tree* $\in \text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}})$,

we also have $H(tree) = F(tree) > F^*$.

Consider a partial tree $ptree$ that is a subgraph of an optimal plan $tree^* \in Trees(n_{root}, \Theta_{leaf})$ with $H(tree^*) = F(tree^*) = F^*$ and that is contained in Q before $tree$ is extracted from Q (there must always be such trees since an optimal solution always exists). Then, since H is an admissible heuristic, we have $H(ptree) \leq F^*$.

We now have $H(ptree) \leq F^* < H(tree)$. Since in Step 3a, *HeuristicPlanSelection* always extracts from Q a tree with the lowest H -value, it follows that $tree$ will not be extracted from Q before $ptree$ is. And when $ptree$ is extracted from Q , it will be expanded in Steps 3c and 3d, and eventually becomes $tree^*$ before $tree$ can ever be extracted from Q . The reason is that $H(tree^*) = F^* < H(tree)$ and H is an admissible heuristic, meaning that any subgraph of $tree^*$ that is expanded from $ptree$ in Steps 3c and 3d will have its H -value smaller than that of $tree$ and therefore, extracted from Q before $tree$. Finally, if $tree^*$ is ever be extracted from Q in Step 3a, it will be returned as a solution by *HeuristicPlanSelection*. In other words, $tree$ will never be returned by *HeuristicPlanSelection*, contradicting our initial assumption. Hence the proof. $\square$

5.4.2 Reducing Execution Time Through Parallel Compositions of Subnets

We now introduce a criterion to evaluate and select conflict resolution plans. The criterion is to maximize the simultaneous execution of operations for subnet composition. An optimization metric to rank the plans quantitatively based on this criterion is formulated, and an admissible heuristic of this metric is designed for *HeuristicPlanSelection*. Importantly, the selected plan provides the opportunity to maximize the parallel use of available computing resources in simultaneous subnet compositions, and can often be executed in minimal total execution time.

Over a conflict resolution planning tree in the AND/OR graph T_N , the measure of simultaneity of execution supported in the operations of subnet composition can be quantified by the depth of the tree, defined recursively as follows.

$$(\forall tree \in Trees(-, -))$$

$$Depth(tree) = \begin{cases} 0, & \text{if } tree = (n_I); \\ 1 + \max(Depth(tree_1), Depth(tree_2)), & \text{if } tree = (n_I, h, tree_1, tree_2). \end{cases}$$

Using this measure, the optimization metric is defined as follows.

$$\begin{aligned} F_p : \text{Trees}(n_{\text{root}}, \Theta_{\text{leaf}}) &\rightarrow \mathbb{N} \\ \text{tree} &\mapsto \text{Depth}(\text{tree}). \end{aligned}$$

where $\mathbb{N} = \{0, 1, 2, \dots\}$ is the set of natural numbers.

Example 5.6. *The depth of the tree in Figure 5.5(c) is 2, while that of the trees in Figures 5.5(a) and 5.5(b) is 3. It can be seen that while there are three composition operations required by the conflict resolution plan tree in Figure 5.5(c), the first two operations, i.e., those of composing $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$ and composing $\mathcal{N}_1^3$ and $\mathcal{N}_1^4$, can be performed simultaneously. Therefore, if there are two computational resources that can operate in parallel, the plan can be completed in two sequential steps, with the first step is to simultaneously compose $\mathcal{N}_1^1$ and $\mathcal{N}_1^2$, and $\mathcal{N}_1^3$ and $\mathcal{N}_1^4$, and the second step is to compose $\mathcal{N}_2^{\{1,2\}}$ and $\mathcal{N}_2^{\{3,4\}}$. Each of the trees in Figures 5.5(a) and 5.5(b) also has three composition operations. However, these operations have to be performed sequentially. Thus, no matter how many computational resources we have, each of these plans requires three sequential steps to complete.*

We now design an admissible heuristic H_p for F_p . Recall that the set of nodes of $T_{\mathcal{N}}$ is $S_{\mathcal{N}} = \{\mathcal{N}_r \subseteq \mathcal{N} \mid \mathcal{N}_r \text{ is constraint-connected}\}$, namely, each node of $T_{\mathcal{N}}$ represents a constraint-connected subnet of $\mathcal{N}$. For each $n \in S_{\mathcal{N}}$, let $\text{NumBasicSubnet}(n)$ denote the number of basic subnets in the constraint-connected subnet represented by node n .

Let H'_p be a real function on $\text{Trees}(-, -)$, defined recursively as follows.

$$H'_p(\text{tree}) = \begin{cases} \log_2(\text{NumBasicSubnet}(n_I)), & \text{if } \text{tree} = (n_I); \\ 1 + \max(H'_p(\text{tree}_1), H'_p(\text{tree}_2)), & \text{if } \text{tree} = (n_I, h, \text{tree}_1, \text{tree}_2). \end{cases}$$

Then an admissible heuristic H_p for F_p can be specified as

$$\begin{aligned} H_p : \text{Trees}(n_{\text{root}}, -) &\rightarrow \mathbb{R} \\ \text{ptree} &\mapsto H'_p(\text{ptree}). \end{aligned}$$

The proof of Lemma 5.1 unravels the intuition behind the design of H_p .

Lemma 5.1. *H_p is an admissible heuristic for F_p .*

Proof. To prove this lemma, we have to prove that the following two conditions hold:

- (i) If $tree \in Trees(n_{root}, \Theta_{leaf})$ then $H_p(tree) = F_p(tree)$, and
- (ii) $(\forall ptree \in Trees(n_{root}, -)) (\forall tree \in Trees(n_{root}, \Theta_{leaf})) (ptree \text{ is a subgraph of } tree) \text{ implies } H_p(ptree) \leq F_p(tree)$.

To prove (i), we shall show that if $tree \in Trees(n_{root}, \Theta_{leaf})$ then $H'_p(tree) = Dept(tree)$. This can be done by a simple induction on the depth of trees as follows.

- *Base:* First, since $\log_2(1) = 0$, any tree that contains only one node representing a basic subnet of $\mathcal{N}$ has both its H'_p -value and its depth equal to 0.
- *Inductive Hypothesis:* Now, assume that any tree whose depth smaller than or equal to an integer $d \geq 0$ and whose terminal nodes are all in Θ_{leaf} has its depth equal to its H'_p -value.

We then show that any tree with depth $d + 1$ and with all terminal nodes in Θ_{leaf} will also have its depth equal to its H'_p -value as follows.

- Let $tree = (n_I, h, tree_1, tree_2)$ be a tree with $Dept(tree) = d + 1$ and with every terminal node in Θ_{leaf} . Since $Dept(tree) = 1 + \max(Dept(tree_1), Dept(tree_2))$, $\max(Dept(tree_1), Dept(tree_2)) = d$.
- It follows that both the depths of $tree_1$ and $tree_2$ are equal to or smaller than d . Furthermore, every terminal node of $tree_1$ and $tree_2$ is in Θ_{leaf} . Therefore, by the inductive hypothesis, $Depth(tree_1) = H'_p(tree_1)$ and $Depth(tree_2) = H'_p(tree_2)$.
- It then follows that $\max(Dept(tree_1), Dept(tree_2)) = \max(H'_p(tree_1), H'_p(tree_2))$, or $Dept(tree) = 1 + \max(H'_p(tree_1), H'_p(tree_2))$. By the definition of H'_p , therefore, $Dept(tree) = H'_p(tree)$.
- Thus, by induction, if $tree \in Trees(n_{root}, \Theta_{leaf})$ then $H'_p(tree) = Depth(tree)$. By the definitions of F_p and H_p , it then follows that if $tree \in Trees(n_{root}, \Theta_{leaf})$, $H_p(tree) = F_p(tree)$.

To prove (ii), consider a partial tree $ptree$ in $T_{\mathcal{N}}$ that starts from n_{root} and terminates at a set of nodes that are not necessarily leaf nodes.

Consider a terminal node n_t of $ptree$ that is not a leaf node, which represents a constraint-connected subnet of $\mathcal{N}$. Let $stree$ be an arbitrary tree that starts at n_t and terminates at a subset of leaf nodes. $stree$ is then a sub-plan for $\mathcal{N}$, namely, a plan to synthesize the subnet represented by n_t . The depth of $stree$ must then be equal to or greater than $\log_2(NumBasicSubnet(n_t))$, since to synthesize the subnet represented by n_t , we need to successively compose two different subnets of it at a time.

Since the depth of a tree starting from an arbitrary terminal node n_t of $ptree$ and terminating at Θ_{leaf} is equal to or greater than $\log_2(NumBasicSubnet(n_t))$, by the recursive definitions of H'_p , it follows that the depth of any tree in $Trees(n_{root}, \Theta_{leaf})$ that encompasses $ptree$ as a subgraph is equal to or greater than $H_p(ptree)$. In other words, H_p is an admissible heuristic for F_p . $\square$

Thus, by Lemma 5.1, Heuristic H_p can be incorporated into *HeuristicPlanSelection* for the selection of a plan with the lowest F_p value.

Example 5.7. We apply the algorithms developed in this chapter to construct a conflict resolution plan for the manufacturing transfer line system presented in Chapter 4. For ease of reference, the DCSN and CRN for this system are redrawn in Figure 5.10. The DCSN is a strongly connected DCSN with four basic subnets $\mathcal{N}_1^1 = (\{1, 2\}, E_{\{1,2\}}^1)$, $\mathcal{N}_1^2 = (\{1, 2\}, E_{\{1,2\}}^2)$, $\mathcal{N}_1^3 = (\{1, 3\}, B_{\{1,3\}}^3)$ and $\mathcal{N}_1^4 = (\{2, 3\}, B_{\{2,3\}}^4)$.

Figure 5.10: The DCSN and CRN for the manufacturing transfer line system presented in Chapter 4.

To construct a conflict resolution plan, we first apply *GenerateANDORGraph* to decompose the CRN in Figure 5.10(b) and generate the AND/OR graph plan. The generated AND/OR graph is shown in Figure 5.11.

Figure 5.11: The AND/OR graph of conflict resolution plans for the manufacturing transfer line system.

Then we apply *HeuristicPlanSelection* using the heuristic H_p to select an optimal conflict resolution plan from the generated AND/OR graph, namely, one that allows maximal simultaneity in the execution of subnet composition operations. The selected conflict resolution plan is shown in Figure 5.12. This plan can be executed in two steps: (i) In step 1, compose N_1^1 and N_1^2 to form $N_2^{\{1,2\}}$, as well as N_1^3 and N_1^4 to form $N_2^{\{3,4\}}$, using *DeconflictBasicSubnet*; and (ii) in step 2, compose $N_2^{\{1,2\}}$ and $N_2^{\{3,4\}}$ to form N using *DeconflictSubnet*. Note that the first step requires parallel execution of two subnet composition operations. This plan was the one that we used in Example 4.3 in Chapter 4 for conflict resolution.

Figure 5.12: A conflict resolution plan for the manufacturing transfer line system.

5.5 Chapter Summary

In this chapter, we have presented a compact representation of conflict resolution plans for a given DCSN using AND/OR graphs, and developed an algorithm to generate an AND/OR graph representation of conflict resolution plans. That the algorithm can be easily extended to support problem-dependent planning such as for multiagent nonblocking reconfigurability has also been discussed. Importantly, the AND/OR graph representation provides an efficient structure to traverse the space of possible conflict resolution plans, and can be used as the basis for selecting an optimal plan. In this direction, a heuristic search algorithm for selecting an optimal conflict resolution plan over an AND/OR graph representation has been developed and discussed. A metric to rank these plans based on the optimization criterion of maximal execution simultaneity of subnet compositions is formulated, and an admissible heuristic of this optimization metric is presented for the search algorithm.

To summarize on the complexity issue, the total complexity of the compositional synthesis approach depends both on the complexities of constructing CM's and deconflicting CM's and on the number of nodes in the selected AND/OR graph plan (since each node entails the synthesis of CM's or deconflicting CM's for the agents concerned). The former depends on the number and sizes of the agents and constraints for a given coordination problem (specified as a DCSN). The latter depends on the total number of nodes in the

generated AND/OR graph representation of confliction resolution plans, which in turn is greatly affected by the connectivity structure of the input DCSN.

Together with the basic CM synthesis procedures in Chapter 3 and the conflict resolution algorithms in Chapter 4, this chapter brings to fruition a complete algorithmic framework for automated coordination design that addresses the networked coordination problem. The framework developed benefits from the comprehensibility of DCSN and the representational power of AND/OR graphs to more effectively manage the specification network and synthesis complexity of automated coordination design. Importantly, we can now easily specify the inter-agent networking structure of a coordination design problem using the DCSN formalism, and address the problem systematically using the proposed compositional approach supported by the framework.

