

negotiation

machine learning

specification

argumentation

verification

agent

social choice

decision making

programming

second edition

Multiagent Systems

edited by

Gerhard Weiss

coalition formation

auctions

autonomy

organization

joint planning

communication

intelligence

cooperation

competition

Intelligent Robotics and Autonomous Agents

Edited by Ronald C. Arkin

Dorigo, Marco, and Marco Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*

Arkin, Ronald C., *Behavior-Based Robotics*

Stone, Peter, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*

Wooldridge, Michael, *Reasoning About Rational Agents*

Murphy, Robin R., *An Introduction to AI Robotics*

Mason, Matthew T., *Mechanics of Robotic Manipulation*

Kraus, Sarit, *Strategic Negotiation in Multiagent Environments*

Nolfi, Stefano, and Dario Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*

Siegwart, Roland, and Illah R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*

Breazeal, Cynthia L., *Designing Sociable Robots*

Bekey, George A., *Autonomous Robots: From Biological Inspiration to Implementation and Control*

Choset, Howie, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*

Mataric, Maja J., *The Robotics Primer*

Wellman, Michael P., Amy Greenwald, and Peter Stone, *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*

Floreano, Dario and Claudio Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*

Sterling, Leon S. and Kuldar Taveter, *The Art of Agent-Oriented Modeling*

Stoy, Kasper, David Brandt, and David J. Christensen, *An Introduction to Self-Reconfigurable Robots*

Lin, Patrick, Keith Abney, and George A. Bekey, editors, *Robot Ethics: The Ethical and Social Implications of Robotics*

Weiss, Gerhard, editor, *Multiagent Systems*, second edition

Multiagent Systems

second edition

edited by Gerhard Weiss

The MIT Press
Cambridge, Massachusetts
London, England

©2013 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

This book was set in Times Roman by the editor. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Multiagent systems / edited by Gerhard Weiss. — Second edition.

p. cm.—(Intelligent robotics and autonomous agents series)

Includes bibliographical references and index.

ISBN 978-0-262-01889-0 (hardcover : alk. paper)

1. Intelligent agents (Computer software) 2. Distributed artificial intelligence.

I. Weiss, Gerhard, 1962– editor of compilation.

QA76.76.I58M85 2013

006.3—dc23

2012024600

10 9 8 7 6 5 4 3 2 1

Contents in Brief

Preface	xxxv
----------------	-------------

The Subject of This Book • Main Features of This Book •
Readership and Prerequisites • Changes from the First Edition •
Structure and Chapters • The Exercises • How to Use This Book •
Slides and More – The Website of the Book • Acknowledgments

Contributing Authors	xliii
-----------------------------	--------------

<u>Part I</u> Agent Architectures and Organizations	1
--	----------

1 Intelligent Agents	3
-----------------------------	----------

Michael Wooldridge

2 Multiagent Organizations	51
-----------------------------------	-----------

Virginia Dignum and Julian Padget

<u>Part II</u> Communication	99
-------------------------------------	-----------

3 Agent Communication	101
------------------------------	------------

Amit K. Chopra and Munindar P. Singh

4 Negotiation and Bargaining	143
-------------------------------------	------------

Shaheen Fatima and Iyad Rahwan

5 Argumentation among Agents	177
-------------------------------------	------------

Iyad Rahwan

<u>Part III</u>	Basic Coordination	211
6	Computational Social Choice	213
	<i>Felix Brandt, Vincent Conitzer, and Ulle Endriss</i>	
7	Mechanism Design and Auctions	285
	<i>Kevin Leyton-Brown and Yoav Shoham</i>	
8	Computational Coalition Formation	329
	<i>Edith Elkind, Talal Rahwan, and Nicholas R. Jennings</i>	
9	Trust and Reputation in Multiagent Systems	381
	<i>Jordi Sabater-Mir and Laurent Vercouter</i>	
<u>Part IV</u>	Distributed Cognitive Abilities	421
10	Multiagent Learning	423
	<i>Karl Tuyls and Kagan Tumer</i>	
11	Multiagent Planning, Control, and Execution	485
	<i>Ed Durfee and Shlomo Zilberstein</i>	
12	Distributed Constraint Handling and Optimization	547
	<i>Alessandro Farinelli, Meritxell Vinyals, Alex Rogers, and Nicholas R. Jennings</i>	

<u>Part V</u>	Development and Engineering	585
13	Programming Multiagent Systems	587
	<i>Rafael H. Bordini and Jürgen Dix</i>	
14	Specification and Verification of Multiagent Systems	641
	<i>Jürgen Dix and Michael Fisher</i>	
15	Agent-Oriented Software Engineering	695
	<i>Michael Winikoff and Lin Padgham</i>	
<u>Part VI</u>	Technical Background	759
16	Logics for Multiagent Systems	761
	<i>Wiebe van der Hoek and Michael Wooldridge</i>	
17	Game-Theoretic Foundations of Multiagent Systems	811
	<i>Edith Elkind and Evangelos Markakis</i>	
Subject Index		849

Contents

Preface

xxxv

The Subject of This Book • Main Features of This Book •
Readership and Prerequisites • Changes from the First Edition •
Structure and Chapters • The Exercises • How to Use This Book •
Slides and More – The Website of the Book • Acknowledgments

Contributing Authors

xliii

Part I Agent Architectures and Organizations

1

1 Intelligent Agents

3

Michael Wooldridge

1	Introduction	3
2	What Are Agents?	4
2.1	Examples of Agents	7
2.2	Intelligent Agents	8
2.3	Agents and Objects	10
2.4	Agents and Expert Systems	12
2.5	Sources and Further Reading	13
3	Architectures for Intelligent Agents	13
3.1	Logic-Based Architectures	14
3.1.1	Sources and Further Reading	19
3.2	Reactive Architectures	20
3.2.1	The Subsumption Architecture	21
3.2.2	Markov Decision Processes	25
3.2.3	Sources and Further Reading	27
3.3	Belief-Desire-Intention Architectures	28
3.3.1	Sources and Further Reading	35
3.4	Layered Architectures	36

3.4.1	TouringMachines	38
3.4.2	InteRRaP	40
3.4.3	Sources and Further Reading	42
4	Conclusions	42
5	Exercises	42
	References	45
2	Multiagent Organizations	51
	<i>Virginia Dignum and Julian Padget</i>	
1	Introduction	51
2	Background	53
2.1	From Intelligent Agents to Multiagent Systems	53
2.2	From Multiagent Systems to Multiagent Organizations	55
2.3	Sources of Inspiration	56
2.3.1	Organization as Structure	56
2.3.2	Organization as Institution	58
2.3.3	Organization as Agent	59
2.4	Autonomy and Regulation	60
2.5	Example Scenario	62
3	Multiagent Organizations	62
3.1	Organization Concepts	64
3.2	Example of Organization Modeling: The OperA Framework	65
3.2.1	The Social Structure	68
3.2.2	The Interaction Structure	70
3.2.3	The Normative Structure	71
3.2.4	The Communication Structure	72
4	Institutions	72
4.1	Organizations, Institutions, and Norms	73
4.2	Events and States	75
4.3	Obligations, Permission, and Power	77
4.4	Example of Institutional Modeling: InstAL	78
4.4.1	The Formal Model	78
4.4.2	The Conference Scenario	78
5	Agents in Organizations	82
6	Evolution of Organizations	85
6.1	Organizational Adaptation	86
6.2	Emergent Organizations	87
7	Conclusions	88
8	Exercises	89
	References	92

Part II Communication 99

3 Agent Communication 101

Amit K. Chopra and Munindar P. Singh

1	Introduction	101
1.1	Autonomy and Its Implications	102
1.2	Criteria for Evaluation	105
2	Conceptual Foundations of Communication in MAS	106
2.1	Communicative Acts	106
2.2	Agent Communication Primitives	107
3	Traditional Software Engineering Approaches	108
3.1	Choreographies	110
3.2	Sequence Diagrams	111
3.3	State Machines	112
3.4	Evaluation with Respect to MAS	113
4	Traditional AI Approaches	114
4.1	KQML	115
4.2	FIPA ACL	116
4.3	Evaluation with Respect to MAS	117
5	Commitment-Based Multiagent Approaches	118
5.1	Commitments	118
5.2	Commitment Protocol Specification	119
5.3	Evaluation with Respect to MAS	121
6	Engineering with Agent Communication	122
6.1	Programming with Communications	123
6.2	Modeling Communications	124
6.2.1	Business Patterns	125
6.2.2	Enactment Patterns	125
6.2.3	Semantic Antipatterns	126
6.3	Communication-Based Methodologies	127
7	Advanced Topics and Challenges	128
7.1	Primacy of Meaning	128
7.2	Verifying Compliance	129
7.3	Protocol Refinement and Aggregation	129
7.4	Role Conformance	130
8	Conclusions	130
9	Exercises	133
	References	136

4 Negotiation and Bargaining 143

Shaheen Fatima and Iyad Rahwan

1	Introduction	143
2	Aspects of Negotiation	144
3	Game-Theoretic Approaches for Single-Issue Negotiation	146
3.1	Cooperative Models of Single-Issue Negotiation	147
3.2	Non-Cooperative Models of Single-Issue Negotiation	151
4	Game-Theoretic Approaches for Multi-Issue Negotiation	156
4.1	Cooperative Models of Multi-Issue Negotiation	157
4.2	Non-Cooperative Models of Multi-Issue Negotiation	157
5	Heuristic Approaches for Multi-Issue Negotiation	161
5.1	Heuristics for Generating Counteroffers	161
5.2	Heuristics for Predicting Opponent's Preferences and Generating Counteroffers	163
5.3	Heuristics for Generating Optimal Agendas	164
5.4	Heuristics for Reasoning about Deliberation Cost	164
6	Negotiating with Humans	165
7	Argumentation-Based Negotiation	167
8	Conclusions	169
9	Exercises	170
	References	171