Chapter 6

ON-LINE AGENT COORDINATION PROBLEM

6.1 Chapter Overview

In Chapter 3, we formulated and addressed the fundamental discrete-event multiagent coordination problem. The solution approach entails computing the required coordinating actions for all anticipated interacting situations and storing them as a CM for each agent. At run-time, the correct coordinating actions are then simply retrieved from the CM's and enforced accordingly by the agents. This off-line synthesis approach has both merits and demerits. On the one hand, the approach is suitable for applications where coordinating decisions have to be made as fast as possible during the agents' run-time interaction. On the other hand, the off-line planning associated with the approach may be too expensive for applications where the number of anticipated interacting situations is too large, or when off-line planning time is limited. To complement the off-line approach, in this chapter, we develop an on-line coordination synthesis approach. The on-line approach enables agents to compute coordinating actions only in response to situational changes. A significant merit is that it avoids altogether the off-line construction of CM's as with the off-line approach. For a clear exposition to the new concepts and ideas introduced in this chapter, we present and explain the theoretical results for two coordinating agents. We then show how to extend the results to multiple agents

Within our discrete-event coordination framework, we introduce and study a novel pred-

icate coordination problem as that of developing distributed on-line strategies for agents to interact and communicate continually between themselves to conform to a given global predicate [21] specifying an inter-agent constraint. In essence, a predicate P_c is an inter-agent constraint of the fundamental safety type, specifying that no bad states can ever be visited during multiagent interaction. The problem is shown to be solvable in some autonomy permitted setting (Theorem 6.1) for coordinable predicates (Definition 6.3) not less restrictive than a given P_c on the state space. The solution basis developed in this chapter is an optimal policy (Theorem 6.2) by which agents can interact and communicate to guarantee *coordination quality*, in that the executing event sequences are transitions of their composite states that remain confined to the largest feasible state subset of the state set defined by the given predicate, and can reach all the states in this state subset. Formally, this feasible state subset corresponds to the supremal coordinable predicate (denoted as P_c^{sup}) of the given predicate P_c (Definition 6.4).

To implement the optimal coordination policy strategically, two cooperative agents A_1 and A_2 can coordinate as follows. Upon executing a local event or receiving a state information update, each agent would always take the action of enabling every event (defined at its current local state) provided it does not (eventually) lead the coordinated state space out of the feasible subset characterized by P_c^{sup} , and disabling the event otherwise. Besides, each agent (say A_1) would have to decide whether or not to send its current local state to the other agent A_2 , to provide the latter with sufficient information for computing and updating its coordinating actions. Two decision strategies are formulated in this chapter:

(1) In the first solution strategy, agent A_1 would always send its updated local state to agent A_2 whenever it enters a new state.

(2) In the second, agent A_1 would only do so when it detects that agent A_2 might no longer be coordination-ready (to correctly maintain P_c^{sup}) as the latter's coordinating actions might have been invalidated. Agent A_1 detects the coordination-readiness (Definition 6.8) of agent A_2 by checking a set of local conditions for co-stability, a new coordination concept formulated in Definition 6.9.

Unlike the first strategy proposed which entails full communication, the second strategy, importantly, can significantly reduce the communication bandwidth, as demonstrated by experimental evaluation, while still maintaining coordination quality (Theorem 6.3).

The rest of this chapter is organized as follows. We formally state the predicate coordination problem in Section 6.2, and formulate the optimal coordination policy in Section 6.3.

In Section 6.4, the two novel concepts of coordination-readiness and co-stability are presented, based on which two on-line coordination strategies are proposed, including one that can achieve significant savings in communication bandwidth. We then present an example to illustrate the potential applicability of our problem in Section 6.5, and demonstrate the effectiveness in bandwidth reduction of the developed procedure with experimental evaluation in Section 6.6. Finally, Section 6.8 concludes this chapter.

6.2 Problem Formulation

Consider a system of two discrete-event agents modeled by the respective reachable automata $A_i = (X^{A_i}, \Sigma^{A_i}, \delta^{A_i}, x_0^{A_i}, X_m^{A_i})$ ($i \in \{1, 2\}$), where $\Sigma^{A_1} \cap \Sigma^{A_2} = \emptyset$. The event set Σ^{A_i} of agent A_i is partitioned into the controllable set $\Sigma_c^{A_i}$ and the uncontrollable set $\Sigma_{uc}^{A_i}$. To reiterate, in our framework, automaton A_i is viewed as the local plan of the respective agent, encompassing all possible local ways to achieve the agent's own goal; and an uncontrollable event in $\Sigma_{uc}^{A_i}$ is inherently autonomous and can be executed solely at the free will of the agent. As a rule, an event is pre-specified as uncontrollable if it is critical to the owner agent that disabling the event and limiting its autonomy is undesirable, expensive or even impossible.

Let $A = A_1 \parallel A_2$ model the multiagent system of A_1 and A_2 freely interacting with $\Sigma_c^A = \Sigma_c^{A_1} \cup \Sigma_c^{A_2}$ and $\Sigma_{uc}^A = \Sigma_{uc}^{A_1} \cup \Sigma_{uc}^{A_2}$. The two agents A_1 and A_2 would need to coordinate between themselves if, due to system needs or limitations, the execution of some event sequences in $L(A)$ is undesirable and must be prevented. In other words, their coordinating actions would need to satisfy an inter-agent constraint that excludes undesirable event sequences.

Consider an inter-agent constraint specified by an automaton C (representing the language $L(C)$). Then the coordination problem becomes that of A_1 and A_2 interacting and communicating to conform to C , such that none of the sequences in the *bad* sequence set $L(A) - L(C)$ can ever be generated during multiagent interaction. The automaton C is essentially an inter-agent constraint of the safety type, specifying that *nothing bad can happen*.

In this chapter, we focus on a C which is a nonempty sub-automaton of A , i.e., $X^C \subseteq X^A$, $x_0^C = x_0^A$, and δ^C is a restriction of δ^A on $\Sigma^A \times X^C$. Such an automaton C can be equivalently represented by a predicate defined on the set X^A . In essence, a predicate P

defined on X^A is a function $P : X^A \rightarrow \{0, 1\}$. For automaton C , the equivalent predicate P_c is defined as follows:

$$(\forall x \in X^A) P_c(x) = \begin{cases} 1, & \text{if } x \in X^C; \\ 0, & \text{otherwise.} \end{cases}$$

Henceforth, such a constraint automaton C and its equivalent predicate P_c can be used interchangeably. For a state $x \in X^A$, we say x satisfies P_c , and write $x \models P_c$, if $P_c(x) = 1$. For two predicates P_1 and P_2 defined on X^A , we say that P_1 is not less restrictive than P_2 , denoted by $P_1 \preceq P_2$, if $(\forall x \in X^A)(x \models P_1 \Rightarrow x \models P_2)$.

In addressing what can now be called a predicate coordination problem, P_c is an inter-agent constraint of the fundamental safety type, specifying that no states in the bad state set $X^A - X^C$ can ever be visited during multiagent interaction.

The coordinating actions for a pair of agents are governed by a coordination policy, formally defined as follows.

Definition 6.1. A coordination policy $\pi_{\langle A_1, A_2 \rangle}$ is a pair of agent policies $\langle \pi_{A_1}, \pi_{A_2} \rangle$, where π_{A_i} for agent A_i is a mapping from a state $x \in X^A$ to an event subset of Σ^{A_i} , such that $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$.

Thus π_{A_i} attaches to each state x of X^A a subset of Σ^{A_i} that contains $\Sigma_{uc}^{A_i} \cap \Sigma^A(x)$ - the subset of uncontrollable events of A_i that are defined at x . Using coordination policy $\pi_{\langle A_1, A_2 \rangle}$, agent A_i , upon observing the system state $x \in X^A$, enables every event $\sigma \in \pi_{A_i}(x)$, and disables all other events. The condition $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$ characterizes the fact that uncontrollable events can never be disabled.

Definition 6.2. The system of agents A_1 and A_2 interacting using a coordination policy $\pi_{\langle A_1, A_2 \rangle}$ is a (discrete-event) system represented by an automaton $A_\pi = \text{Reach}(X^A, \Sigma^A, \delta_\pi^A, x_0^A, X_m^A)^1$, where $(\forall \sigma \in \Sigma^{A_i})(\forall x \in X^A) \delta_\pi^A(\sigma, x) = \delta^A(\sigma, x)$ if $\delta^A(\sigma, x)!$ and $\sigma \in \pi_{A_i}(x)$, and is undefined otherwise.

Since we are only interested in the reachable part of the coordinated system, the condition $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$ of a coordination policy $\pi_{\langle A_1, A_2 \rangle}$ in Definition 6.1 can be relaxed to $(\forall x \in X^{A_\pi}) \pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$.

¹Recall from Chapter 2 that, for an automaton A , $\text{Reach}(A)$ denotes a reachable automaton computed from A by deleting every state that is not reachable from x_0^A .

Definition 6.3. A predicate P defined on X^A is said to be coordinable if, for every $x \in X^A$ satisfying P , the following conditions are satisfied:

1. $(\exists s \in (\Sigma^A)^*)[\delta^A(s, x_0^A) = x \text{ and } (\forall w \leq s)\delta^A(w, x_0^A) \models P]$.
2. $(\forall \sigma \in \Sigma_{uc}^A)[\delta^A(\sigma, x)! \Rightarrow \delta^A(\sigma, x) \models P]$.

By Definition 6.3, coordinability asserts that if x satisfies P then (i) x is reachable from x_0^A via a sequence of states satisfying P [Condition (1)], and (ii) if $\sigma \in \Sigma_{uc}^A$ and $\delta^A(\sigma, x)!$, then $\delta^A(\sigma, x)$ satisfies P [Condition (2)].²

We can now present the first main result of this chapter.

Theorem 6.1. Let C be a nonempty sub-automaton of A . Then there exists a coordination policy $\pi_{\langle A_1, A_2 \rangle}$ such that $A_\pi = C$ if and only if P_c is coordinable.

Proof. (If) Let $\pi_{\langle A_1, A_2 \rangle}$ be a coordination policy with each agent policy π_{A_i} given as: $(\forall x \in X^A)\pi_{A_i}(x) = \Sigma^C(x) \cap \Sigma^{A_i}$. Since P_c is coordinable, by Condition (2) of Definition 6.3, $(\forall x \in X^A)\pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$. Moreover, by Condition (1) of Definition 6.3, C is a reachable automaton. Hence it follows that $A_\pi = C$.

(Only If) Let $\pi_{\langle A_1, A_2 \rangle}$ be a coordination policy with $A_\pi = C$. Since A_π is a reachable automaton, P_c trivially satisfies Condition (1) of Definition 6.3. And, since $(\forall x \in X^{A_\pi})\pi_{A_i}(x) \supseteq \Sigma_{uc}^{A_i} \cap \Sigma^A(x)$, it follows that P_c satisfies Condition (2) of Definition 6.3. Hence P_c is coordinable. $\square$

It can be shown that the set of coordinable predicates that are not less restrictive than P_c is nonempty and closed under arbitrary predicate disjunctions, and so its supremal element exists. Formally, the supremal coordinable predicate P_c^{sup} of a given predicate P_c is defined as follows.

Definition 6.4. Given a predicate P_c defined on the system state space X^A . The supremal coordinable predicate of P_c , denoted by P_c^{sup} , is the unique predicate defined on X^A which satisfies the following properties:

1. $P_c^{sup} \preceq P_c$.

²The concept of a coordinable predicate given herein is mathematically equivalent to that of a controllable predicate in supervisory control theory [101]. However, as explained in [80] and in Chapter 3, supervisory control and multiagent coordination are conceptually different problems.

2. P_c^{sup} is coordinable.
3. $(\forall P \preceq P_c)[(P \text{ is coordinable}) \Rightarrow (P \preceq P_c^{sup})]$.

Let C^{sup} denote the equivalent automaton of P_c^{sup} . The predicate coordination problem can now be formally stated as follows.

Problem 6.1. *Given a predicate constraint P_c defined on the system state space X^A , construct the (unique) optimal coordination policy $\pi_{\langle A_1, A_2 \rangle}$ such that $A_\pi = C^{sup}$.*

The solution policy for Problem 6.1 is said to be optimal since the agents implementing it do not disable their own controllable events unless not doing so may eventually lead the coordinated state space out of the state subset satisfying P_c . Therefore, the policy enables the agents to visit as many states in X^C as possible, and have maximal autonomy over their own actions.

Note that (i) $C^{sup} = C$ if P_c is coordinable, and (ii) P_c^{sup} can be a false predicate, i.e., $(\forall x \in X^A) P_c^{sup}(x) = 0$; in this case, C^{sup} is an empty automaton, and Problem 6.1 has no solution.

6.3 Coordination Synthesis

Definition 6.5. *A state $x \in X^A$ is said to be P_c -safe if $(\forall s \in (\Sigma_{uc}^A)^*)[\delta^A(s, x)! \Rightarrow \delta^A(s, x) \models P_c]$. Otherwise, x is said to be P_c -unsafe.*

Thus a state x is P_c -safe if every state reachable from x via a string of uncontrollable events satisfies P_c . Following, it can be shown that C^{sup} is an empty automaton if and only if the initial system state x_0^A is P_c -unsafe. In synthesizing the solution for Problem 6.1, we shall henceforth assume that x_0^A is P_c -safe.

Given a state $x \in X^A$, the following *CheckSafety* procedure returns *true* if x is P_c -safe, and *false* otherwise.

Procedure *CheckSafety* ($x \in X^A$)

```

begin
  | if Safety[ $x$ ]  $\neq$  NIL then Return Safety[ $x$ ];
  | Return BFSChecking( $x$ );
end

```

CheckSafety determines the safety value of a given system state x by performing a search over part of or the entire system model $A = A_1 \parallel A_2$, and possibly also using stored results from prior (step) computations. It maintains a global logic variable *Safety* to store the safety value of every state in X^A . *Safety*[x] is true if x is P_c -safe, false if x is P_c -unsafe, and *NIL* if the safety value of x has not been determined. If *Safety*[x] $\neq$ *NIL*, i.e., the safety value of x has been computed in prior computations, the procedure simply returns the stored value. Otherwise, it invokes a procedure called *BFSChecking* to determine the safety value of x .

Procedure *BFSChecking* ($x \in X^A$)

```

begin
  Expanded  $\leftarrow \{\}$ ; Q  $\leftarrow \{x\}$ ; father[ $x$ ]  $\leftarrow$  NIL;
  while Q  $\neq \emptyset$  do
     $u \leftarrow$  the head element of Q; Q  $\leftarrow$  Q  $- \{u\}$ ;
    if  $u \in$  Expanded then continue;
    Expanded  $\leftarrow$  Expanded  $\cup \{u\}$ ;
    foreach  $\sigma \in \Sigma^A(u) \cap \Sigma_{uc}^A$  do
       $v \leftarrow \delta^A(\sigma, u)$ ;
      if Safety[ $v$ ]  $==$  false or  $v \not\models P_c$  then
        while  $v \neq$  NIL do
           $\lfloor$  Safety[ $v$ ]  $\leftarrow$  false;  $v \leftarrow$  father[ $v$ ];
           $\lfloor$  Return false;
        if Safety[ $v$ ]  $==$  NIL and  $v \notin$  Expanded then
           $\lfloor$  father[ $v$ ]  $\leftarrow$   $u$ ; Q  $\leftarrow$  Q  $\cup \{v\}$ ;
      foreach  $u \in$  Expanded do
         $\lfloor$  Safety[ $u$ ]  $\leftarrow$  true;
    Return true;
end

```

BFSChecking builds a Σ_{uc}^A -tree rooted at x by expanding all the consecutive $(\Sigma_{uc}^A)^*$ -successors of x in a breadth-first fashion. It maintains several data structures for this expansion process: A first-in, first-out queue *Q* to manage the set of states to be expanded next, a queue *Expanded* to store every state that has already been expanded, and, for every expanded state u , a variable *father*[u] to store its predecessor. By Definition 6.5, *BFSChecking* determines the safety value of x as follows. If, during the expansion pro-

cess, there is some $(\Sigma_{uc}^A)^*$ -successor v of x that does not satisfy P_c , or has already been shown to be P_c -unsafe in prior computations, then x and all of its $(\Sigma_{uc}^A)^*$ -successors in the path leading x to v are P_c -unsafe. In this case, the procedure simply stores the P_c -unsafe values of these states in variable *Safety* and returns *false*. Otherwise, if the Σ_{uc}^A -tree rooted at x is expanded fully without encountering any P_c -unsafe state, then x and all its consecutive $(\Sigma_{uc}^A)^*$ -successors are P_c -safe. Thus, the procedure simply stores the P_c -safe values of every expanded state $u \in Expanded$ in variable *Safety*, and returns *true*. Using a breadth-first search, procedure *CheckSafety* has worst-case time complexity which is linear in the total number of states and uncontrollable transitions of its input automaton A . Thus the highest complexity bound is linear in the (state *plus* transition) size of A . In an upper bound, the complexity is $O(|X^A| + |X^A| \times |\Sigma_{uc}^A|)$ or $O(|X^A| \times |\Sigma_{uc}^A|)$. In many real systems where $|X^A| \gg |\Sigma_{uc}^A|$, the complexity becomes $O(|X^A|)$, which is linear in the state size of A .

Procedure *CheckSafety* could be invoked by either agent to determine if a given state $x \in X^A$ is P_c -safe. The procedure does not involve inter-agent communication, but implicitly assumes that the invoking agent has complete knowledge of the discrete-event model of the other agent. In some cases, this assumption may not be satisfied *a priori*. Moreover, the procedure performs a search over the state space of the system model $A = A_1 \parallel A_2$, and therefore may incur a considerable amount of time. These issues could be addressed if we are able to build algorithms that enable the agents to perform an individual search over their own discrete-event model with no knowledge of the other agent model, and cooperate with each other via communication to determine the safety value of a given system state. In what follows, we provide insights on how one such checking process can be implemented.

For $x \in X^A$, let $UR^A(x)$ denote the set of states that are reachable from x via a sequence of uncontrollable events in Σ_{uc}^A , i.e., $UR^A(x) = \{u \in X^A \mid (\exists s \in (\Sigma_{uc}^A)^*) \delta^A(s, x) = u\}$. Recall that a state $x \in X^A$ is a tuple (x_1, x_2) with $x_1 \in X^{A_1}$ and $x_2 \in X^{A_2}$. Since $\Sigma^{A_1} \cap \Sigma^{A_2} = \emptyset$, for $x = (x_1, x_2) \in X^A$, $UR^A(x) = UR^{A_1}(x_1) \times UR^{A_2}(x_2)$, where $UR^{A_i}(x_i)$ is the set of states in X^{A_i} that are reachable from x_i via a sequence of uncontrollable events in $\Sigma_{uc}^{A_i}$, and ‘ $\times$ ’ denotes the cartesian product of the two sets. By Definition 6.5, the following proposition is immediate.

Proposition 6.1. *A state $x = (x_1, x_2) \in X^A$ is P_c -safe if and only if $(\forall u \in UR^{A_1}(x_1))(\forall v \in UR^{A_2}(x_2))(u, v) \models P_c$.*

Proposition 6.1 suggests an alternative to procedure *CheckSafety*, by which the two agents can cooperatively determine the P_c -safety value of a state $x = (x_1, x_2)$ as follows. First, each agent A_i , $i \in \{1, 2\}$, individually generates the respective state set $UR^{A_i}(x_i) \subseteq X^{A_i}$ by expanding the corresponding $\Sigma_{uc}^{A_i}$ -tree rooted at x_i in its discrete-event model. Next, one of the agents communicates its generated state set to the other agent, which then checks whether $UR^{A_1}(x_1) \times UR^{A_2}(x_2)$ satisfies P_c . This whole process can be shown to have time complexity of $O(\max(|X^{A_1}| \times |\Sigma_{uc}^{A_1}|, |X^{A_2}| \times |\Sigma_{uc}^{A_2}|))$, excluding the time for inter-agent communication and for one agent checking the P_c -satisfaction of $UR^{A_1}(x_1) \times UR^{A_2}(x_2)$. Thus, compared to procedure *CheckSafety*, it is more efficient if the inter-agent communication time is negligible. However, if communication time is significant, procedure *CheckSafety* may turn out to be more efficient.

To fix a functional reference, henceforth, we shall assume that the coordinating agents use procedure *CheckSafety*.

Theorem 6.2. *Let C be a nonempty sub-automaton of A . Assume that x_0^A is P_c -safe. Let $\pi_{\langle A_1, A_2 \rangle}$ be a coordination policy with each agent policy π_{A_i} given as: $(\forall x \in X^A)(\forall \sigma \in \Sigma^{A_i})[\sigma \in \pi_{A_i}(x)$ if and only if $\delta^A(\sigma, x)!$ and $\delta^A(\sigma, x)$ is P_c -safe]. Then $A_\pi = C^{sup}$ (i.e., $\pi_{\langle A_1, A_2 \rangle}$ is the optimal solution policy of Problem 6.1).*

Proof. Since x_0^A is P_c -safe, $(\forall x \in X^{A_\pi})\pi_{A_i}(x) \supseteq \Sigma^A(x) \cap \Sigma_{uc}^{A_i}$. Let P_{A_π} be an equivalent predicate of A_π . Trivially, $P_{A_\pi} \preceq P_c$. We now have to show that (i) P_{A_π} is coordinable, and (ii) for any coordinable predicate $P \preceq P_c$, $P \preceq P_{A_\pi}$.

(i) Since A_π is a reachable automaton, P_{A_π} trivially satisfies Condition (1) of Definition 6.3. Moreover, since $(\forall x \in X^{A_\pi})(\forall \sigma \in \Sigma_{uc}^A)\delta^A(\sigma, x)$ is P_c -safe, implying $\delta^A(\sigma, x) \models P_c$. It follows that P_{A_π} satisfies Condition (2) of Definition 6.3. Hence P_{A_π} is coordinable.

(ii) Let $P \preceq P_c$ be a coordinable predicate. Let $x \in X^A$ be an arbitrary state that satisfies P . Then x is reachable from x_0^A via a sequence of states satisfying P , and $(\forall \sigma \in \Sigma^A(x) \cap \Sigma_{uc}^A)\delta^A(\sigma, x) \models P$. For a string $s \in (\Sigma_{uc}^A)^*$, by induction on the length of s , we infer that if $\delta^A(s, x)!$ then $\delta^A(s, x) \models P$. Hence, since $P \preceq P_c$, $(\forall s \in (\Sigma_{uc}^A)^*)[\delta^A(s, x)! \Rightarrow \delta^A(s, x) \models P_c]$, which in turn implies that x is P_c -safe, or $x \models P_{A_\pi}$. Hence $P \preceq P_{A_\pi}$. $\square$

Thus, by Theorem 6.2, to implement the optimal solution policy of Problem 6.1, the following *ComputeEnabledEventSet* procedure could be used by agent A_i ($i \in \{1, 2\}$) to determine the set of events to enable next each time it observes a new system state x .

Procedure *ComputeEnabledEventSet* ($x \in X^A$)

```

begin
   $\pi_{A_i}(x) \leftarrow \Sigma^A(x) \cap \Sigma_{uc}^{A_i};$ 
  foreach  $\sigma \in \Sigma^A(x) \cap \Sigma_c^{A_i}$  do
    if CheckSafety( $\delta^A(\sigma, x)$ ) == true then
       $\pi_{A_i}(x) \leftarrow \pi_{A_i}(x) \cup \{\sigma\}$ 
  Return  $\pi_{A_i}(x)$ 
end

```

The procedure has worst-case time complexity of $O(|\Sigma_c^{A_i}|(|X^A| \times |\Sigma_{uc}^A|))$ since it has to invoke procedure *CheckSafety* exactly $|\Sigma^A(x) \cap \Sigma_c^{A_i}|$ times.

6.4 On-line Coordination Strategies

To implement the optimal coordination policy given in Theorem 6.2, i.e., policy $\pi_{\langle A_1, A_2 \rangle}$ for which $A_\pi = C^{sup}$, we now present two on-line coordination strategies that enable the agents to interact and individually compute their next coordinating actions in response to continual situational changes.

6.4.1 With Full Communication

The first on-line strategy is called OnlineCoAgent-ComFull (Fig. 6.1), and follows directly from Theorem 6.2. Using the strategy, the agents start by exchanging their initial states; and upon entering a new state, an agent would immediately send its updated local state to the other agent. Each time the agents have individually updated the system state, they would apply *ComputeEnabledEventSet* to determine their next set of events to enable.

Although easy to implement, OnlineCoAgent-ComFull entails full communication and therefore may not be desirable for situations in which communication bandwidth is a scarce resource. For such situations, other coordination strategies which could reduce communication are needed. OnlineCoAgent-ComFull can however be used to benchmark against the effectiveness in bandwidth reduction of these strategies.

Figure 6.1: OnlineCoAgent-ComFull(A_i)($i \in \{1, 2\}$)- On-line coordination strategy with full communication for agent A_i . For definiteness of description, the strategy instance for A_1 is shown; that for A_2 is the same except that its reciprocal agent is A_1 .