5 Argumentation among Agents 177

Iyad Rahwan

1	Introduction	177
2	What Is an Argument?	178
2.1	Arguments as Chained Inference Rules	178
2.2	Argument as an Instance of a Scheme	180
2.3	Abstract Arguments	181
3	Evaluating an Argument	181
4	Argumentation Protocols	185
4.1	Abstract Argument Games	186
4.2	Dialogue Systems	188
5	Strategic Argumentation and Game Theory	190
5.1	Glazer and Rubinstein's Model	191
5.2	Game Theory Background	192
5.2.1	Mechanism Design	194
5.2.2	The Revelation Principle	195
5.3	Argumentation Mechanism Design	196
5.4	Case Study: Implementing the Grounded Semantics	198

6	The Argument Interchange Format	201
7	Conclusion	204
8	Exercises	205
	References	206

Part III Basic Coordination 211

6 Computational Social Choice 213

Felix Brandt, Vincent Conitzer, and Ulle Endriss

1	Introduction	213
1.1	Introductory Example	214
1.2	History of the Field	216
1.3	Applications	217
1.4	Chapter Outline	219
2	Preference Aggregation	219
2.1	Social Welfare Functions	219
2.2	Social Choice Functions	223
2.2.1	The Weak Axiom of Revealed Preference	223
2.2.2	Contraction and Expansion Consistency	224
3	Voting	226
3.1	Voting Rules	227
3.1.1	Scoring Rules	227
3.1.2	Condorcet Extensions	229
3.1.3	Other Rules	231
3.2	Manipulation	232
3.2.1	The Gibbard-Satterthwaite Impossibility	233
3.2.2	Restricted Domains of Preferences	233
3.2.3	Computational Hardness of Manipulation	235
3.2.4	Probabilistic Voting Rules	238
3.2.5	Irresolute Voting Rules	239
3.3	Possible and Necessary Winners	240
4	Combinatorial Domains	241
4.1	Preference Representation	243
4.2	Sequential Voting	245
4.3	Voting with Compactly Represented Preferences	246
5	Fair Division	247
5.1	Preference Representation	249
5.2	Fairness and Efficiency	251
5.3	Computing Fair and Efficient Allocations	253

5.4	Convergence to Fair and Efficient Allocations	255
6	Conclusion	257
6.1	Additional Topics	258
6.2	Further Reading	259
7	Exercises	260
	References	266
7	Mechanism Design and Auctions	285
	<i>Kevin Leyton-Brown and Yoav Shoham</i>	
1	Introduction	285
2	Mechanism Design with Unrestricted Preferences	286
2.1	Implementation	287
2.2	The Revelation Principle	288
2.3	Impossibility of General, Dominant-Strategy Implementation	291
3	Quasilinear Preferences	291
3.1	Mechanism Design in the Quasilinear Setting	292
4	Efficient Mechanisms	296
4.1	Groves Mechanisms	297
4.2	The VCG Mechanism	299
4.3	Properties of VCG	301
4.3.1	VCG and Individual Rationality	301
4.3.2	VCG and Weak Budget Balance	302
4.3.3	Drawbacks of VCG	302
4.4	Budget Balance and Efficiency	303
5	Single-Good Auctions	304
5.1	Canonical Auction Families	305
5.1.1	English Auctions	305
5.1.2	Japanese Auctions	305
5.1.3	Dutch Auctions	306
5.1.4	Sealed-Bid Auctions	306
5.2	Auctions as Bayesian Mechanisms	306
5.3	Second-Price, Japanese, and English Auctions	308
5.4	First-Price and Dutch Auctions	310
5.5	Revenue Equivalence	311
6	Position Auctions	312
7	Combinatorial Auctions	315
8	Conclusions	318
9	Exercises	320
	References	325

8 Computational Coalition Formation 329

Edith Elkind, Talal Rahwan, and Nicholas R. Jennings

1	Introduction	329
1.1	Coalitional Games: A Bird's Eye View	330
2	Definitions	331
2.1	Outcomes	332
2.2	Subclasses of Characteristic Function Games	333
2.2.1	Monotone Games	333
2.2.2	Superadditive Games	333
2.2.3	Convex Games	334
2.2.4	Simple Games	335
3	Solution Concepts	335
3.1	Shapley Value	335
3.2	Banzhaf Index	337
3.3	Core	338
3.3.1	The Core of Simple Games	339
3.3.2	The Core of Convex Games	340
3.4	The Least Core	341
3.5	Other Solution Concepts	342
4	Representation Formalisms	343
4.1	Weighted Voting Games	344
4.1.1	Computational Issues	345
4.1.2	Expressivity and Vector Weighted Voting Games	346
4.2	Combinatorial Optimization Games	348
4.2.1	Induced Subgraph Games	348
4.2.2	Network Flow Games	348
4.2.3	Matching and Assignment Games	349
4.3	Complete Representation Languages	349
4.3.1	Marginal Contribution Nets	349
4.3.2	Synergy Coalition Groups	350
4.3.3	Skill-Based Representations	351
4.3.4	Agent-Type Representation	351
5	Coalition Structure Generation	352
5.1	Space Representation	353
5.2	Dynamic Programming Algorithms	354
5.3	Anytime Algorithms	356
5.3.1	Identifying Subspaces with Worst-Case Guarantees	356
5.3.2	Integer Partition-Based Search	359
5.3.3	Integer Programming	360

5.4	Metaheuristic Algorithms	361
5.5	Coalition Structure Generation under Compact Representations	362
5.5.1	Distributed Constraint Optimization	362
5.5.2	Marginal Contribution Nets	364
5.5.3	Coalitional Skill Games	366
5.5.4	Agent-Type Representation	368
5.6	Constrained Coalition Formation	369
6	Conclusions	372
7	Exercises	372
	References	374
9	Trust and Reputation in Multiagent Systems	381
	<i>Jordi Sabater-Mir and Laurent Vercouter</i>	
1	Introduction	381
2	Computational Representation of Trust and Reputation Values	382
2.1	Boolean Representation	383
2.2	Numerical Values	383
2.3	Qualitative Labels	384
2.4	Probability Distribution and Fuzzy Sets	384
2.5	Trust and Reputation as Beliefs	385
2.6	The Reliability of a Value	387
3	Trust Processes in Multiagent Systems	388
3.1	General Overview of Trust-Related Processes	388
3.2	Trust Evaluations	390
3.2.1	Filtering the Inputs	391
3.2.2	Statistical Aggregation	392
3.2.3	Logical Beliefs Generation	393
3.3	Trust Decision	394
3.3.1	Single Trust Values and Probability Distributions	395
3.3.2	Trust Beliefs	395
3.4	Coping with the Diversity of Trust Models	396
4	Reputation in Multiagent Societies	396
4.1	Reputation-Building Process	398
4.1.1	Communicated Image as a Source for Reputation	398
4.1.2	Communicated Reputation	400
4.1.3	Inherited Reputation	400
4.1.4	Putting It All Together	401

4.2	Centralized vs. Decentralized Models	402
4.2.1	Centralized Approaches	402
4.2.2	Decentralized Approaches	403
4.3	Using Reputation	404
4.3.1	Reputation as a Source of Trust	404
4.3.2	Reputation for Social Order	405
4.4	Pitfalls When Using Reputation	405
4.4.1	Unfair Ratings	405
4.4.2	Ballot-Stuffing	406
4.4.3	Dynamic Personality	406
4.4.4	Whitewashing	406
4.4.5	Collusion	406
4.4.6	Sybil Attacks	407
4.4.7	Reputation Lag Exploitation	407
5	Trust, Reputation, and Other Agreement Technologies	407
5.1	Argumentation	408
5.2	Negotiation	410
5.3	Norms	410
5.4	Organizations	411
5.5	Ontologies and Semantics	411
6	Conclusions	413
7	Exercises	414
	References	415

Part IV Distributed Cognitive Abilities 421

10 Multiagent Learning 423

Karl Tuyls and Kagan Tumer

1	Introduction	423
2	Challenges in Multiagent Learning	425
2.1	State, Action, and Outcome Space Problems	426
2.2	Multiagent Credit Assignment Problem	426
2.3	Agent Rewards and System Dynamics	427
2.4	Two Simple Multiagent Learning Paradigms	429
2.4.1	Action-Value Learning	430
2.4.2	Direct Policy Adjustment	431
3	Reinforcement Learning for Multiagent Systems	432
3.1	Markov Decision Processes	433
3.2	Action Selection and Exploration-Exploitation Dilemma	434

3.3	Model-Free and Model-Based Approaches	435
3.4	Multiagent MDP Formulations	437
3.5	Markov Games	438
3.6	State-of-the-Art Algorithms	439
3.6.1	Joint Action Learning	439
3.6.2	Nash-Q Learning	440
3.6.3	Gradient Ascent Algorithms	441
3.6.4	Other Approaches	442
4	Evolutionary Game Theory as a Multiagent Learning Paradigm . .	443
4.1	Matrix Games	443
4.2	Solution Concepts	444
4.3	Evolutionary Stable Strategies	445
4.4	Replicator Dynamics	446
4.5	The Role of Game Theory for Multiagent Learning	447
4.6	Evolutionary Game Theory as a Theoretical Framework .	448
5	Swarm Intelligence as a Multiagent Learning Paradigm	451
5.1	Ant Colony Optimization	453
5.2	Bee Colony Optimization	455
6	Neuro-Evolution as a Multiagent Learning Paradigm	457
6.1	Evolutionary Algorithm Basics	458
6.2	Linking Multiagent Reinforcement Learning to the Neuro-Evolutionary Approach	460
7	Case Study: Air Traffic Control	460
7.1	Motivation	460
7.2	Simulation and System Performance	461
7.3	Agent-Based Air Traffic	463
7.4	Multiagent Air Traffic Results	466
7.5	Summary	467
8	Conclusions	468
9	Exercises	468
	References	475

11 Multiagent Planning, Control, and Execution 485

Ed Durfee and Shlomo Zilberstein

1	Introduction	485
2	Characterizing Multiagent Planning and Control	487
3	Coordination Prior to Local Planning	488
3.1	Social Laws and Conventions	489
3.2	Organizational Structuring	490
3.2.1	Organizational Design	491

	3.2.2	Organizational Execution and Functionally-Accurate Cooperation	493
	3.3	The Contract-Net Protocol and Role Assignment	495
4		Local Planning Prior to Coordination	497
	4.1	State-Space Techniques	498
	4.2	Plan-Space Techniques	499
	4.2.1	Single-Agent Plans	499
	4.2.2	Multiagent Plans	501
	4.2.3	Multiagent Plan Coordination by Plan Modification	505
	4.3	Hierarchical Multiagent Plan Coordination	510
5		Decision-Theoretic Multiagent Planning	512
	5.1	Models for Decision-Theoretic Multiagent Planning	513
	5.1.1	Solution Representation and Evaluation	515
	5.1.2	The Complexity of DEC-POMDPs	518
	5.2	Solving Finite-Horizon DEC-POMDPs	519
	5.3	Solving Infinite-Horizon DEC-POMDPs	522
	5.3.1	Correlated Joint Controllers	523
	5.3.2	Policy Iteration for Infinite-Horizon DEC-POMDPs	524
	5.3.3	Optimizing Fixed-Size Controllers Using Non-Linear Programming	526
6		Multiagent Execution	527
	6.1	Multiagent Plan Monitoring	527
	6.2	Multiagent Plan Recovery	528
	6.3	Multiagent Continuous Planning	529
7		Conclusions	532
8		Exercises	533
		References	539

12 Distributed Constraint Handling and Optimization 547

*Alessandro Farinelli, Meritxell Vinyals, Alex Rogers, and
Nicholas R. Jennings*

1		Introduction	547
2		Distributed Constraint Handling	549
	2.1	Constraint Networks	549
	2.2	Distributed Constraint Processing	550
3		Applications and Benchmarking Problems	551
	3.1	Real-World Applications	551
	3.1.1	Meeting Scheduling	552

	3.1.2	Target Tracking	553
	3.2	Exemplar and Benchmarking Problems	553
4		Solution Techniques: Complete Algorithms	554
	4.1	Search-Based: ADOPT	555
	4.2	Dynamic Programming: DPOP	561
5		Solution Techniques: Approximate Algorithms	565
	5.1	Local Greedy Approximate Algorithms	565
	5.1.1	The Distributed Stochastic Algorithm	566
	5.1.2	The Maximum Gain Message Algorithm	567
	5.2	GDL-Based Approximate Algorithms	568
	5.2.1	The Max-Sum Algorithm	568
6		Solution Techniques with Quality Guarantees	570
	6.1	Off-line Guarantees	571
	6.2	Online Guarantees	574
7		Conclusions	577
8		Exercises	578
		References	580

Part V Development and Engineering 585

13 Programming Multiagent Systems 587

Rafael H. Bordini and Jürgen Dix

1		Introduction	587
	1.1	Relation to Other Chapters	589
	1.2	Organization of This Chapter	589
2		From AGENT0 to Modern Agent Languages	590
	2.1	A Brief History of Agent-Oriented Programming (AOP)	590
	2.2	Features of Multiagent-Oriented Programming (MAOP)	591
3		Abstractions in the MAOP Paradigm	593
	3.1	Agent Level	593
	3.2	Environment Level	595
	3.3	Social Level	596
4		Examples of Agent Programming Languages	596
	4.1	JASON	596
	4.1.1	Beliefs	597
	4.1.2	Goals	598
	4.1.3	Plans	599
	4.1.4	Semantics	600
	4.2	Other BDI-Based Languages	601

4.3	Approaches Based on Executable Logics	603
4.3.1	METATEM	607
4.3.2	ConGolog and IndiGolog	608
5	Organization and Environment Programming	609
5.1	Organizations	609
5.1.1	MOISE	610
5.1.2	Other Approaches	611
5.2	Environments	611
5.2.1	CARTAGO	613
5.2.2	EIS	617
6	Example of Full MAOP in JaCaMo	620
6.1	The Application Scenario	621
6.2	Organization Program	623
6.3	Agent Programs	625
6.4	Environment Program	628
7	Conclusions	629
8	Exercises	630
	References	633

14 Specification and Verification of Multiagent Systems 641

Jürgen Dix and Michael Fisher

1	Introduction	641
1.1	Why Logic, Specification, and Verification?	642
1.2	Limits and Relation to Other Chapters	644
1.3	Organization of This Chapter	644
2	Agent Specification	644
2.1	Logics of Agency and Specification Languages	644
2.2	Approaches Based on Temporal Logics	646
2.2.1	LTL	648
2.2.2	CTL and CTL*	649
2.2.3	ATL and ATL*	650
2.3	Approaches Based on Dynamic Logic	652
2.4	Combinations	653
2.4.1	BDI	653
2.4.2	KARO	653
2.4.3	Dynamic Epistemic Logic	654
2.5	Sample Specifications	654
3	From Specifications to Implementations	656
3.1	Toward Formal Verification	656
3.2	Refinement	657

3.3	Synthesis	657
3.4	Specifications as Programs	658
4	Formal Verification	659
4.1	What Is Formal Verification?	659
4.2	Deductive Verification	660
4.3	Algorithmic Verification	660
4.4	Program Verification	662
4.5	Runtime Verification	663
5	Deductive Verification of Agents	663
5.1	The Problem	664
5.2	Direct Proof	665
5.3	Use of Logic Programming	666
5.4	Example	667
6	Algorithmic Verification of Agent Models	667
6.1	The Representation and Size of the Model	668
6.2	(Im-)Perfect Information, (Im-)Perfect Recall	669
6.3	Modular Interpreted Systems	671
6.4	MC Complexity for LTL, CTL, ATL, and MIS	672
6.5	Model Checking Agent Language Models	674
7	Algorithmic Verification of Agent Programs	676
7.1	General Problem	676
7.2	AIL Semantic Toolkit	677
7.3	Multiple Semantic Definitions	678
7.4	Model Checking AIL Through MCAPL/AJPF	679
7.5	Example	679
8	Conclusions	680
9	Exercises	681
	References	683

15 Agent-Oriented Software Engineering 695

Michael Winikoff and Lin Padgham

1	Introduction	695
1.1	History of AOSE	698
2	Agent Concepts	700
3	Running Example	702
4	Requirements	704
5	Design	710
6	Detailed Design	717
6.1	Example Design: BDI Platform	719
6.1.1	Initial Structure	719

6.1.2	Subgoal Structure for build2	720
6.1.3	Subgoal Structure for addPart and complete . .	720
6.1.4	Finalizing the Design	722
6.1.5	Multiple Plans	724
6.1.6	Control Information	725
6.2	Example Design: Finite-State Automaton	725
6.3	Final Features	726
7	Implementation	727
8	Assurance	728
8.1	Testing and Debugging	729
8.2	Formal Methods	732
9	Software Maintenance	734
10	Comparing Methodologies	735
11	Conclusions	736
12	Exercises	740
	Appendix: Agent UML Sequence Diagram Notation	742
	References	744

Part VI Technical Background 759

16 Logics for Multiagent Systems 761

Wiebe van der Hoek and Michael Wooldridge

1	Introduction	761
1.1	A Logical Toolkit	764
2	Representing Cognitive States	769
2.1	Intention Logic	770
2.2	BDI Logic	773
2.3	Discussion	778
2.4	Cognitive Agents in Practice	778
2.4.1	Specification Language	778
2.4.2	Implementation	780
2.4.3	Verification	782
3	Representing the Strategic Structure of a System	785
3.1	Coalition Logic	786
3.2	Strategic Temporal Logic: ATL	790
3.3	Knowledge in Strategic Temporal Logics: ATEL	794
3.4	CL-PC	796
4	Conclusion and Further Reading	797

5	Exercises	799
	References	800
17	Game-Theoretic Foundations of Multiagent Systems	811
	<i>Edith Elkind and Evangelos Markakis</i>	
1	Introduction	811
2	Normal-Form Games	812
2.1	Dominant Strategy	814
2.2	Nash Equilibrium	816
2.3	Mixed Strategies and Mixed Nash Equilibrium	818
2.4	Elimination of Dominated Strategies	820
2.5	Games with Infinite Action Spaces	821
2.5.1	Games with Differentiable Payoff Functions	823
2.6	Zero-Sum Games	824
2.7	Computational Aspects	827
3	Extensive-Form Games	828
3.1	Nash Equilibrium and Critiques	831
3.2	Subgame-Perfect Equilibrium	832
3.3	Backward Induction	834
4	Bayesian Games	836
4.1	Two Examples	837
4.2	Formal Definitions	840
5	Conclusions	842
6	Exercises	842
	References	847
	Subject Index	849

List of Figures

1.1	An agent in its environment.	5
1.2	Action selection in deliberate agents.	16
1.3	Vacuum world.	17
1.4	The value iteration algorithm for Markov decision processes.	27
1.5	Schematic diagram of a generic belief-desire-intention architecture.	32
1.6	Information and control flows in three types of layered agent architectures (Source: [41, p. 263]).	37
1.7	TOURINGMACHINES: a horizontally layered agent architecture.	38
1.8	INTERRAP: a vertically layered two-pass agent architecture.	40
2.1	Conference management system.	63
2.2	An excerpt of the organization ontology.	66
2.3	The OperA development process.	67
2.4	Role dependencies in a conference.	69
2.5	Landmark pattern for <i>review process</i> .	71
2.6	Interaction structure in the conference scenario.	71
2.7	Relating institutions, norms, and organizations.	74
2.8	An event-based view of institutions.	76
2.9	An event-based formal model of institutions.	79
2.10	Conference types, predicates, and events.	80
2.11	Institutional events changing institutional state.	80
2.12	Reviewer assignment.	81
2.13	A visualization of a possible answer set trace.	82
2.14	Implicit and explicit organizations.	86
3.1	Updating an offer.	105
3.2	FIPA request interaction protocol.	112
3.3	A protocol specified as a state machine.	113
3.4	An alternative, more flexible state machine.	114
3.5	Distinguishing message syntax and meaning: two views of the same enactment.	120