6.4.2 With Reduced Communication

The second coordination strategy attempts to reduce communication bandwidth. It uses the concept of an agent's (coordination) view to implement the optimal policy given in Theorem 6.2.

The local view of agent A_1 is represented by the tuple $(x_1, x_2^{r_1}, x_1^{s_2})$, where x_1 is its current state, $x_2^{r_1}$ is A_1 's view of A_2 's current state and is the most recent state information A_1 received from agent A_2 , and $x_1^{s_2}$ is the most recent state information that A_1 sent to A_2 . The local view of agent A_2 is similarly represented by the tuple $(x_2, x_1^{r_2}, x_2^{s_1})$. Note that since inter-agent communication is assumed instantaneous, $x_1^{s_2} = x_1^{r_2}$ and $x_2^{s_1} = x_2^{r_1}$. Note also that without one agent always updating the other agent with its current local state, $x_1^{r_2}$ and $x_2^{r_1}$ might be different from x_1 and x_2 , respectively. When $x_1^{r_2} = x_1$ and $x_2^{r_1} = x_2$, i.e.,

both the agents have the most current information about each other's local state, the agents are said to be totally synchronized.

To always achieve total synchronization, the agents would have to send their updated local state to the other whenever they enter a new state, i.e., they would have to follow the full communication approach. Since total synchronization may not always be necessary for coordination, the communication needs between coordinating agents could be reduced by enabling the agents to strategically decide, based on their local view, whether or not to send their current local state to the other. In developing one such strategy, the following definition is needed.

Definition 6.6. Given $x_1 \in X^{A_1}$, two states $x_2, x'_2 \in X^{A_2}$ are said to be equivalent with respect to x_1 (on P_c), and denoted by $x_2 \equiv_{x_1} x'_2 \pmod{P_c}$, if $(\forall \sigma \in \Sigma^{A_1}(x_1) \cap \Sigma_c^{A_1}) [(\delta^A(\sigma, x_1), x_2) \text{ is } P_c\text{-safe} \text{ if and only if } (\delta^A(\sigma, x_1), x'_2) \text{ is } P_c\text{-safe}]$. The notion $x_1 \equiv_{x_2} x'_1$ for $x_1, x'_1 \in X^{A_1}$ and $x_2 \in X^{A_2}$ is defined similarly.

Intuitively, for $i, j \in \{1, 2\}$, $x_i \equiv_{x_j} x'_i \pmod{P_c}$ means that whether agent A_i is in state x_i or x'_i , the coordinating actions of agent A_j (regarding which events it should enable or disable) are the same when maintaining P_c . An important implication is that if A_j is in state x_j , A_i does not need to inform A_j when it moves from state x_i to x'_i .

For economy of notation, we will often omit ' $\pmod{P_c}$ ' and simply write $x_i \equiv_{x_j} x'_i$ in place of $x_i \equiv_{x_j} x'_i \pmod{P_c}$ when no ambiguity can arise. Given $x_j \in X^{A_j}$, $\equiv_{x_j}$ defines an equivalence relation on the state set X^{A_i} . As per usual, a partial order relation can be defined over the set of those equivalence relations.

Definition 6.7. For two states $x_1, x'_1 \in X^{A_1}$, $\equiv_{x_1}$ is said to be finer than $\equiv_{x'_1}$, and denoted by $\equiv_{x_1} \preceq \equiv_{x'_1}$, if $(\forall x_2, x'_2 \in X^{A_2})(x_2 \equiv_{x_1} x'_2) \Rightarrow (x_2 \equiv_{x'_1} x'_2)$. The notion $\equiv_{x_2} \preceq \equiv_{x'_2}$ for $x_2, x'_2 \in X^{A_2}$ is defined similarly.

Thus, $\equiv_{x_j} \preceq \equiv_{x'_j}$ means that if A_i does not need to inform A_j in state x_j when it moves from state x_i to x'_i , it also does not need to do so if A_j is in state x'_j .

We now define the main concept called coordination-readiness that characterizes when the two coordinating agents A_1 and A_2 can correctly determine their next set of enabled events to maintain P_c .

Definition 6.8. Two agents A_1 and A_2 , with their respective local views $(x_1, x_2^{r_1}, x_1^{s_2})$ and $(x_2, x_1^{r_2}, x_2^{s_1})$, are said to be coordination-ready (for P_c) if $x_1^{s_2} \equiv_{x_2} x_1$ and $x_2^{s_1} \equiv_{x_1} x_2$.

Thus, the two agents are coordination-ready if, $x_i^{s_j}$, the most recent state information A_i sent to A_j , presents A_j with the equivalent next-state information associated with x_i , the current state of A_i , for determining the same P_c -safety value of every next system state that can result from A_j 's execution of a controllable event from its current state x_j , $i, j \in \{1, 2\}$.

Hence, to implement the optimal solution policy $\pi_{\langle A_1, A_2 \rangle}$ for which $A_\pi = C^{sup}$ (Theorem 6.2), the agents, following every event execution, would need to re-establish coordination-readiness prior to determining their next set of enabled events. Note that always re-establishing total synchronization as with OnlineCoAgent-ComFull is the most conservative way that trivially and implicitly re-establishes coordination-readiness. Checking for coordination-readiness first, with $x_i^{s_j} \equiv_{x_j} x_i$ by agent A_i , might reduce A_i to communicating its current local state to the other agent A_j only when the check fails. However, such direct checking clearly requires agent A_i to also know the current local state x_j of agent A_j , which is not always possible. This necessitates a stronger notion called co-stability, whose conditions can be mutually checked by the agents.

Definition 6.9. *Two agents A_1 and A_2 with their respective local views $(x_1, x_2^{r_1}, x_1^{s_2})$ and $(x_2, x_1^{r_2}, x_2^{s_1})$, are said to be co-stable (for P_c) if (1) $x_1^{s_2} \equiv_{x_2^{r_1}} x_1$, (2) $x_2^{s_1} \equiv_{x_1^{r_2}} x_2$, (3) $\equiv_{x_1^{s_2}} \preceq \equiv_{x_1}$, and (4) $\equiv_{x_2^{s_1}} \preceq \equiv_{x_2}$.*

The following proposition formally states that co-stability is a sufficient condition for coordination-readiness.

Proposition 6.2. *Whenever agents A_1 and A_2 are co-stable (for P_c), they are coordination-ready (for P_c).*

Importantly, the co-stability conditions could be mutually checked by the two agents A_1 and A_2 as follows. Conditions (1) and (3), which only require information access to $x_1, x_2^{r_1}$ and $x_1^{s_2}$, can be checked by agent A_1 using its local view $(x_1, x_2^{r_1}, x_1^{s_2})$. To check Condition (1), i.e., whether $x_1^{s_2} \equiv_{x_2^{r_1}} x_1$, A_1 can simply check, for each $\sigma \in \Sigma^{A_2}(x_2^{r_1}) \cap \Sigma_c^{A_2}$, whether the two system states $(x_1^{s_2}, \delta^{A_2}(\sigma, x_2^{r_1}))$ and $(x_1, \delta^{A_2}(\sigma, x_2^{r_1}))$ have the same P_c -safety value. This checking process has worst-case time complexity of $O(|\Sigma_c^{A_2}|(|X^A| \times |\Sigma_{uc}^A|))$ since it involves invoking procedure *CheckSafety* exactly $2 \times |\Sigma^{A_2}(x_2^{r_1}) \cap \Sigma_c^{A_2}|$ times. Similarly, to check Condition (3), i.e., whether $\equiv_{x_1^{s_2}} \preceq \equiv_{x_1}$, A_1 can iterate over every state pair $x_2, x_2' \in X^{A_2} \times X^{A_2}$ and check whether $(x_2 \equiv_{x_1^{s_2}} x_2')$ implies $(x_2 \equiv_{x_1} x_2')$. This checking process can be shown to have worst-case time complexity of $O(|\Sigma_c^{A_2}|(|X^A| \times |\Sigma_{uc}^A|)|X^{A_2}|^2)$. Conditions (2) and (4) can be checked by agent A_2 in a similar manner.

Thus, to re-establish coordination-readiness following an event execution, the agents can check the co-stability conditions, and when necessary, interact by communicating their local state to re-establish co-stability using a new strategy called OnlineCoAgent-ComReduce (Fig. 6.4.2); we call this process co-stabilization.

Figure 6.2: OnlineCoAgent-ComReduce(A_i) ($i \in \{1, 2\}$) - On-line coordination strategy with reduced communication for agent A_i . For definiteness of description, the strategy instance for A_1 is shown; that for A_2 is the same except that its reciprocal agent is A_1 .

Note that the strategy OnlineCoAgent-ComFull (Fig. 6.1) implicitly and trivially guarantees attaining co-stability, by requiring the two agents to always communicate to re-establish total synchronization between themselves. As Theorem 6.3 below formally states, the strategy OnlineCoAgent-ComReduce (Fig. 6.4.2) can also attain co-stability, but without the agents always having to achieve total synchronization. Importantly, this suggests that the latter strategy can reduce inter-agent communication.

Theorem 6.3. *Using OnlineCoAgent-ComReduce, two agents A_1 and A_2 can always, after every event execution, co-stabilize between themselves (and hence become coordination-ready) for P_c .*

Proof. Using OnlineCoAgent-ComReduce, the agents start by exchanging their initial local states, and are initially co-stable. Now, assume that the agents are currently co-stable with their respective local views $(x_1, x_2^{r_1}, x_1^{s_2})$ and $(x_2, x_1^{r_2}, x_2^{s_1})$. Upon executing a local event σ and moving to a new local state, using OnlineCoAgent-ComReduce, one agent (say A_1) updates its local state to x'_1 and initiates the communication process with the other agent A_2 as follows.

- A_1 first checks if the two conditions $\equiv_{x_1^{s_2}} \preceq \equiv_{x'_1}$ and $x_1^{s_2} \equiv_{x_2^{r_1}} x'_1$ are satisfied. If so, the agents are already co-stable, and no inter-agent communication is needed.
- If, however, $\equiv_{x_1^{s_2}} \not\preceq \equiv_{x'_1}$ or $x_1^{s_2} \not\equiv_{x_2^{r_1}} x'_1$, A_1 communicates x'_1 to A_2 , and updates its local view to $(x'_1, x_2^{r_1}, x'_1)$, validating Condition (1) $[x'_1 \equiv_{x_2^{r_1}} x'_1]$ and Condition (3) $[\equiv_{x'_1} \preceq \equiv_{x'_1}]$ of co-stability. Upon receiving x'_1 , A_2 updates its local view to $(x_2, x'_1, x_2^{s_1})$. Condition (4) $[\equiv_{x_2^{s_1}} \preceq \equiv_{x_2}]$ is still satisfied because the agents are co-stable prior to A_1 executing event σ . A_2 then proceeds to check Condition (2) $[x_2^{s_1} \equiv_{x'_1} x_2]$ of co-stability. There are two cases:
 - *Case (1):* $\equiv_{x_1^{s_2}} \preceq \equiv_{x'_1}$. Together with $x_2^{s_1} \equiv_{x_1^{r_2}} x_2$ (because the agents are co-stable prior to A_1 executing event σ), it implies $x_2^{s_1} \equiv_{x'_1} x_2$, i.e., Condition (2) is satisfied. In this case, the agents are co-stable after A_2 has acted upon the communication message x'_1 received from A_1 , and thus no further communication is needed.
 - *Case (2):* $\equiv_{x_1^{s_2}} \not\preceq \equiv_{x'_1}$. In this case, Condition (2) may not be satisfied. If so, A_2 then communicates x_2 to A_1 , and in response, A_1 updates its view of A_2 's current state from $x_2^{r_1}$ to x_2 . Now the two agents are totally synchronized and are therefore co-stable.

Thus, the two agents can always, after every event execution, co-stabilize between themselves. $\square$

Thus, OnlineCoAgent-ComReduce enables two coordinating agents A_1 and A_2 to maintain the supremal coordinable predicate of the constraint P_c , and hence implement the optimal solution policy of Problem 6.1.

6.5 Illustrative Example

We now present an example to explain the effectiveness of our on-line coordination strategy with reduced communication. The example under study is an exploration problem consisting of two agents A_1 and A_2 concurrently exploring a common space. The common space to be explored consists of three regions R_1 , R_2 and R_3 which are far away from each other, and each region consists of n rooms to be explored (Fig. 6.5). To explore the space, each agent moves from one room to another, and explores each room individually. Since the regions are far from each other, the cost of moving from one region to another is much more expensive than the cost of moving among the rooms in the same region. Therefore, after entering one region, the agents would want to freely explore every room in the region before moving to another region. Thus it is reasonable to pre-specify the events representing each agent moving inside a region as uncontrollable, and those representing each agent moving from one region to another as controllable.