3.6	Flexible enactment.	122
3.7	Example of operational patterns.	124
4.1	A Boulware, a linear, and a Conceder strategy.	162
4.2	Colored Trails user interface example.	167
4.3	Dialogue between agents i (black) and j (gray).	168
5.1	A simple argument graph.	181
5.2	Graph with three labelings/complete extensions.	183
5.3	Argumentation framework and dispute tree.	186
5.4	Hiding an argument is beneficial.	199
5.5	Argument graph (see Example 5.11).	201
5.6	Examples of simple arguments.	203
6.1	Condorcet's paradox [86].	221
6.2	Example due to Fishburn [118], which shows that no scoring rule is a Condorcet extension.	229
7.1	Transportation network with selfish agents.	286
7.2	The revelation principle: how to construct a new mechanism with a truthful equilibrium, given an original mechanism with equilibrium (s_1, \dots, s_n) .	289
7.3	Transportation network with selfish agents.	300
7.4	A case analysis to show that honest bidding is a dominant strategy in a second-price auction with independent private values.	309
8.1	The coalition structure graph for four agents.	353
8.2	The integer partition-based representation for four agents.	354
8.3	A skill graph and its tree decomposition with width 2.	366
8.4	Feasible coalitions and coalition structures.	371
9.1	Representing reputation as a probability distribution.	385
9.2	Representing reputation using fuzzy sets.	385
9.3	Reputation as beliefs in the BDI+RepAge model.	387
9.4	The dual nature of trust.	389
9.5	Reputation evaluation.	399
9.6	Example of argument for a reputation value.	408
10.1	Basic reinforcement learning scheme.	433
10.2	General form of repeated matrix game.	440
10.3	Examples of matrix games.	444
10.4	Visualization of the replicator dynamics of BoS, PD, and MP games.	447

10.5	Overview of the evolutionary dynamics of the studied learning methods.	451
10.6	Policy trajectories of FAQ and LFAQ in three different games.	452
10.7	Ant foraging.	454
10.8	Path integration.	456
10.9	Distance and direction by wagging dance.	456
10.10	Graphical user interface of the FACET simulator.	462
10.11	Performance of agents controlling miles in trail on the congestion problem, with 1,000 aircraft, 20 agents, and $\alpha = 5$.	466
10.12	Scaling properties of agents controlling miles in trail on the congestion problem, with $\alpha = 5$.	466
10.13	Performance of agents controlling miles in trail on historical data from New York area.	467
10.14	The subsidy matrix game.	469
10.15	Subsidy game instances.	470
10.16	The dynamics of the game without subsidy (left) and with subsidy (right).	470
10.17	Repeated matrix game (general form).	471
10.18	The climbing game.	471
11.1	Initial state for simple blocks world problem.	501
11.2	Single POCL plan for stacking block A on block B.	502
11.3	Single POCL plan for stacking block B on block C.	502
11.4	Initial (inconsistent) multiagent parallel POCL plan.	505
11.5	Ordering constraint added to resolve causal-link threat.	506
11.6	Solution multiagent parallel POCL plan.	509
11.7	Illustration of a two-agent DEC-POMDP.	513
11.8	Relationship among several models for multiagent planning.	514
11.9	Optimal policy trees for the multiagent tiger problem with horizons 1–4.	516
11.10	Optimal policy tree for the multiagent tiger problem with horizon 5.	516
11.11	Optimal one-node and two-node deterministic controllers for the multiagent tiger problem.	517
11.12	Optimal three-node deterministic controllers for multiagent tiger.	517
11.13	Optimal four-node deterministic controllers for multiagent tiger.	518
11.14	Stochastic two-node controllers for multiagent tiger.	518
11.15	A set of <i>maxTrees</i> policy tree can be represented compactly by reusing a fixed number of <i>maxTrees</i> subpolicies of the previous level.	522

11.16	A slice of a two-agent DEC-POMDP where actions are selected based on internal states q_i with (right) and without (left) a correlation device q_c .	524
12.1	Exemplar constraint network.	557
12.2	Message exchange in the ADOPT algorithm: <i>Value</i> and <i>Cost</i> messages.	557
12.3	Message exchange in the ADOPT algorithm: <i>Threshold</i> messages.	559
12.4	Exemplar constraint network showing induced DFS graphs.	562
12.5	Message exchange in DPOP.	563
12.6	Message exchange in max-sum.	569
12.7	Loopy constraint network.	575
12.8	Tree-like constraint network formed by BMS.	575
13.1	A goal-plan tree for a Mars rover scenario [64].	597
13.2	The abstract architecture of JADEx.	602
13.3	A screenshot of the 2APL platform.	603
13.4	The architecture of AGENTFACTORY.	604
13.5	The architecture of BRAHMS.	604
13.6	The structure of a module in GOAL.	605
13.7	The contract-net protocol.	606
13.8	Ideal world: Heterogenous agents acting within a shared environment.	612
13.9	Environment support levels [71].	614
13.10	A bakery used as a metaphor to frame the notion of workspaces and artifacts as resources and tools used by agents to work.	615
13.11	CARTAGO meta-model based on A&A.	615
13.12	Java-based implementation of a <i>Counter</i> artifact type having one operation <i>inc</i> and one observable property <i>count</i> .	616
13.13	Snippets of two JASON agents creating and using a shared counter (on the left) and observing the shared counter (on the right).	616
13.14	Components of EIS.	618
13.15	Environment-MAS model.	619
13.16	MOISE organization: Structural specification.	624
13.17	MOISE organization: Functional specification.	624
14.1	Two robots and a carriage: a schematic view (left) and a transition system \mathcal{M}_0 that models the scenario (right).	647
14.2	Two robots and a carriage: a refined version of our example and a concurrent game structure (CGS).	651
14.3	Automata-theoretic view of model checking.	661

14.4	“On-the-fly” exploration of the product automaton.	662
14.5	A general view of run-time model checking.	664
14.6	Two robots and a carriage: a schematic view (left) and an imperfect information concurrent game structure \mathcal{M}_2 that models the scenario (right).	670
14.7	AJPF architecture [29].	678
14.8	AJPF “On-the-fly” exploration.	679
15.1	A brief history of AOSE.	698
15.2	Relationship between properties and supporting concepts.	700
15.3	Assembly cell layout.	703
15.4	Scenario steps.	706
15.5	Simple goal model.	708
15.6	Lock protocol.	713
15.7	Top-level protocol to manufacture a part.	714
15.8	AddPart protocol.	715
15.9	System overview diagram.	716
15.10	Fasten protocol.	718
15.11	Initial step in design development for Robot1 internals.	720
15.12	Developing the build2Plan for Robot1 internals.	721
15.13	Developing the addPartPlan and completePlan for Robot1 internals.	722
15.14	Agent overview design diagram for Robot1 internals.	723
15.15	Behavior of Robot1 (finite-state automaton a la O-MaSE).	726
16.1	An inference system for knowledge.	764
16.2	Semantics and axioms of linear temporal logic.	766
16.3	A branching-time model.	767
16.4	Two games G_1 and G_2 that are the same in terms of effectivity.	788
16.5	The axioms and inference rules of coalition logic.	789
17.1	An extensive-form game with two players.	828
17.2	Cupcake division game.	830
17.3	An extensive-form game.	835

List of Tables

2.1	Organizational forms according to Mintzberg.	57
2.2	<i>PC member</i> role description.	68
2.3	Script for the <i>review process</i> scene.	70
3.1	A commitment protocol.	120
3.2	Comparison of agent communication approaches.	122
4.1	Prisoner's Dilemma game.	147
5.1	The relationships between extensions and labelings.	185
5.2	An example L_c in Prakken's framework [35].	189
5.3	Abstract argumentation as a mechanism.	198
8.1	Sample characteristic function given five agents.	373
11.1	The NLP defining a set of optimal fixed-size DEC-POMDP controllers.	526
11.2	Tiger observation, transition, and reward tables.	538
13.1	Comparison of some agent platforms.	617
13.2	MOISE organization: Normative specification.	625
14.1	Overview of the complexity results: most are completeness results.	673
16.1	Atomic modalities in Cohen and Levesque's logic.	772
16.2	Systems of BDI logic.	777
17.1	The payoff matrix for the Prisoner's Dilemma.	814
17.2	The payoff matrix for the Battle of the Sexes.	815
17.3	The payoff matrix for the Matching Pennies.	817
17.4	Elimination of strictly dominated strategies: two examples.	821
17.5	An example of a Bayesian game.	837
17.6	Payoffs in a Bayesian game.	838

Preface

Gerhard Weiss

The Subject of This Book

Multiagent systems are systems composed of multiple interacting intelligent agents. An agent is a computational entity such as a software program or a robot that is situated in some environment and that to some extent is able to act autonomously in order to achieve its design objectives. As interacting entities, agents do not simply exchange data but are actively engaged in cooperative and competitive scenarios; they may communicate on the basis of semantically rich languages, and they achieve agreements and make decisions on the basis of processes such as negotiation, argumentation, voting, auctioning, and coalition formation. As intelligent entities, agents act flexibly, that is, both reactively and deliberatively, in a variety of environmental circumstances on the basis of processes such as planning, learning, and constraint satisfaction. As autonomous entities, agents have far-reaching control over their behavior within the frame of their objectives, possess decision authority in a wide variety of circumstances, and are able to handle complex and unforeseen situations on their own and without the intervention of humans or other systems. And as entities situated in some environment, agents perceive their environment at least partially and act upon their environment without being in full control of it. Concrete multiagent systems and their environments have several relevant attributes in which they can differ; the table below (on page xxxvi) indicates the wide range of possible instantiations of these attributes.

Since its inception in the late 1970s, the field of multiagent systems has evolved impressively and today it is an established and vibrant field in computer science. The field has a profound and broad conceptual and theoretical foundation, drawing on and bringing together results, techniques, and tools not only from computer science and artificial intelligence (which traditionally has dealt

	attribute	range
agents	number	from two upward
	uniformity	homogeneous ... heterogeneous
	goals	contradicting ... complementary
	flexibility	purely reactive ... purely deliberative
	abilities (sensors, effectors, cognition)	simple ... advanced
	autonomy	low ... high
interaction	frequency	low ... high
	persistence	short-term ... long-term
	level	signal-passing ... knowledge-intensive
	language	elementary ... semantically rich
	pattern (flow of data and control)	decentralized ... hierarchical
	variability	fixed ... changeable
	purpose	competitive ... cooperative
environment	predictability	forseeable ... unforeseeable
	accessibility and knowability	unlimited ... limited
	dynamics	fixed ... variable
	diversity	poor ... rich
	availability of resources	restricted ... ample

Variety of multiagent systems and their environments.

with single-agent systems) but also from mathematics, logics, game theory, and other areas. The multiagent systems field is multidisciplinary in nature. Examples of disciplines to which the field is related are cognitive psychology, sociology, organization science, economics, and philosophy.

A main reason for the vast interest and attention multiagent systems are receiving is that they are seen as an enabling technology for applications that rely on distributed and parallel processing of data, information, and knowledge in complex – networked, open, and large-scale – computing environments. With advancing technological progress in interconnectivity and interoperability of computers and software, such applications are becoming standard in a variety of domains such as e-commerce, logistics, supply chain management, telecommunication, health care, and manufacturing. More generally, such applications are characteristic of several widely recognized computing paradigms known as grid computing, peer-to-peer computing, pervasive computing, ubiquitous computing, autonomic com-

puting, service-oriented computing, and cloud computing. In a sense multiagent technology is complementary to and cuts across these paradigms. Another reason for the broad interest in multiagent systems is that these systems are seen as a technology and tool that helps to analyze and develop models and theories of interactivity in large-scale human-centered systems. Research motivated by this interest is mainly based on computer-based experimental analysis and simulation rather than mathematical studies. The general goal of this line of research is to gain a deeper understanding of complex organizational, societal, economic, and political phenomena (including their underlying mechanisms and observable dynamics) for which the individual abilities and traits of the involved humans are crucial and thus cannot be neglected.

Main Features of This Book

The book offers several features that support its use as a textbook and reference volume:

- *Scope* – It captures the state of the art in the field in breadth and depth.
- *Theory* – It conveys the theoretical foundations and underpinning of multiagent systems.
- *Clarity* – It provides many illustrations and examples.
- *Practice* – It includes many exercises of varying degrees of difficulty.
- *Expertise* – Its chapters are written by leading experts and authorities in the field.

It is worth saying a few words about the last-mentioned feature. This feature ensures that the book is built on an outstanding, broad, and profound basis of knowledge and experience. As the readers will also see, tremendous effort has been invested in carefully coordinating the individual chapters and in ensuring overall coherence of the book. In a way, there is no approach to writing a book on *multiagent* systems that is more natural and obvious than the *multi-author* approach taken for this book. A list of the thirty-one contributing authors is provided on pages xliii–xlvi.

Readership and Prerequisites

The book is primarily intended for use in undergraduate, graduate, and postgraduate courses and is also suited for self-study. The main academic audiences are *stu-*

dents and teachers of artificial intelligence, computer science, information technology, and related fields. Because of the multidisciplinary nature of multiagent systems (both as a technology and a field), it can also serve as a course text for students and teachers from disciplines such as psychology, economics, sociology, and philosophy. Moreover, because of its breadth and depth, the book can serve as a basic reference volume for both *researchers* who want to branch out beyond their own subfields and *professionals from industry* who want to explore potential usages of multiagent technology in their application areas.

As far as possible, the chapters are written so they can be understood without advanced prior knowledge. The main prerequisite for making the most of the book and for understanding its contents in detail is familiarity with basic concepts of computer science (especially algorithms and programming) and mathematics (especially logics and game theory) at the freshman level. Some useful background in logics and game theory is supplied in Part VI of this book.

Changes from the First Edition

The first edition appeared in 1999, and since then the field of multiagent systems has developed considerably. Some topics and themes that were characteristic of the field some twelve years ago play only a minor role today, and some of today's core topics played no or only a marginal role at that time. This second edition captures all these changes and shows the current state of the art in the field. Much more work was necessary in creating the new edition other than just rewriting or restructuring some of the first-edition chapters. Only one of the seventeen chapters was already included in the first edition (some passages of the chapter have been rewritten and some material has been added); the other sixteen chapters are entirely new.

What remained *unchanged* is the unique conception and vision behind the book: to have a high-quality course book and reference volume on multiagent systems whose parts are all written by acknowledged authorities in the field.

Structure and Chapters

This book is divided into six parts:

- Part I introduces basic concepts and principles of computational agency and covers key issues of both individual agents (Chapter 1) and agent organizations (Chapter 2).

- Part II focuses on communication among agents and discusses agent communication languages (Chapter 3) as well as two forms of agent-agent interaction – negotiation and bargaining (Chapter 4) and argumentation (Chapter 5) – for which communication is particularly crucial.
- Part III focuses on coordination among agents from different perspectives, including social choice (Chapter 6), mechanism design and auctions (Chapter 7), coalition formation (Chapter 8), and trust and reputation (Chapter 9).
- Part IV focuses on distributed cognition in multiagent systems and deals with several basic cognitive abilities, namely, learning (Chapter 10), planning and decision making (Chapter 11), and constraint handling and optimization (Chapter 12).
- Part V focuses on the development and engineering of multiagent systems and deals with programming (Chapter 13), specification and verification (Chapter 14), and agent-oriented software engineering (Chapter 15).
- Part VI provides relevant and useful background knowledge in logics (Chapter 16) and game theory (Chapter 17).

Each chapter starts with a motivating introduction, then delves into its topic, and concludes with considerations on the current state of the art, open challenges, and latest developments. The chapters provide a number of pointers to relevant literature and put particular emphasis on providing examples and illustrations. Moreover, each chapter comes with various exercises.

The Exercises

At the end of each chapter, exercises of varying difficulty are provided, which concern relevant theoretical and practical aspects of multiagent systems. The following four levels of difficulty are distinguished to roughly indicate the amount of effort required for solving the exercises:

- **Level 1** Simple test of comprehension or slightly more subtle problem, solvable within a few hours or days. (Appropriate for Bachelor education)
- **Level 2** Much harder problem (e.g., requires writing a non-trivial program); solving it could take several days or weeks. (Bachelor)
- **Level 3** Even harder problem, typically related to a “hot” topic of current research; solving it could take weeks or months. (Bachelor/Master)

- **Level 4** An open research question; solution cannot be expected within a few months or could even be a topic of a PhD. (Master/PhD)

I recommend addressing as many Level-1 and Level-2 exercises as possible and dealing with at least a few of the Level-3 and Level-4 exercises. Most of the Level-1 and Level-2 exercises can be solved with the knowledge provided in this book, whereas Level-3 and Level-4 exercises typically require additional literature studies and extensive theoretical and/or experimental studies. Carefully working through Level-1 and Level-2 exercises will reward a reader with a real understanding of the material treated in the chapters, and solving Level-3 and Level-4 exercises will turn a reader into a real expert!

How to Use This Book

The book can be used for teaching as well as self-study. The chapters and thus the overall book are designed to be self-contained and understandable without additional material. Of course, there are many relationships between the chapters, but in principle they can be treated independently and read in any sequence. In general, I recommend starting off with Part I (i.e., Chapters 1 and 2) in order to set up a proper contextual understanding, especially if the reader is new to this field.

All chapters together can easily fill two one-semester courses. There are several ways to use the book. One possibility is to work linearly through the book from front to back, thereby covering each of the individual chapters fully or just partially. Another possibility, resulting from the self-containment of the chapters, is to tailor a course to specific needs and interests by using only some selected chapters in any preferable order while dropping the others. The book can also be employed as a *complementary text for courses on “classical” (single-agent) AI* in order to selectively cover elements of multi-agency. For instance, if such a course deals with machine learning, then an obvious option is to include also one or another multiagent learning algorithm. Last but not least, the chapters in this book can be used as *supportive material for specialized courses* on topics such as electronic auctions, smart devices, cooperative information systems, and autonomous systems engineering.

The book contains a number of exercises that allow readers to test and further deepen their knowledge. Course instructors may find them helpful and inspiring in view of formulating review questions, in-class exercises, homework problems, or course exams. Some exercises are fairly simple and are intended to make sure that basic material provided in the chapters is mastered. Others are more difficult and challenging and may serve as subjects of class discussion or advanced team work.

Throughout the book numerous references to relevant literature are provided. They enable interested students to further explore specific aspects, and they support teachers in choosing additional course material.

Slides and More – The Website of the Book

This book is accompanied by the website accessible via

- <http://mitpress.mit.edu/multiagentsystems>

The site is intended to provide useful teaching material for students and teachers. The website starts with lecture slides for the chapters (prepared by the respective chapter authors) and some other resources such as copies of the figures and a list of exercises in the book. I hope to extend the supplementary material once this book is in use.

Teachers, students, and industrial professionals are invited and encouraged to contribute additional resources based on the contents of the book. Examples of such additional resources are

- exercises, lab projects, and exam papers,
- alternative slides and lecture videos,
- descriptions/syllabi of courses employing this book, and
- errors slipped in the book (errata).

Teachers using the book are asked to notify me of their courses (with URLs). I will maintain an online list of these courses. The material I receive will be made available on the website so that all readers and the multiagent community as a whole can benefit from it. Additional resources and related questions can be mailed to gerhard.weiss@maastrichtuniversity.nl

Acknowledgments

This book has been developing for nearly two years (taking into account that the 17 chapters had been prepared in parallel by different authors, this amounts to a total development period of 34 years invested in this book!). During that time many people have contributed to the book project. I cannot thank all of them, and so below I mention those to whom I am particularly indebted. Please also see the acknowledgments at the end of the individual chapters.

I am most grateful to the contributing authors for their engagement and enthusiasm. They not only provided the chapters and chapter slides, but also gave many

useful comments and suggestions on how the overall quality of the book could be further improved. In particular, they all put tremendous effort in carefully coordinating the contents of their chapters. Developing a textbook like this, with 31 leading experts in the field being involved as the chapter authors, is an endeavor that is thrilling and appealing on the one hand but can easily fail for many reasons on the other. Obviously this book project did not fail, and in large part this is due to the professionalism and commitment of the involved authors. I enjoyed a lot working with them on the new edition. Here is my advice to all who think about editing a comparable multi-author textbook: do it, but only if you have such a strong author team behind you as I had.

My special thanks also goes to the authors of the first edition: their excellent chapters laid the foundation for this second edition.

I want to thank Tuomas Sandholm for valuable discussions in the early phase of this book project on possibilities for covering certain key topics in the best possible way.

At MIT Press, I am grateful to Marc Lowenthal, James DeWolf and Virginia Crossman for providing professional guidance, assistance, and support during this book project whenever necessary. Ada Brunstein provided great support in the start-up phase of the project. Thanks also to the anonymous reviewers provided by MIT Press for their useful remarks and comments.