Consider an inter-agent (predicate) constraint P_c specifying that ‘the two agents must not explore the same room at the same time’. Since an agent could uncontrollably move from one room to another in the same region, it is easy to see that the supremal coordinable predicate of P_c , denoted by P_c^{sup} , is that ‘the two agents must not explore the same region at the same time’. Agents A_1 and A_2 can utilize the proposed OnlineCoAgent-ComReduce strategy to interact and communicate to maintain P_c^{sup} as follows.

Defining only those states necessary for illustration, for $1 \leq i \leq n$, let $a_{1,i}$ denote the state of agent A_1 when it is exploring room i in region R_1 ; and $b_{2,i}$ and $c_{2,i}$ denote the states of agent A_2 when it is exploring room i in region R_2 and region R_3 respectively. From Definitions 6.6 and 6.7, it can be verified that: $(\forall i, j, k, h \in [1, n]) \ b_{2,i} \equiv_{a_{1,k}} b_{2,j}, c_{2,i} \equiv_{a_{1,k}} c_{2,j}, \equiv_{a_{1,h}} \preceq \equiv_{a_{1,k}} \text{ and } \equiv_{b_{2,j}} \preceq \equiv_{b_{2,i}} \ (1)$.

Suppose A_1 is exploring room k in region R_1 and A_2 is exploring room i in region R_2 , and their respective local views are $(a_{1,k}, b_{2,j}^{r_1}, a_{1,h}^{s_2})$ and $(b_{2,i}, a_{1,h}^{r_2}, b_{2,j}^{s_1})$, where $b_{2,j}^{r_1}$ and $a_{1,h}^{r_2}$ might be different from $b_{2,i}$ and $a_{1,k}$. By (1), the agents are co-stable. Now, suppose agent A_2 moves to another room. Whether A_2 needs to inform A_1 of its updated local state will

Figure 6.3: A multiagent coordination example

depend on whether the agents need to re-establish co-stability:

(i) If A_2 moves to room r in region R_2 , its local view becomes $(b_{2,r}, a_{1,h}^{r_2}, b_{2,j}^{s_1})$. By (1), A_2 can locally verify that the agents are still co-stable, and it therefore does not need to communicate its updated local state $b_{2,r}$ to A_1 .

(ii) If, however, A_2 moves to room r in region R_3 , its local view becomes $(c_{2,r}, a_{1,h}^{r_2}, b_{2,j}^{s_1})$. Since $b_{2,j} \not\leq c_{2,r}$, A_2 needs to communicate its local state $c_{2,r}$ to A_1 , using which A_1 updates its local view to $(a_{1,k}, c_{2,r}^{r_1}, a_{1,h}^{s_2})$. By (1), A_1 can verify that the agents are now co-stable, so no further communication is needed.

Thus, using OnlineCoAgent-ComReduce, to maintain constraint P_c , each agent needs to inform the other only when it moves from one region to a different region. In contrast, using OnlineCoAgent-ComFull (Fig. 6.1) to maintain constraint P_c , the agents would have to immediately inform the other each time they move to a different room, and not till they move to a different region. Therefore, if, for example, the agents spend 90% of their event execution time exploring their current region, then compared to using OnlineCoAgent-ComFull, using OnlineCoAgent-ComReduce could save about 90% of their communication bandwidth.

6.6 Experimental Evaluation

In this section, we present an experimental investigation of the effectiveness in bandwidth reduction of OnlineCoAgent-ComReduce. We compare the number of local-state messages communicated between the agents when using OnlineCoAgent-ComReduce (Fig. 6.4.2) with that when using the benchmark strategy OnlineCoAgent-ComFull (Fig. 6.1). For the experiments, we created different pairs of agent models and different predicate constraints

as follows.

Agent Model: We randomly created three different pairs of coordinating agent models with 30, 35 and 40 states, and 15, 10, and 20 events, respectively. Each event was randomly specified as either controllable or uncontrollable.

Inter-agent Constraint: For each pair of agent models, we randomly created different constraints (CS's) with varying degrees of permissiveness or restrictiveness imposed on the coordinating agents. Constraint permissiveness is defined by the ratio α of the number of states satisfying the CS to the total number of system states. This constraint permissiveness ratio α approaches 1 when the CS is the most permissive, i.e., most of the system states satisfy the constraint; and approaches 0 when the CS is the most restrictive, i.e., only a few of the system states satisfy the constraint. For our experiments, four test CS's with α over the representative range of 0.2, 0.4, 0.6 and 0.9 were created.

For each pair of agent models and an inter-agent constraint, we ran the experiment 50 times, each time of 1000 run-time steps, with each run-time step corresponding to an event execution. At each step, we chose a random event from the set of enabled events for execution. After each experiment, we recorded the number of state messages exchanged between the agents when (i) using OnlineCoAgent-ComFull and (ii) using OnlineCoAgent-ComReduce. We then calculated and summarized in Table 6.1 the average and standard deviation of the bandwidth reduction (in percent) of OnlineCoAgent-ComReduce over OnlineCoAgent-ComFull.

Table 6.1: Bandwidth reduction (in %) of OnlineCoAgent-ComReduce over OnlineCoAgent-ComFull for varying degrees of CS permissiveness α

Constraint	Agent models: Pair 1		Agent models: Pair 2		Agent models: Pair 3	
	Avg	Stdev	Avg	Stdev	Avg	Stdev
CS1 ($\alpha = 0.2$)	15.92	0.81	12.63	1.45	7.46	0.56
CS2 ($\alpha = 0.4$)	5.75	0.23	26.31	1.67	37.9	1.77
CS3 ($\alpha = 0.6$)	28.56	1.47	5.03	0.79	15.01	1.36
CS4 ($\alpha = 0.9$)	95.79	2.68	83.72	1.08	94.21	2.17

From these simulation results, we make the following observations. Firstly, the average bandwidth reduction ranges from 5.75% to 95.79% with relatively small standard deviations, indicating the effectiveness of the reduced communication strategy. Secondly, it is extremely high for CS4. Since CS4 is the constraint randomly generated with $\alpha = 0.9$,

most of the system states satisfy the constraint. Thus the agents are loosely coupled by the constraint. As a result, they were often co-stable during coordination, and seldom needed to communicate with each other. The reduced communication strategy could apparently exploit such loose coupling between the agents and offer a tremendous advantage in terms of bandwidth savings over the full communication strategy.

Finally, the bandwidth reduction is not monotonically increasing with increments of α over the range $[0.2, 0.4, 0.6, 0.9]$. This non-monotonic increase is due apparently to the fact that it is the relationship between the predicate constraint and the transitional structure of the system, not just the coverage of the constraints over the system states, that determines the essential communication needs between the agents. To illustrate this point, consider the example presented in Section 6.5 again. Assume that each of the three regions (R_1 , R_2 and R_3) to explore contains 10 rooms. Then the two-agent system would have 30×30 or 900 states, out of which $2 \times \binom{3}{2} \times 10 \times 10$ or 600 states would satisfy the constraint P_c^{sup} that ‘the agents must not explore the same region at the same time’. Let us consider another constraint $P_{c'}$, which is more restrictive than P_c , specifying that ‘the two agents must not explore the same room at the same time, and while A_1 is exploring a room in region R_3 , A_2 must not explore any room in region R_2 ’. Since both the agents could uncontrollably move from one room to another in the same region, the supremal coordinable predicate of $P_{c'}$, denoted by $P_{c'}^{sup}$, is that ‘the agents must not explore the same region at the same time, and while A_1 is exploring region R_3 , A_2 must not explore region R_2 ’. It follows that $(600 - 100)$ or 500 out of 900 system states would satisfy the constraint $P_{c'}^{sup}$. Let x_i, x'_i denote two arbitrary states of agent A_i , and x_j denote an arbitrary state of agent A_j ($i, j \in \{1, 2\}$). Then we note that $x'_i \equiv_{x_j} x_i \pmod{P_c}$ if and only if $x'_i \equiv_{x_j} x_i \pmod{P_{c'}}$. Therefore, using the same OnlineCoAgent-ComReduce strategy, satisfying either constraint would entail the same essential communication needs, namely, *one agent would need to communicate its updated local state to the other only upon entering a new region*, due to the similar $\equiv_{x_j}$ -relationship between the two constraints and the transitional structure of the system $A = A_1 \parallel A_2$. Hence, the bandwidth reduction in both cases is almost the same even though one constraint is more permissive than the other.

6.7 Multiagent On-line Coordination

We now show how the results developed can be extended to multiple agents. Let $A_1, A_2, \dots, A_n$ be $n \geq 2$ reachable automata modeling n discrete-event agents, where $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset$ for $1 \leq i \neq j \leq n$. Let $A = A_1 \parallel A_2 \parallel \dots \parallel A_n$ with $\Sigma_c^A = \Sigma_c^{A_1} \cup \Sigma_c^{A_2} \cup \dots \cup \Sigma_c^{A_n}$ and $\Sigma_{uc}^A = \Sigma_{uc}^{A_1} \cup \Sigma_{uc}^{A_2} \cup \dots \cup \Sigma_{uc}^{A_n}$. Let P_c be a predicate defined on X^A . Using the notation defined in the previous sections, Theorems 6.4 and 6.5 can be trivially extended to Theorems 6.1 and 6.2 as follows.

Theorem 6.4. *Let C be a nonempty sub-automaton of A . Then there exists a coordination policy $\pi_{\langle A_1, \dots, A_n \rangle}$ such that $A_\pi = C$ if and only if P_c is coordinable.*

Theorem 6.5. *Let C be a nonempty sub-automaton of A . Assume that x_0^A is P_c -safe. Let $\pi_{\langle A_1, A_2, \dots, A_n \rangle}$ be a coordination policy with each agent policy π_{A_i} given as: $(\forall x \in X^A)(\forall \sigma \in \Sigma^{A_i})[\sigma \in \pi_{A_i}(x)$ if and only if $\delta^A(\sigma, x)!$ and $\delta^A(\sigma, x)$ is P_c -safe]. Then $A_\pi = C^{sup}$.*

The proofs of Theorems 6.4 and 6.5 are respectively almost identical to those of Theorems 6.1 and 6.2, except that n agents are used in place of 2 agents in the original proofs whenever appropriate. In what follows, with trivial modification, the full communication strategy OnlineCoAgent-ComFull can be used by the agents to coordinate among themselves to maintain the supremal coordinable predicate of P_c . Using this strategy, the agents start by exchanging their initial states; and upon entering a new state, an agent would immediately send its updated local state to every other agent. Each time the agents have individually updated the system state, they would apply procedure *ComputeEnabledEventSet* to determine their next set of events to enable.

Unlike OnlineCoAgent-ComFull, extending the strategy OnlineCoAgent-ComReduce to multiple agents is not trivial. The reason is that, to do so, coordination-readiness and co-stability concepts, similar to those in Definitions 6.8 and 6.9, would have to be defined for multiple agents. Agents in on-line coordination need to establish or re-establish coordination-readiness upon each situational change before they can correctly determine their next local coordinating actions. To reduce communication, an agent can reason and communicate based on the stronger notion of co-stability. That would require each agent in the multiagent system to maintain a local coordination view of the most recent state information that it sent to and received from every other agent; and upon executing an event

and entering a new state, the agent would have to determine which of the other agents it has to send the state information update to in order to re-establish co-stability. This is a challenging problem and a topic for future research.

6.8 Chapter Summary

This chapter has presented a predicate coordination problem formalizing how discrete-event agents can interact and communicate in an on-line fashion to guarantee the invariance of a predicate specifying an inter-agent constraint of fundamental safety type. Specifically, the necessary and sufficient condition of predicate coordinability for coordinating agents to meet a given predicate is established (Theorem 6.1) and an optimal policy by which the agents can coordinate to maintain the supremal coordinable predicate of a given predicate is presented (Theorem 6.2). To implement the optimal policy, a key concept called coordination-readiness and its stronger notion called co-stability have been developed. The former characterizes the conditions in which coordinating agents can correctly determine their next coordinating actions as required by the optimal policy, and the latter includes conditions that can be checked and established by the agents using their local information. Based on the conditions of coordinable-readiness and co-stability, two on-line coordination strategies with full and reduced communication are proposed. As demonstrated by experimental evaluation, compared to the former, the latter strategy can achieve significant bandwidth reduction while still guaranteeing coordination quality (Theorem 6.3).

As discussed in Section 6.2, a predicate constraint P_c specifies an inter-agent constraint of the fundamental safety type, asserting that no bad states can ever be visited during multiagent interaction. In a different and important development, Yokoo et al. [109, 108] formulate a distributed constraint satisfaction problem (DCSP) for inter-agent constraints of the domain-value type. In their framework, each agent is represented as a variable with an associated domain of values, and an inter-agent constraint to satisfy is equivalent to a set of *nogoods* to exit from, modeling the constraint-violated (or inconsistent) value combinations for the agent variables. An agent state in our framework is reminiscent of an agent domain value in DCSP. However, the essence of coordination in our work is to completely avoid entering the set of bad multiagent (or composite) states induced by a predicate constraint P_c , whereas the essence of a DCSP mechanism is to exit from the set of *nogoods* induced by a domain-valued constraint. In DCSP, an agent action is a *domain value* whereas in our

framework, it is *enabling or disabling an event*. In what follows, the focus in DCSP is to enable multiple variable agents to cooperatively search for a combination of their actions that satisfies a given domain-valued constraint. Our focus is to enable *discrete-event* agents to cooperatively compute their coordinating actions so as to always satisfy the supremal coordinable predicate P_c^{sup} of a given safety constraint P_c . Satisfying P_c^{sup} - which corresponds to staying within the largest feasible good state subset of the agents' composite state space - allows these agents to have maximal autonomy over their own actions during interaction, as explained in Section 6.2. In DCSP, agent autonomy could manifest itself in the form of weak commitment to action selection [107, 108]. All in all, in parallel with DCSP [109, 108], our coordination framework provides a new distributed constraint satisfaction foundation for multiagent cooperation research.

The problem of communication reduction has also attracted increasing attention in recent years. For example, Shen and Lesser [83] and Seow et al. [80] develop algorithms to construct near-optimal agent communication strategies for their coordination problems, but their algorithms require off-line planning which may be expensive. Dutta et al. [25] develop a selective communication strategy which, however, cannot guarantee coordination quality. In contrast, our contributions include novel on-line coordination strategies that guarantee coordination quality in some specific sense defined, including one that can achieve significant savings in communication bandwidth, as theoretically proved in Theorem 6.2 and Theorem 6.3 and empirically verified in Section 6.6.

Chapter 7

CONCLUSION AND FUTURE WORK

In this thesis, we have developed a formal discrete-event framework for multiagent coordination, and addressed a series of problems concerning multiple agents interacting and communicating to achieve conformance to pre-specified inter-agent constraints.

In our opinion, research on formal coordination synthesis is still relatively new. Automata theory provides the general foundation of computer systems, and we have utilized automata - interpreted as discrete-event processes - as the foundation of agents residing in computer systems. We believe that coordination synthesis founded on discrete-event automata, when fully developed, is general and will have wide applicability for a variety of distributed service systems. In Section 7.1, we summarize the main contributions of this thesis in more detail. We then discuss some interesting directions for future research in Section 7.2.

7.1 Contributions of The Thesis

The major contributions of this thesis can be summarized as follows.

- 1) We have developed a formal framework to analyze the interactions between local agent actions and support the design of coordination modules (CM's) for a class of agents modeled as distributed discrete-event processes. As discrete-event systems become increasingly important in today's computer-dependent world, we believe that, in parallel with other frameworks for multiagent coordination, our discrete-event framework contributes a new theoretical basis that addresses the coordination problem for an important class of agents.

2) We have developed the concept of a coordinable language and shown that it is the necessary and sufficient existence condition of CM's for distributed agents to satisfy a pre-specified inter-agent constraint. The coordinability concept developed borrows and re-interprets the mathematical conditions of the language concepts of controllability [66] and observability [45]. This re-interpretation follows from an important insight discussed in this thesis, namely, discrete-event agent coordination and discrete event supervisory control are conceptually different but mathematically related. In what then follows, we could readily develop a new algorithm for near-optimal synthesis of CM's that, importantly, offers design improvement over an existing algorithm [80] in terms of reduction in inter-agent communication and CM state sizes, as explained and demonstrated.

3) We have proposed a new formalism called the Distributed Constraint Satisfaction Network (DCSN) to organize a network of coordinating agents and distributed constraints. The formalism has been shown to be naturally suited in describing the networking constraint relationships among agents. Importantly, a DCSN can be converted to a constraint relational network (CRN) that is amenable to efficient decomposition for recursive plan generation, providing opportunities to managing and mitigating the complexity of synthesizing CM's for a network of coordinating agents.

4) We have developed a compositional approach to synthesize agent CM's for a given DCSN. The approach emanates from two novel research ideas of using DCSN's to specify the networking constraint relationships between agents, and using AND/OR graphs to represent and encompass conflict resolution plans for a DCSN. This approach provides a systematic way of analyzing and designing complex distributed multiagent systems, and can often facilitate a time efficient process of synthesizing CM's for a large network of interacting agents and constraints.

5) We have proposed efficient algorithms for designing deconflicting CM's to resolve the conflicts between agents due to their using local CM's synthesized for different inter-agent constraints. This efficiency stems from using projections of discrete-event agent models onto some selected event sets; and this often enables designing deconflicting CM's without having to compute the synchronous product of all agents and constraints in the subnets concerned.

6) We have proposed a novel approach to generate a correct and complete set of conflict resolution plans for a given DCSN. In particular, we have shown that conflict resolution plans for a DCSN can be compactly represented and encompassed using AND/OR graphs.

An algorithm has been developed to generate an AND/OR graph representation of conflict resolution plans for a given DCSN. This is done by applying cut-set theory when recursively decomposing the CRN constructed from the DCSN into basic subnet components. Importantly, we have shown that cut-set acceptance tests can be developed and incorporated into the algorithm to generate an AND/OR graph representation of conflict resolution plans that satisfy specific problem-dependent requirements. In this direction, we have discussed how the algorithm can be extended to support planning for multiagent nonblocking reconfigurability.

7) We have developed a search algorithm that, when used with an admissible heuristic designed for some criterion-based optimization metric, is guaranteed to select and return an optimal plan from an AND/OR graph plan. To this end, we have presented the design of an important heuristic for selecting a plan that meets the criterion of maximal simultaneity in the execution of subnet compositions. This plan can often be executed in minimal total execution time if there are enough resources for parallel computations.

8) We have formulated and addressed an on-line coordination problem as developing distributed on-line strategies for agents to interact and communicate continually between themselves to conform to a given global predicate specifying an inter-agent constraint. We have developed the concept of a coordinate predicate which characterizes a feasible state subset that the agents can coordinate to remain in during their interaction. We have then proposed an optimal coordination policy by which agents can interact and communicate to guarantee that transitions of their composite states always remain confined to the large feasible state subset of a given predicate constraint. This largest feasible subset is characterized by the supremal coordinable predicate of the given predicate constraint. This optimal policy is important because, by implementing it, the agents can visit as many states as possible during their interaction, and as a result, have maximal autonomy over their own actions while coordinating to respect the constraint.

To implement the optimal policy, a key concept called coordination-readiness and its stronger notion called co-stability have been developed. The former characterizes the conditions in which coordinating agents can correctly determine their next coordinating actions as required by the optimal policy, and the latter includes conditions that can be checked and established by the agents using their local information. Two on-line coordination strategies are then developed. The first strategy requires agents to always inform everyone else whenever they enter a new local state. This is the most conservative way for the agents

to establish or re-establish coordination-readiness upon one agent executing an event. The second strategy is based on the agents reasoning and communicating to establish or re-establish co-stability upon one agent executing an event. Our empirical study has shown that, when compared to the first strategy, the second strategy can often significantly reduce inter-agent communication.

7.2 Future Work

The discrete-event framework developed in this thesis can be extended in several directions. In the following, we propose some interesting directions for future research.

- **Efficient algorithms for the fundamental coordination problem:** In Chapter 3, we have developed an algorithm for near-optimal coordination module synthesis. The algorithm has been shown to improve upon an existing algorithm [80, 82]. However, it still follows the same synthesis idea as the existing algorithm [80, 82], i.e., first synthesize a global supervisor for the whole system, and then “localize” it for each agent as a CM. The main drawback associated with this approach is that, when the number of agents is relatively large, synthesizing a global supervisor can be computationally expensive. To reduce this complexity, it is worth investigating a localized version of synthesis that can avoid the explicit computation of global supervisor altogether.
- **Multiagent coordination with minimal communication costs:** In this thesis, we have been concerned with the development of algorithms that can guarantee minimal event communication (Chapter 3) or reduce state communication (Chapter 6) between coordinating agents. In future work, it is interesting to address a related multiagent coordination problem in which some communication cost function is given. The goal is to develop an algorithm that computes coordination modules which can guarantee minimal communication costs as well between coordinating agents. This should extend our foundation to multiagent applications where the costs of communication are made known, either as directly provided by application agent designers, or indirectly from sources such as the real-time traffic of the underlying communication network.
- **Practical criteria and heuristics for the selection of conflict resolution plans for the networked coordination problem:** As has been discussed, our algorithm for

generating an AND/OR graph representation of conflict resolution plans can be extended to support problem-dependent planning, such as for multiagent nonblocking reconfigurability. This is done by designing and incorporating cut-set acceptance tests into the algorithm. An interesting direction for future work is to investigate different problem-dependent requirements, and derive the corresponding cut-set acceptance tests for use with the algorithm to generate an AND/OR graph representation that encompasses only plans satisfying these requirements. In the same line of research, criteria and their admissible heuristics for use with the general heuristic algorithm to select an optimal conflict resolution plan from an AND/OR graph plan should also be investigated.

- **Networked on-line coordination problem:** The compositional synthesis approach for off-line computation of CM's for a network of agents is developed in Chapters 3, 4 and 5. This is followed in Chapter 6 by an approach for on-line coordination synthesis that addresses the fundamental problem of multiple agents coordinating between themselves under one predicate constraint. In parallel with the off-line compositional synthesis approach, future work would need to address the online computation of coordination strategies for the more general problem of a network of agents coordinating under multiple, distributed constraints. In our opinion, this is a very exciting yet challenging problem.
- **Multi-level hierarchical organization for networked coordination:** Another very exciting direction is to generalize our networked coordination problem to a hierarchical, multi-level discrete-event coordination problem. The key idea is to organize distributed agents in a hierarchy of coordination levels, each of which is a network of coordinating agents. In this organization, each agent at a lower level, besides coordinating with other agents at the same level to satisfy inter-agent constraints specified for that level, has to report to designated agents situated at higher levels. In turn, an agent at a higher level has to coordinate with other agents at the same level to satisfy their higher level constraints, and at the same time, issues commands to groups of agents at different levels below the agent's. This hierarchical organization allows agents situated at higher levels to have relevant views of the levels below them, while agents situated at lower levels handle the details of specific system tasks. In the literature, system organization is often used to tackle the issues of scalability and

comprehensibility [1, 97]. We believe that when fully developed, this hierarchical discrete-event coordination approach can provide a theoretical basis for understanding and developing highly complex systems.

List of Publications

Journal Papers

1. M.T. Pham and K.T. Seow, “Discrete-Event Coordination Design for Distributed Agents,” IEEE Transactions on Automation Science and Engineering [Accepted with minor revisions in March 2011].
2. M.T. Pham and K.T. Seow, “On-line Coordination: Event Interaction and State Communication between Cooperative Agents,” Journal of Web Intelligence and Agent Systems (WIAS) [Invited Paper - Accepted in January 2011].
3. K.T. Seow, M.T. Pham, C. Ma and M. Yokoo, “Coordination Planning: Applying Control Synthesis Methods for a Class of Distributed Agents,” IEEE Transactions on Control Systems Technology, vol. 17, no. 2, pp. 405–415, March 2009, USA.

Conference Papers

1. M.T. Pham, A. Dhananjayan and K.T. Seow, “On the Transparency of Automata as Discrete-Event Control Specifications,” Proceedings of the IEEE International Conference on Robotics and Automation, May 2010, Anchorage, Alaska, USA.
2. M.T. Pham and K.T. Seow, “On-line Coordination: Event Interaction and State Communication between Cooperative Agents,” Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, September 2009, Milan, Italy.
3. M.T. Pham and K.T. Seow, “On-line Coordination Among Discrete-Event Agents,” Proceedings of the International Conference on Autonomous Agents and Multiagent

Systems, May 2009, Budapest, Hungary.

4. M.T. Pham and K.T. Seow, "Towards Synthesizing Optimal Coordination Modules for Distributed Agents," Proceedings of the AAAI Conference on Artificial Intelligence, July 2008, Chicago, Illinois, USA.