I “misused” numerous family-time evenings and weekends to work on this book project – my warmest thanks go to Tina, Alina, and Sofie for their patience and understanding.

Contributing Authors

Rafael H. Bordini

Faculty of Informatics

PUCRS - Pontifical Catholic University of Rio Grande do Sul

Porto Alegre, RS, Brazil

⇒ *Chapter 13*

Felix Brandt

Institut für Informatik

Technische Universität München

Munich, Germany

⇒ *Chapter 6*

Amit K. Chopra

Department of Information Engineering and Computer Science

University of Trento

Trento, Italy

⇒ *Chapter 3*

Vincent Conitzer

Department of Computer Science

Duke University

Durham, NC, USA

⇒ *Chapter 6*

Michael Winikoff

Department of Information Science
University of Otago
Dunedin, New Zealand

⇒ *Chapter 15*

Michael Wooldridge

Department of Computer Science
University of Oxford
Oxford, England

⇒ *Chapters 1 and 16*

Shlomo Zilberstein

Department of Computer Science
University of Massachusetts
Amherst, MA, USA

⇒ *Chapter 11*

Lead authors of the chapters with several authors:

Chapter 2	Virginia Dignum
Chapter 3	Munindar Singh
Chapter 4	Iyad Rahwan
Chapter 6	Felix Brandt
Chapter 7	Kevin Leyton-Brown
Chapter 8	Talal Rahwan
Chapter 9	Jordi Sabater-Mir
Chapter 10	Karl Tuyls
Chapter 11	Ed Durfee
Chapter 12	Alessandro Farinelli
Chapter 13	Jürgen Dix
Chapter 14	Jürgen Dix
Chapter 15	Michael Winikoff
Chapter 16	Wiebe van der Hoek
Chapter 17	Edith Elkind

Part I

Agent Architectures and Organizations

Chapter 1

Intelligent Agents

Michael Wooldridge

1 Introduction

Computers are not very good at knowing what to do: every action a computer performs must be explicitly anticipated, planned for, and coded by a programmer. If a computer program ever encounters a situation that its designer did not anticipate, then the result is ugly – a system crash at best, loss of life at worst. This mundane fact is at the heart of our relationship with computers. It is so self-evident to the computer literate that it is rarely mentioned. And yet it comes as a complete surprise to those programming computers for the first time.

For the most part, we are happy to accept computers as obedient, literal, unimaginative servants. For many applications, it is entirely acceptable. However, for an increasingly large number of applications, we require systems that can *decide for themselves* what they need to do in order to achieve the objectives that we delegate to them. Such computer systems are known as *agents*. Agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can *fail*, are known as *intelligent agents*, or sometimes *autonomous agents*. Here are some examples of recent application areas for intelligent agents:

- When a space probe makes its long flight from earth to the outer planets, a ground crew is usually required to continually track its progress, and decide how to deal with unexpected eventualities. This is costly, and if decisions

are required *quickly*, it is simply not practicable. For these reasons, organizations such as NASA and the European Space Agency are interested in the possibility of making probes more autonomous – giving them richer onboard decision-making capabilities and responsibilities.

- Searching the Internet for the answer to a specific query can be a long and tedious process. So, why not allow a computer program – an agent – do searches for us? The agent would typically be given a query that would require synthesizing pieces of information from various different Internet information sources. Failure would occur when a particular resource was unavailable (perhaps due to network failure), or where results could not be obtained.

This chapter is about intelligent agents. Specifically, it aims to give you an introduction to the main issues associated with the design and implementation of intelligent agents. After reading it, you will understand:

- what intelligent agents are (and are not), and how agents relate to other software paradigms – in particular, expert systems and object-oriented programming; and
- some of the main approaches that have been advocated for designing and implementing intelligent agents, the issues surrounding these approaches, their relative merits, and the challenges that face the agent implementor.

The chapter is structured as follows. First, Section 2 describes what is meant by the term *agent*. Section 3 then discusses *architectures* for agents. The various major design approaches that one can follow in implementing an agent system are outlined in this section. In particular, *logic-based* architectures, *reactive* architectures, *belief-desire-intention* architectures, and finally, *layered* architectures for intelligent agents are described in detail.

2 What Are Agents?

An obvious way to open this chapter would be by presenting a definition of the term *agent*. The definition presented here is adapted from [60]:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives.

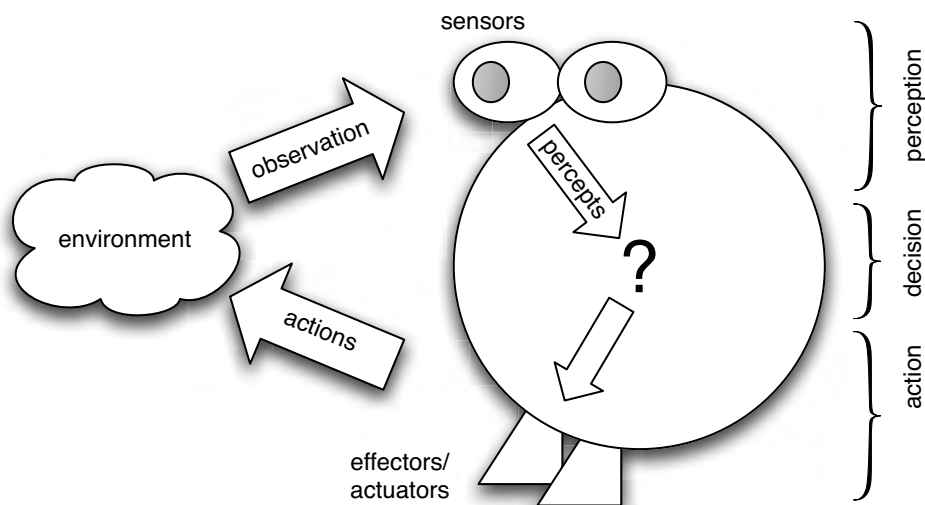


Figure 1.1: An agent in its environment. The agent takes sensory input in the form of *percepts* from the environment, and produces as output actions that affect it. The interaction is usually an ongoing, non-terminating one.

There are several points to note about this definition. First, the definition refers to “agents” and not “intelligent agents.” The distinction is deliberate: it is discussed in more detail below. Second, the definition does not say anything about what *type* of environment an agent occupies. Again, this is deliberate: agents can occupy many different types of environment, as we shall see below. Third, we have not defined *autonomy*. Like agency itself, autonomy is a somewhat tricky concept to tie down precisely. In this chapter, it is used to mean that agents are able to act without the intervention of humans or other systems: they have control both over their own internal state and over their behavior. In Section 2.3, we will contrast agents with the objects of object-oriented programming, and we will elaborate this point there. In particular, we will see how agents embody a much stronger sense of autonomy than objects do.

Figure 1.1 gives an abstract, top-level view of an agent. In this diagram, we can see the action output generated by the agent in order to affect its environment. In most domains of reasonable complexity, an agent will not have *complete* control over its environment. It will have at best *partial* control, in that it can *influence* it. From the point of view of the agent, this means that the same action performed twice in apparently identical circumstances might appear to have entirely different effects, and in particular, it may *fail* to have the desired effect. Thus agents in all but the most trivial of environments must be prepared for the possibility of

failure. We can sum this situation up formally by saying that environments are *non-deterministic*.

Normally, an agent will have a repertoire of actions available to it. This set of possible actions represents the agent's *effectoric capability*: its ability to modify its environment. Note that not all actions can be performed in all situations. For example, an action "lift table" is only applicable in situations where the weight of the table is sufficiently small that the agent *can* lift it. Similarly, the action "purchase a Ferrari" will fail if insufficient funds are available to do so. Actions therefore have *preconditions* associated with them, which define the possible situations in which they can be applied.

The key problem facing an agent is that of deciding *which* of its actions it should perform in order to best satisfy its delegated objectives. *Agent architectures*, of which we shall see several examples later in this chapter, are software architectures for decision-making systems that are embedded in an environment.

The complexity of the action selection process can be affected by a number of different environmental properties. Russell and Norvig suggest the following classification of environment properties [55, p. 46]:

- *Accessible vs. inaccessible*

An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible. The more accessible an environment is, the simpler it is to build agents to operate in it.

- *Deterministic vs. non-deterministic*

As we have already mentioned, a deterministic environment is one in which any action has a single guaranteed effect – there is no uncertainty about the state that will result from performing an action. The physical world can to all intents and purposes be regarded as non-deterministic. Non-deterministic environments present greater problems for the agent designer. As Russell and Norvig observe [55, p. 46], if an environment is sufficiently complex, then the fact that it is *actually* deterministic is not much help: to all intents and purposes, it may as well be non-deterministic.

- *Episodic vs. non-episodic*

In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios. An example of an episodic environment would be a mail sorting system [56]. Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to

perform based only on the current episode – it need not reason about the interactions between this and future episodes.

- *Static vs. dynamic*

A static environment is one that can be assumed to remain unchanged except for the performance of actions by the agent. A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control. The physical world is a highly dynamic environment.

- *Discrete vs. continuous*

An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.

The most complex general class of environments are those that are inaccessible, non-deterministic, non-episodic, dynamic, and continuous.

2.1 Examples of Agents

At this point, it is worth pausing to consider some examples of agents (though not, as yet, intelligent agents):

- Any *control* system can be viewed as an agent. A simple (and overused) example of such a system is a thermostat. Thermostats have a sensor for detecting room temperature. This sensor is directly embedded within the environment (i.e., the room), and it produces as output one of two signals: one that indicates that the temperature is too low, and another that indicates that the temperature is OK. The actions available to the thermostat are “heating on” or “heating off.” The action “heating on” will generally have the effect of raising the room temperature, but this cannot be a *guaranteed* effect – if the door to the room is open, for example, switching on the heater may have no effect. The (extremely simple) decision-making component of the thermostat implements the following rules:

too cold	→	heating on
temperature OK	→	heating off

More complex environment control systems, of course, have considerably richer decision structures. Examples include autonomous space probes, fly-by-wire aircraft, nuclear reactor control systems, and so on.