Appendix: Modular Control of Discrete-Event Systems

In this appendix, we review some important results in modular discrete-event control [99, 56, 28, 104, 19]. This modular control paradigm is usually used to tackle the computational complexity associated with the monolithic control paradigm reviewed in Chapter 2. The basic idea is to synthesize a set of modular supervisors, rather than a single monolithic supervisor, to control a large system to satisfy a set of multiple constraints. The paradigm makes extensive use of two special properties of natural projections [12], namely, natural observer [99] and output control consistency (OCC) [28]. The practical importance lies in the fact that the appropriate use of projections with observer and OCC properties in nonblocking modular control synthesis can significantly reduce the computational effort.

Modular Supervisors and Control Coordinator

Suppose a plant to be controlled is modeled by an automaton $A = A_1 \parallel A_2 \parallel \dots \parallel A_n$, which consists of n plant components A_i , $n \geq 1$, and there are two specifications modeled by two automata E and F with $\Sigma^E, \Sigma^F \subseteq \Sigma^A$. Let $A_E = \bigparallel_{\Sigma^{A_i} \cap \Sigma^E \neq \emptyset} A_i$ and $A_F = \bigparallel_{\Sigma^{A_i} \cap \Sigma^F \neq \emptyset} A_i$.

Following the monolithic synthesis paradigm [66, 65] reviewed in Chapter 2, to achieve conformance to the specifications, a single, maximally permissive supervisor S needs to be synthesized to control A such that $S \parallel A \equiv \text{Supcon}(E \parallel F, A)$. Algorithmically, S is often initially computed as $S = \text{Supcon}(E \parallel F, A)$, then a supervisor reduction procedure [95, 88] may be applied to reduce the state size of S . However, this monolithic approach is computationally very expensive. The reason is that the Supcon procedure requires the computation of $E \parallel F \parallel A$, and as a result, when n is large, the complexity of synthesizing

S is prohibitively high. Indeed, $\text{Supcon}(E \parallel F, A)$ has an exponential complexity order of $O(M^{2n}|E|^2|F|^2)$, where M is an upper bound of the state sizes of automaton models of plant components A_i 's, and $|E|$ and $|F|$ are the state sizes of automata E and F , respectively [30].

Following the modular synthesis paradigm [99, 56, 28, 104, 19], two modular supervisors $S_E = \text{Supcon}(E, A_E)$ and $S_F = \text{Supcon}(F, A_F)$ are first synthesized, and then a control coordinator S_c is designed to coordinate S_E and S_F , so that when operating concurrently, S_c , S_E and S_F will control A to achieve the same controlled behavior as the monolithic supervisor, i.e., $S_c \parallel S_E \parallel S_F \parallel A \equiv \text{Supcon}(E \parallel F, A)$. Since $\text{Supcon}(E, A_E)$ only requires the computation of $E \parallel A_E$ and $\text{Supcon}(F, A_F)$ only requires the computation of $F \parallel A_F$, if S_c can be designed efficiently, the exponential complexity associated with the monolithic synthesis approach can be mitigated.

Coordinator S_c is needed because the modular supervisors S_E and S_F , which are synthesized using only ‘local’ plant information A_E and A_F , may conflict with each other. In other words, the system of S_E and S_F controlling A may contain blocking states. It has been shown that if S_E and S_F are nonconflicting, i.e., $\overline{L_m(S_E)} \parallel \overline{L_m(S_F)} = L(S_E) \parallel L(S_F)$, they can concurrently control A to achieve the same controlled behavior as the monolithic supervisor, i.e., $S_E \parallel S_F \parallel A \equiv \text{Supcon}(E \parallel F, A)$ [19, 104]. One trivial sufficient condition for the nonconflict of S_E and S_F is that $\Sigma^E \cap \Sigma^F = \emptyset$, namely, specifications E and F are defined over two disjoint event sets. A more general condition that is often used in the control literature for testing the nonconflict of modular supervisors is given in the following proposition.

Proposition A.1 ([56]). *Let $\Sigma_o \supseteq \Sigma^E \cap \Sigma^F$. If P_{Σ^E, Σ_o} is a $L_m(S_E)$ -observer and P_{Σ^F, Σ_o} is a $L_m(S_F)$ -observer, then S_E and S_F are nonconflicting if and only if $P_{\Sigma^E, \Sigma_o}(S_E)$ and $P_{\Sigma^F, \Sigma_o}(S_F)$ are nonconflicting, namely,*

$$\overline{L_m(S_E)} \parallel \overline{L_m(S_F)} = L(S_E) \parallel L(S_F) \Leftrightarrow$$

$$\overline{L_m(P_{\Sigma^E, \Sigma_o}(S_E))} \parallel \overline{L_m(P_{\Sigma^F, \Sigma_o}(S_F))} = L(P_{\Sigma^E, \Sigma_o}(S_E)) \parallel L(P_{\Sigma^F, \Sigma_o}(S_F)).$$

Nevertheless, if S_E and S_F are conflicting, i.e., $\overline{L_m(S_E)} \parallel \overline{L_m(S_F)} \subsetneq L(S_E) \parallel L(S_F)$, a control coordinator S_c needs to be designed [100, 26]. Indeed, S_c is just another supervisor that removes blocking states from the controlled plant $S_E \parallel S_F \parallel A$. In other words, S_c can

be computed as $Supcon(G, S_E \parallel S_F)$, where G is a one-state automaton that generates and marks $(\Sigma^E \cup \Sigma^F)^*$. This is formally stated in the following proposition.

Proposition A.2 ([28]). *For the aforementioned plant and supervisors, the following holds.*

$$Supcon(G, S_E \parallel S_F) \parallel S_E \parallel S_F \parallel A \equiv Supcon(E \parallel F, A).$$

However, designing coordinator S_c as $Supcon(G, S_E \parallel S_F)$ will require the computation of $S_E \parallel S_F$, which is of the same complexity order as that of $E \parallel F \parallel A$. Thus, for the modular synthesis paradigm to be practical, there is a need to exploit the structures of controlled plants to efficiently design control coordinators.

Control Coordinator Design

One of the most common assumptions about the controlled plant structure is that its components are pair-wise independent, i.e., $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset, \forall i \neq j$. When this is the case, the following proposition offers a computationally efficient approach to design a control coordinator for modular supervisors.

Proposition A.3 ([28]). *For the aforementioned plant and supervisors and with $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset, \forall i \neq j$, let $\Sigma_o \supseteq \Sigma^E \cap \Sigma^F$. If P_{Σ^E, Σ_o} and P_{Σ^F, Σ_o} are respectively a $L_m(S_E)$ -observer and a $L_m(S_F)$ -observer, and for $1 \leq i \leq n$, $P_{\Sigma^{A_i}, \Sigma_o}$ are OCC for $L(A_i)$, then $S_c = Supcon(G_o, P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F))$ is a control coordinator for S_E and S_F , i.e.,*

$$S_c \parallel S_E \parallel S_F \parallel A \equiv Supcon(E \parallel F, A),$$

where G_o is a one-state automaton that generates and marks Σ_o^* .

Thus, when all the stated conditions in Proposition 7.2 are satisfied, a coordinator S_c can be computed by $Supcon(G_o, P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F))$, rather than by $Supcon(G, S_E \parallel S_F)$ as in Proposition 7.2. The implication is that instead of $S_E \parallel S_F$, we only need to compute $P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F)$. To see why $P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F)$ often requires less computational effort than $S_E \parallel S_F$, we first note that since $\Sigma_o \supseteq \Sigma^E \cap \Sigma^F$, $P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F) \equiv P_{\Sigma^E \cup \Sigma^F, \Sigma_o}(S_E \parallel S_F)$ by Proposition 4.2. Then, since P_{Σ^E, Σ_o} is a $L_m(S_E)$ -observer and P_{Σ^F, Σ_o} is a $L_m(S_F)$ -observer, $P_{\Sigma^E, \Sigma_o}(S_E)$ and $P_{\Sigma^F, \Sigma_o}(S_F)$ can be

computed from S_E and S_F in polynomial time [99], and their state sizes are often smaller than those of S_E and S_F , respectively. Furthermore, by Proposition 4.3, $P_{\Sigma^E \cup \Sigma^F, \Sigma_o}$ is a $(L_m(S_E) \parallel L_m(S_F))$ -observer. Therefore, the state size of $P_{\Sigma^E, \Sigma_o}(S_E) \parallel P_{\Sigma^F, \Sigma_o}(S_F)$, which is equivalent to $P_{\Sigma^E \cup \Sigma^F, \Sigma_o}(S_E \parallel S_F)$, is also often smaller than that of $S_E \parallel S_F$.

To summarize, following the modular control paradigm, we can, given two specifications E and F for a plant $A = A_1 \parallel A_2 \parallel \dots \parallel A_n$ with $\Sigma^{A_i} \cap \Sigma^{A_j} = \emptyset, \forall i \neq j$, first synthesize two modular supervisors $S_E = \text{Supcon}(E, A_E)$ and $S_F = \text{Supcon}(F, A_F)$, and then use Proposition 7.2 to check whether or not S_E and S_F are nonconflicting. If they are nonconflicting, we are done; otherwise we can follow Proposition 7.2 to design a control coordinator S_c for S_E and S_F .

References

- [1] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2010)*, pages 981–988, Toronto, Canada, May 2010.
- [2] George R. Barrett and Stéphane Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45(9):1620–1638, 2000.
- [3] Raphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Transition-independent decentralized markov decision processes. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 41–48, Melbourne, Australia, July 2003.
- [4] B. Berard, S. Haddad, L.M. Hillah, F. Kordon, and Y. Thierry-Mieg. Collision avoidance in intelligent transport systems: towards an application of control theory. In *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'04)*, pages 346–351, Goteborg, Sweeden, May 2008.
- [5] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 1287–1292, Edinburgh, Scotland, UK, August 2005.
- [6] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 478–485, Stockholm, Sweden, August 1999.

- [7] John Bresina, Richard Dearden, Nicolas Meuleau, David Smith, and Rich Washington. Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence (UAI'02)*, pages 77–84, University of Alberta, Edmonton, Alberta, Canada, August 2002.
- [8] Kai Cai and Walter M. Wonham. Supervisor localization: A top-down approach to distributed control of discrete-event systems. In *Second Mediterranean Conference on Intelligent Systems and Automation (CISA'09). AIP Conference Proceedings*, volume 1107, pages 302–308, Zarzis, Tunisia, March 2009.
- [9] Kai Cai and Walter M. Wonham. Supervisor localization: a top-down approach to distributed control of discrete-event systems. *IEEE Transactions on Automatic Control*, 55(3):605–618, 2010.
- [10] Xi-Ren Cao. *Stochastic Learning and Optimization: A Sensitivity-based Approach*. Springer, 2007.
- [11] J. C. Carroll and D. Long. *Theory of Finite Automata*. Prentice Hall, 1989.
- [12] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.
- [13] Song Foh Chew, Shengyong Wang, and Mark A. Lawley. Robust supervisory control for product routings with multiple unreliable resources. *IEEE Transactions on Automation Science And Engineering*, 6(1):195–200, 2009.
- [14] Kwang Hyun Cho and Jong Tae Lim. Multiagent supervisory control for antifault propagation in serial production systems. *IEEE Transactions on Industrial Electronics*, 48(2):460–466, April 2001.
- [15] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions*. Cambridge, MA: MIT Press, 2006.
- [16] Kevin Crowston. Coordination theory: A ten-year retrospective. In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*, pages 1–2, Porto, Portugal, April 2004.

- [17] Antonio E.C. da Cunha and José E. R. Cury. Hierarchical supervisory control based on discrete event systems with flexible marking. *IEEE Transactions on Automatic Control*, 52(12):2242–2253, 2007.
- [18] Michael S. Darms, Paul E. Rybski, Christopher Baker, and Chris Urmson. Obstacle detection and tracking for the urban challenge. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):199–234, 2009.
- [19] M.H. de Queiroz and J.E.R. Cury. Modular control of composed systems. In *Proceedings of the American Control Conference*, pages 4051–4055, Chicago, IL, USA, June 2000.
- [20] Narsingh Deo. *Graph Theory with Applications to Engineering and Computer Science*. New York: Prentice-Hall, 1974.
- [21] Edsger W. Dijkstra. *A discipline of programming*. Englewood Cliffs: Prentice-Hall, 1976.
- [22] Thomas Donnellan. *Lattice theory*. Pergamon Press, 1968.
- [23] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice Hall, 11th edition, 2007.
- [24] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, March 1989.
- [25] Partha Sarathi Dutta, Claudia V. Goldman, and Nicholas R. Jennings. Communicating effectively in resource-constrained multi-agent systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI’07)*, pages 1269–1274, Hyderabad, India, January 2007.
- [26] Lei Feng. Computationally efficient supervisor design for discrete-event systems. Phd thesis, Electrical and Computer Engineering, University of Toronto, 2007.
- [27] Lei Feng and Walter M. Wonham. Computationally efficient supervisor design: Abstraction and modularity. In *Proceedings of the Eighth International Workshop on Discrete Event Systems (WODES’06)*, pages 3–8, New York, USA, July 2006.