- Most software daemons (such as background processes in the UNIX operating system), which monitor a software environment and perform actions to modify it, can be viewed as agents. An example is the venerable X Windows program `xbiff`. This utility continually monitors a user's incoming e-mail, and indicates via an icon on the user interface whether or not they have unread messages. Whereas our thermostat agent in the previous example inhabited a *physical* environment, the `xbiff` program inhabits a *software* environment. It obtains information about this environment by executing software commands (e.g., system programs such as `ls` on a UNIX operating system, for example), and the actions it performs are also computer programs. Notice that the decision-making component in `xbiff` is just as simple as the one in our thermostat example.

To summarize, agents are simply computer systems that are capable of autonomous action in some environment in order to meet objectives that are delegated to them by us. An agent will typically sense its environment (by physical sensors in the case of agents situated in part of the real world, or by software sensors in the case of software agents), and will have available a repertoire of actions that can be executed to modify the environment, which may appear to respond non-deterministically to the execution of these actions.

2.2 Intelligent Agents

We do not think of thermostats or UNIX daemons as agents, and certainly not as *intelligent* agents. So, what properties would we demand of a system in order to call it an “intelligent agent”? For the purposes of this chapter, we consider an intelligent agent to be an agent that exhibits the following types of behavior in order to meet its delegated objectives [60]:

- *proactiveness*: intelligent agents are able to exhibit goal-directed behavior by *taking the initiative* in order to satisfy their delegated objectives;
- *reactivity*: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their delegated objectives;
- *social ability*: intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

These properties are more demanding than they might at first appear. To see why, let us consider them in turn. First, consider *proactiveness*: goal-directed behavior. It is not hard to build a system that exhibits goal-directed behavior – we do it every

time we write a procedure in PASCAL, a function in C, or a method in JAVA. When we write such a procedure, we describe it in terms of the *assumptions* on which it relies (formally, its *precondition*) and the *effect* it has if the assumptions are valid (its *postcondition*). The effects of the procedure are its *goal*: what the author of the software intends the procedure to achieve. If the precondition holds when the procedure is invoked, then we expect that the procedure will execute *correctly*: that it will terminate, and that upon termination, the postcondition will be true, i.e., the goal will be achieved. This is goal-directed behavior: the procedure is simply a plan or recipe for achieving the goal. This programming model is fine for many environments. For example, it works well when we consider *functional systems* – those that simply take some input x , and produce as output some function $f(x)$ of this input. Compilers are a classic example of functional systems.

But for non-functional systems, this simple model of goal-directed programming is not acceptable, as it makes some important limiting assumptions. In particular, it assumes that the environment *does not change* while the procedure is executing. If the environment does change, and in particular, if the assumptions (precondition) underlying the procedure become false while the procedure is executing, then the behavior of the procedure may not be defined – often, it will simply crash. Also, it is assumed that the goal, that is, the reason for executing the procedure, remains valid at least until the procedure terminates. If the goal does *not* remain valid, then there is simply no reason to continue executing the procedure.

In many environments, neither of these assumptions are valid. In particular, in domains that are *too complex* for an agent to observe completely, that are *multi-agent* (i.e., they are populated with more than one agent that can change the environment), or where there is *uncertainty* in the environment, these assumptions are not reasonable. In such environments, blindly executing a procedure without regard to whether the assumptions underpinning the procedure are valid is a poor strategy. In such dynamic environments, an agent must be *reactive*, in just the way that we described above. That is, it must be responsive to events that occur in its environment, where these events affect either the agent's goals or the assumptions which underpin the procedures that the agent is executing in order to achieve its goals.

As we have seen, building purely goal-directed systems is not hard. As we shall see later in this chapter, building *purely reactive* systems – ones that *continually* respond to their environment — is also not difficult. However, what turns out to be hard is building a system that achieves an effective *balance* between goal-directed and reactive behavior. We want agents that will attempt to achieve their goals systematically, typically by making use of complex procedures. But we don't want our agents to continue blindly executing these procedures in an

attempt to achieve a goal either when it is clear that the procedure will not work, or when the goal is for some reason no longer valid. In such circumstances, we want our agent to be able to react to the new situation, in time for the reaction to be of some use. However, we do not want our agent to be *continually* reacting, and hence never focusing on a goal long enough to actually achieve it.

On reflection, it should come as little surprise that achieving a good balance between goal-directed and reactive behavior is hard. After all, it is comparatively rare to find humans that do this very well. How many of us have had a manager who stayed blindly focused on some project long after the relevance of the project was passed, or it was clear that the project plan was doomed to failure? Similarly, how many have encountered managers who seem unable to stay focused at all, who flit from one project to another without ever managing to pursue a goal long enough to achieve *anything*? This problem — of effectively integrating goal-directed and reactive behavior — is one of the key problems facing the agent designer. As we shall see, a great many proposals have been made for how to build agents that can do this.

Finally, let us say something about *social ability*, the final component of flexible autonomous action as defined here. In one sense, social ability is trivial: every day, millions of computers across the world routinely exchange information with both humans and other computers. But the ability to exchange bit streams is not really social ability. Consider that in the human world, comparatively few of our meaningful goals can be achieved without the *cooperation* of other people, who cannot be assumed to *share* our goals — in other words, they are themselves autonomous, with their own agenda to pursue. To achieve our goals in such situations, we must *negotiate* and *cooperate* with others. We may be required to understand and reason about the goals of others, and to perform actions (such as paying them money) that we would not otherwise choose to perform in order to get them to cooperate with us, and achieve our goals. This type of social ability is much more complex, and much less well understood, in computational terms than simply the ability to exchange binary information. Social ability in general (and topics such as negotiation and cooperation in particular) are dealt with elsewhere in this book, and will not therefore be considered here. In this chapter, we will be concerned with the decision making of *individual* intelligent agents in environments that may be dynamic, unpredictable, and uncertain, but do not contain other agents.

2.3 Agents and Objects

Programmers who are familiar with object-oriented languages such as JAVA and C++ sometimes fail to see anything novel or new in the idea of agents. When one

stops to consider the relative properties of agents and objects, this is perhaps not surprising.

Objects are defined as computational entities that *encapsulate* some state, are able to perform actions, or *methods*, on this state, and communicate by message-passing. While there are obvious similarities between agents and objects, there are also significant differences. The first is in the degree to which agents and objects are autonomous. Recall that the defining characteristic of object-oriented programming is the principle of encapsulation – the idea that objects can have control over their own internal state. In programming languages like JAVA, we can declare instance variables (and methods) to be `private`, meaning they are only accessible from within the object. (We can of course also declare them `public`, meaning that they can be accessed from anywhere, and indeed we must do this for methods so that they can be used by other objects. But the use of `public` instance variables is usually considered poor programming style.) In this way, an object can be thought of as exhibiting autonomy over its state: it has control over it. But an object does not exhibit control over its *behavior*. That is, if a method `m` is made available for other objects to invoke, then they can do so whenever they wish – once an object has made a method `public`, then it subsequently has no control over whether or not that method is executed. Of course, an object *must* make methods available to other objects, or else we would be unable to build a system out of them. This is not normally an issue, because if we build a system, then we design the objects that go in it, and they can thus be assumed to share a “common goal.” But in many types of multiagent systems (in particular, those that contain agents built by different organizations or individuals), no such common goal can be assumed. It cannot be taken for granted that an agent *i* will execute an action (method) *a* just because another agent *j* wants it to – *a* may not be in the best interests of *i*. We thus do not think of agents as invoking methods upon one another, but rather as *requesting* actions to be performed. If *j* requests *i* to perform *a*, then *i* may perform the action or it may not. The locus of control with respect to the decision about whether to execute an action is thus different in agent and object systems. In the object-oriented case, the decision lies with the object that invokes the method. In the agent case, the decision lies with the agent that receives the request.

Note that there is nothing to stop us from implementing agents using object-oriented techniques. For example, we can build some kind of decision making about whether to execute a method into the method itself, and in this way achieve a stronger kind of autonomy for our objects. The point is that autonomy of this kind is not a component of the basic object-oriented model.

The second important distinction between object and agent systems is with respect to the notions of reactive, proactive, social, and autonomous behavior. The

standard object model has nothing whatsoever to say about how to build systems that integrate these types of behavior. Again, one could object that we can build object-oriented programs that *do* integrate these types of behavior. But this argument misses the point, which is that the standard object-oriented programming model has nothing to do with these types of behavior.

The third important distinction between the standard object model and our view of agent systems is that agents are each considered to have their own thread of control – in the standard object model, there is a single thread of control in the system. Of course, a lot of work has recently been devoted to *concurrency* in object-oriented programming. For example, the JAVA language provides built-in constructs for multithreaded programming. There are also many programming languages available that were specifically designed to allow concurrent object-based programming. But such languages do not capture the idea we have of agents as *autonomous* entities. Perhaps the closest that the object-oriented community comes is in the idea of *active objects*:

An active object is one that encompasses its own thread of control [...]. Active objects are generally autonomous, meaning that they can exhibit some behavior without being operated upon by another object. Passive objects, on the other hand, can only undergo a state change when explicitly acted upon. [5, p. 91]

Thus active objects are essentially agents that do not necessarily have the ability to exhibit flexible autonomous behavior. Objects in the standard object-oriented sense are simple *passive service providers*.

To summarize, the traditional view of an object and our view of an agent have at least three distinctions:

- agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent;
- agents are capable of reactive, proactive, social behavior, and the standard object model has nothing to say about such types of behavior; and
- a multiagent system is inherently multithreaded in that each agent is assumed to have its own thread of control, and is continually executing.

2.4 Agents and Expert Systems

Expert systems were the most important AI technology of the 1980s [25]. An expert system is one that is capable of solving problems or giving advice in some

knowledge-rich domain [26]. A classic example of an expert system is MYCIN, which was intended to assist physicians in the treatment of blood infections in humans. MYCIN worked by a process of interacting with a user in order to present the system with a number of (symbolically represented) facts, which the system then used to derive some conclusion. MYCIN acted very much as a *consultant*: it did not operate directly on humans, or indeed any other environment. Thus perhaps the most important distinction between agents and expert systems is that expert systems like MYCIN are inherently *disembodied*. By this, we mean that they do not interact directly with any environment: they get their information not via sensors, but through a user acting as a middle man. In the same way, they do not *act* on any environment, but rather give feedback or advice to a third party. In addition, we do not generally require expert systems to be capable of co-operating with other agents. Despite these differences, some expert systems (particularly those that perform real-time control tasks) look very much like agents. A good example is the ARCHON system [27].