- [28] Lei Feng and Walter Murray Wonham. Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, 53(6):1449–1461, July 2008.
- [29] Alessandro Giua and Carla Seatzu. Modeling and supervisory control of railway networks using petri nets. *IEEE Transactions on Automation Science And Engineering*, 5(3):431–445, 2008.
- [30] Peyman Gohari and Walter M. Wonham. On the complexity of supervisory control design in the rw framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 30(5):643–652, 2000.
- [31] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.
- [32] Alireza Haji-Valizadeh and Kenneth A. Loparo. Minimizing the cardinality of an events set for supervisors of discrete-event dynamical systems. *IEEE Transactions on Automatic Control*, 41(11):1579–1593, 1996.
- [33] R. Hill and D. Tilbury. Modular supervisory control of discrete event systems with abstraction and incremental hierarchical construction. In *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06)*, pages 399–406, Ann Arbor, Michigan, USA, July 2006.
- [34] C. A. R. Hoare and J. C. Shepherdson. *Mathematical Logic and Programming Languages*. Prentice Hall, 1984.
- [35] W. M. L. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.
- [36] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Reading, MA : Addison-Wesley, 1979.
- [37] Nicholas R. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley and Sons, Inc., New York, 1996.

- [38] Nick R. Jennings. Commitments and conventions : The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
- [39] R. E. Kalman, P. L. Falb, and M. A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill, 1969.
- [40] Jana Košecká and Ruzena Bajcsy. Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12(3-4):187–198, April 1994.
- [41] Sarit Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1-2):79–97, 1997.
- [42] Ratnesh Kumar and Vijay K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, 1995. The Kluwer International Series in Engineering and Computer Science.
- [43] S. C. Lauzon, A. K. L. Ma, J. K. Mills, and B. Benhabib. Application of discrete event system theory to flexible manufacturing. *IEEE Control Systems Magazine*, 16(1):41–48, February 1996.
- [44] Feng Lin. Control of large scale discrete event systems: Task allocation and coordination. *Systems and Control Letters*, 17(3):169–175, 1991.
- [45] Feng Lin and Walter M. Wonham. On observability of discrete event systems. *Information Sciences*, 44(3):173–198, 1988.
- [46] Feng Lin and Walter M. Wonham. Decentralized control and coordination of discrete event systems with partial observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, 1990.
- [47] Kenneth C. Loudon. *Programming languages: Principles and practice*. Course Technology, USA, 2003.
- [48] Chuan Ma and Walter Murray Wonham. *Nonblocking Supervisory Control of State Tree Structures. Lecture Notes in Control and Information Sciences, Vol 317*. Springer-Verlag, New York, 2005.

- [49] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.
- [50] Janusz Marecki, Tapana Gupta, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Not all agents are equal: Scaling up distributed POMDPs for agent networks. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'08)*, pages 485–492, Estoril, Portugal, May 2008.
- [51] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2006.
- [52] Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, David Stavens, and Sven Strohband. Winning the darpa grand challenge with an ai robot. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 982–987, Boston, Massachusetts, USA, July 2006.
- [53] Nils J. Nilsson. *Principles of artificial intelligence*. New York: Springer-Verlag, 1980.
- [54] H. Van. Dyke Parunak, Sven Brueckner, Mitch Fleischer, and James Odell. A preliminary taxonomy of multi-agent interactions. In *Proceedings of The Second International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, pages 1090–1091, Melbourne, Australia, July 2003.
- [55] J. Pearl. *Heuristics: Intelligent Search Strategies For Computer Problem Solving*. Addison-Wesley Publishing Co., Reading, MA, 1984.
- [56] Patrícia N. Pena, José E. R. Cury, and Stéphane Lafortune. Testing modularity of local supervisors: An approach based on abstractions. In *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06)*, pages 107–112, Ann Arbor, Michigan, USA, July 2006.
- [57] Patrícia N. Pena, Antonio E.C. da Cunha, José E. R. Cury, and Stéphane Lafortune. New results on the nonconflict test of modular supervisors. In *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08)*, pages 468–473, Goteborg, Sweden, May 2008.

- [58] Gilles Pesant. A regular language membership constraint for finite sequences of variables. *Principles and Practice of Constraint Programming (CP 2004)*, 3258:482–495, 2004.
- [59] Manh Tung Pham, Amrith Dhananjayan, and Kiam Tian Seow. On the transparency of automata as discrete-event control specifications. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA'2010)*, pages 1474–1479, Anchorage, Alaska, USA, May 2010.
- [60] Manh Tung Pham and Kiam Tian Seow. Towards synthesizing optimal coordination modules for distributed agents. In *Proceedings of the 23th AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 1479–1480, Chicago, Illinois, USA, July 2008.
- [61] Manh Tung Pham and Kiam Tian Seow. On-line coordination: Event interaction and state communication between cooperative agents. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'2009)*, pages 315–322, Milano, Italy, September 2009.
- [62] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems – Industrial Track (AAMAS'08)*, pages 125–132, Estoril, Portugal, May 2008.
- [63] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Proceedings of the 17th Conference on Neural Information Processing Systems (NIPS'03)*, pages 823–830, Whistler, British Columbia, Canada, December 2003.
- [64] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 2005.
- [65] P. J. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

- [66] Peter J. Ramadge and Walter Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, January 1987.
- [67] Nicholas Rescher and Alasdair Urquhart. *Temporal logic*. Springer-Verlag, Library of Exact Philosophy, 1971.
- [68] S. Laurie Ricker, N. Sarkar, and Karen Rudie. A discrete event systems approach to modeling dextrous manipulation. *Robotica*, 14(5):515–525, 1996.
- [69] Kurt R. Rohloff, Samir Khuller, and Guy Kortsarz. Approximating the minimal sensor selection for supervisory control. *Discrete Event Dynamic Systems: Theory and Applications*, 16(1):143–170, 2006.
- [70] Kurt R. Rohloff and Jan H. van Schuppen. Approximating minimal communicated event sets for decentralized supervisory control. In *IFAC World Congress 2005*, <http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2005/Fullpapers/02992.pdf>.
- [71] J.S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press Cambridge, Mass, 1994.
- [72] Karen Rudie, Stéphane Lafortune, and Feng Lin. Minimal communication for essential transitions in a distributed discrete-event system. *IEEE Transactions on Automatic Control*, 52(8):1495–1502, 2007.
- [73] Karen Rudie, Stéphane Lafortune, and Feng Lin. Minimal communication in a distributed discrete-event system. *IEEE Transactions on Automatic Control*, 48(6):957–975, June 2003.
- [74] Karen Rudie and Jan C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1319, 1995.
- [75] Karen Rudie and Walter Murray Wonham. Protocol verification using discrete-event systems. In *Proceedings of the 31th IEEE Conference on Decision and Control*, pages 3770–3777, Tucson, AZ, December 1992.

- [76] Karen Rudie and Walter Murray Wonham. Think globally, act locally : Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.
- [77] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
- [78] K. Schmidt, H. Marchand, and B. Gaudin. Modular and decentralized supervisory control of concurrent discrete event systems using reduced systems models. In *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06)*, pages 149–154, Ann Arbor, Michigan, USA, July 2006.
- [79] Kiam Tian Seow. Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata. *IEEE Transactions on Automation Science and Engineering*, 4(3):451–464, July 2007.
- [80] Kiam Tian Seow, Chuan Ma, and Makoto Yokoo. Multiagent planning as control synthesis. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, pages 972–979, Columbia University, New York City, USA, July 2004.
- [81] Kiam Tian Seow and Michel Pasquier. Supervising passenger land-transport systems. *IEEE Transactions on Intelligent Transportation Systems*, 5(3):165–176, September 2004.
- [82] Kiam Tian Seow, Manh Tung Pham, Chuan Ma, and Makoto Yokoo. Coordination planning: Applying control synthesis methods for a class of distributed agents. *IEEE Transactions on Control Systems Technology*, 17(2):405–415, 2009.
- [83] Jiaying Shen and Victor R. Lesser. Communication management using abstraction in distributed bayesian networks. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06)*, pages 622–629, Future University-Hakodate, Hakodate, Japan, May 2006.
- [84] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press New York, NY, USA, 2008.

- [85] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
- [86] Yoav Shoham and Moshe Tennenholtz. On the emergence of social conventions: modeling, analysis, and simulations. *Artificial Intelligence*, 94(1):139–166, 1997.
- [87] Rong Su. Distributed diagnosis for discrete-event systems. Phd thesis, Electrical and Computer Engineering, University of Toronto, 2004.
- [88] Rong Su and Walter Murray Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems : Theory and Applications*, 14(1):31–53, 2004.
- [89] Makoto Tasaki, Yuichi Yabu, Yuki Iwanari, Makoto Yokoo, Milind Tambe, Janusz Marecki, and Pradeep Varakantham. Introducing communication in Dis-POMDPs with locality of interaction. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’2008)*, pages 169–175, Sydney, Australia, December 2008.
- [90] Jason Tsai, Shyamsunder Rathi, Christopher Kiekintveld, Fernando Ordóñez, and Milind Tambe. Iris - a tool for strategic security allocation in transportation networks. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems – Industrial Track (AAMAS’09)*, pages 37–44, Budapest, Hungary, May 2009.
- [91] John N. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(2):95–107, 1989.
- [92] Arash Vahidi, Martin Fabian, and Bengt Lennartson. Efficient supervisory synthesis of large systems. *Control Engineering Practice*, 14(10):1157–1167, 2006.
- [93] W. van der Hoek and Michael Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS’02)*, pages 1167–1174, Bologna, Italy, July 2002.

- [94] Jan H. van Schuppen. Decentralized control with communication between controllers. *Vincent D. Blondel and Alexandre Megretski, editors, Unsolved problems in mathematical systems and control theory*, pages 144–150, 2004.
- [95] A. F. Vaz and Walter Murray Wonham. On supervisor reduction in discrete event systems. *International Journal of Control*, 44(2):475–491, August 1986.
- [96] Nageshwara Rae Vempaty. Solving constraint satisfaction problems using finite state automata. In *Proceedings of the 10th AAAI Conference on Artificial Intelligence (AAAI'92)*, page 453458, San Jose, CA, USA, July 1992.
- [97] Gerhard Weiss, editor. *Multiagent System : A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, London, U.K, 1999.
- [98] Kai Cheung Wong and Jan H. van Schuppen. Decentralized supervisory control of discrete-event systems with communication. In *Proceedings of the Third International Workshop on Discrete Event Systems (WODES'96)*, pages 284–289, Edinburgh, U.K, August 1996.
- [99] Kai Cheung Wong and Walter Murray Wonham. On the computation of observers in discrete-event systems. *Discrete Event Dynamic Systems : Theory and Applications*, 14(1):55–107, 2004.
- [100] Kai Cheung Wong and Walter Murray Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems : Theory and Applications*, 6(3):241–273, July 1996.
- [101] W. Murray Wonham. *Notes on Control of Discrete-Event Systems ECE 1636F/1637S*. Systems Control Group, University of Toronto, Updated 1st July 2009. <http://www.control.toronto.edu/cgi-bin/dldes.cgi>.
- [102] Walter M. Wonham. *Control Design Software: TCT*. Developed by Systems Control Group, University of Toronto, Canada, Updated 1st July 2008. <http://www.control.toronto.edu/cgi-bin/dlxptct.cgi>.
- [103] Walter M. Wonham and Peter J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

- [104] Walter Murray Wonham and Peter J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals and Systems*, 1(1):13–30, 1988.
- [105] Michael Wooldridge. *An Introduction to Multiagent Systems, Second Edition*. John Wiley and Sons, Inc., New York, 2009.
- [106] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents : Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
- [107] Makoto Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the 1st Conference on Principles and Practice of Constraint Programming (CP'95)*, pages 88–102, July-August 1995.
- [108] Makoto Yokoo. *Distributed Constraint Satisfaction : Foundations of Cooperation in Multi-Agent Systems*. Springer-Verlag, Heidelberg, Germany, 2000. Springer Series on Agent Technology.
- [109] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem : Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.
- [110] Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96)*, pages 401–408, 1996.
- [111] Tae-Sic Yoo and Stéphane Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 12(3):335–377, 2002.
- [112] Tae-Sic Yoo and Stéphane Lafortune. NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1495–1499, 2002.
- [113] Hao Zhong and Walter Murray Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35(10):1125–1134, 1990.