2.5 Sources and Further Reading

A view of artificial intelligence as the process of agent design is presented in [55], and, in particular, Chapter 2 of [55] presents much useful material. The definition of agents presented here is based on [60], which also contains an extensive review of agent architectures and programming languages. In addition, [60] contains a detailed survey of *agent theories* – formalisms for reasoning about intelligent, rational agents – which is outside the scope of this chapter. This question of “what is an agent?” is one that continues to generate some debate; a collection of answers may be found in [42]. The relationship between agents and objects has not been widely discussed in the literature, but see [21]. Other readable introductions to the idea of intelligent agents include [28] and [13].

3 Architectures for Intelligent Agents

In this section, we will introduce the main approaches to building agents. Specifically, we consider four classes of agents:

- *logic-based agents* – in which the decision about what action to perform is made via logical deduction;
- *reactive agents* – in which decision making is implemented in some form of direct mapping from situation to action;

- *belief-desire-intention agents* – in which decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent; and finally,
- *layered architectures* – in which decision making is realized via various software layers, each of which is more or less explicitly reasoning about the environment at different levels of abstraction.

In what follows, we will use a little light notation to help explain the main ideas. We use $A = \{a, a', \dots\}$ to denote the set of possible actions that the agent can perform, and $S = \{s, s', \dots\}$ to denote the set of states that the environment can be in.

3.1 Logic-Based Architectures

The “traditional” approach to building artificially intelligent systems (known as *symbolic AI*) suggests that intelligent behavior can be generated in a system by giving that system a *symbolic* representation of its environment and its desired behavior, and syntactically manipulating this representation. In this section, we focus on the apotheosis of this tradition, in which these symbolic representations are *logical formulae*, and the syntactic manipulation corresponds to *logical deduction*, or *theorem proving*.

The idea of agents as theorem provers is seductive. Suppose we have some theory of agency – some theory that explains how an intelligent agent should behave. This theory might explain, for example, how an agent generates goals so as to satisfy its delegated objectives, how it interleaves goal-directed and reactive behavior in order to achieve these goals, and so on. Then this theory ρ can be considered as a *specification* for how an agent should behave. The traditional approach to implementing a system that will satisfy this specification would involve *refining* the specification through a series of progressively more concrete stages until finally an implementation was reached. In the view of agents as theorem provers, however, no such refinement takes place. Instead, ρ is viewed as an *executable specification*: it is *directly executed* in order to produce the agent’s behavior.

To see how such an idea might work, we shall develop a simple model of logic-based agents, which (following Genesereth and Nilsson [22]) we shall call *deliberate* agents. Such agents are assumed to maintain an internal database of formulae of classical first-order predicate logic, which represents in a symbolic form the information they have about their environment.

For example, an agent’s belief database might contain formulae such as the following:

$Open(valve221)$
 $Temperature(reactor4726, 321)$
 $Pressure(tank776, 28)$

It is not difficult to see how formulae such as these can be used to represent environment properties. The database is the *information* that the agent has about its environment. An agent's database plays a somewhat analogous role to that of *belief* in humans. Thus a person might have a belief that valve 221 is open – the agent might have the predicate $Open(valve221)$ in its database. Of course, just like humans, agents can be wrong. Thus I might believe that valve 221 is open when it is in fact closed; the fact that an agent has $Open(valve221)$ in its database does not mean that valve 221 (or indeed any valve) is open. The agent's sensors may be faulty, its reasoning may be faulty, the information may be out of date, or the interpretation of the formula $Open(valve221)$ intended by the agent's designer may be something entirely different.

Let L be the set of sentences of classical first-order logic. The internal state of a deliberate agent – the agent's "beliefs" – is then a subset of L , i.e., a set of formulae of first-order logic. We write Δ, Δ_1, \dots to denote such *belief databases*. An agent's decision-making process is modeled through a set of *deduction rules*, ρ . These are simply rules of inference for the logic. We write $\Delta \vdash_{\rho} \phi$ if the first-order formula ϕ can be proved from the database Δ using only the deduction rules ρ .

The pseudo-code definition of the action selection process for a deliberate agent is then given in Figure 1.2. This function $action(\dots)$ takes as input the beliefs of the agent (Δ) and deduction rules (ρ) and returns as output either an action (in which case this is the action selected for execution) or else *null* (indicating that no action can be found).

The idea is that the agent programmer will encode the deduction rules ρ and database Δ in such a way that if a formula $Do(a)$ can be derived, where a is a term that denotes an action, then a is the best action to perform. Thus, in the first part of the function (lines (3)–(7)), the agent takes each of its possible actions a in turn, and attempts to prove the formula $Do(a)$ from its database (passed as a parameter to the function) using its deduction rules ρ . If the agent succeeds in proving $Do(a)$, then a is returned as the action to be performed.

What happens if the agent fails to prove $Do(a)$, for all actions $a \in A$? In this case, it attempts to find an action that is *consistent* with the rules and database, i.e., one that is not explicitly forbidden. In lines (8)–(12), therefore, the agent attempts to find an action $a \in A$ such that $\neg Do(a)$ cannot be derived from its database using its deduction rules. If it can find such an action, then this is returned as the action to be performed. If, however, the agent fails to find an action that is at least

```

1.  function action( $\Delta, \rho$ ) returns an action
2.  begin
3.      for each  $a \in A$  do
4.          if  $\Delta \vdash_{\rho} Do(a)$  then
5.              return  $a$ 
6.          end-if
7.      end-for
8.      for each  $a \in A$  do
9.          if  $\Delta \not\vdash_{\rho} \neg Do(a)$  then
10.              return  $a$ 
11.          end-if
12.      end-for
13.      return null
14. end function action

```

Figure 1.2: Action selection in deliberate agents.

consistent, then it returns a special action *null* (or *noop*), indicating that no action has been selected.

In this way, the agent's behavior is determined by the agent's deduction rules (its "program") and its current database (representing the information the agent has about its environment).

To illustrate these ideas, let us consider a small example (based on the vacuum cleaning world example of [55, p. 51]). The idea is that we have a small robotic agent that will clean up a house. The robot is equipped with a sensor that will tell it whether it is over any dirt, and a vacuum cleaner that can be used to suck up dirt. In addition, the robot always has a definite orientation (one of *north*, *south*, *east*, or *west*). In addition to being able to suck up dirt, the agent can move forward one "step" or turn right 90° . The agent moves around a room, which is divided grid-like into a number of equally sized squares (conveniently corresponding to the unit of movement of the agent). We will assume that our agent does nothing but clean – it never leaves the room, and, further, we will assume in the interests of simplicity that the room is a 3×3 grid, and the agent always starts in grid square (0,0) facing north.

To summarize, our agent can receive a percept *dirt* (signifying that there is dirt beneath it), or *null* (indicating no special information). It can perform any one of three possible actions: *forward*, *suck*, or *turn*. The goal is to traverse the room, continually searching for and removing dirt. See Figure 1.3 for an illustration of the vacuum world.

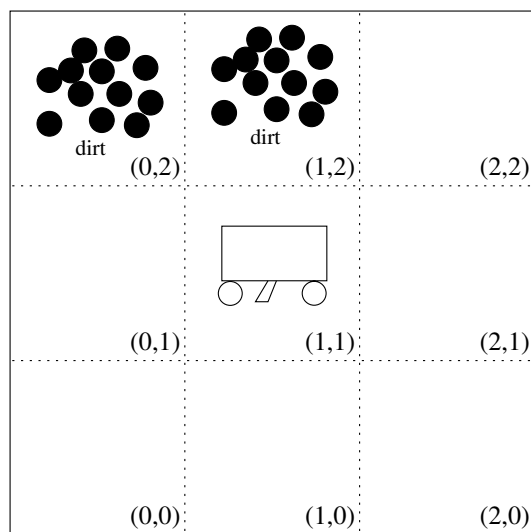


Figure 1.3: Vacuum world.

First, note that we make use of three simple *domain predicates* in this exercise:

$In(x,y)$	agent is at (x,y)
$Dirt(x,y)$	there is dirt at (x,y)
$Facing(d)$	the agent is facing direction d

Now we can move on to the rules ρ that govern our agent's behavior. The rules we use have the form

$$\phi(\dots) \longrightarrow \psi(\dots)$$

where ϕ and ψ are predicates over some arbitrary list of constants and variables. The idea being that if ϕ matches against the agent's database, then ψ can be concluded, with any variables in ψ instantiated.

The first rule deals with the basic cleaning action of the agent: this rule will take priority over all other possible behaviors of the agent (such as navigation).

$$In(x,y) \wedge Dirt(x,y) \longrightarrow Do(suck) \quad (1.1)$$

Hence if the agent is at location (x,y) and it perceives dirt, then the prescribed action will be to suck up dirt. Otherwise, the basic action of the agent will be to traverse the world. Taking advantage of the simplicity of our environment, we will hardwire the basic navigation algorithm, so that the robot will always move from $(0,0)$ to $(0,1)$ to $(0,2)$ and then to $(1,2)$, to $(1,1)$, and so on. Once the agent

reaches (2,2), it must head back to (0,0). The rules dealing with the traversal up to (0,2) are very simple.

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward) \quad (1.2)$$

$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward) \quad (1.3)$$

$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn) \quad (1.4)$$

$$In(0,2) \wedge Facing(east) \longrightarrow Do(forward) \quad (1.5)$$

Notice that in each rule, we must explicitly check whether the antecedent of rule (1.1) fires. This is to ensure that we only ever prescribe one action via the $Do(\dots)$ predicate. Similar rules can easily be generated that will get the agent to (2,2), and once at (2,2) back to (0,0).

At this point, let us step back and examine the pragmatics of this logic-based approach to building agents. Probably the most important point to make is that a literal, naive attempt to build agents in this way would be more or less entirely impractical. To see why, suppose we have designed our agent's rule set ρ such that for any database Δ , if we can prove $Do(a)$, then a is an *optimal* action – that is, a is the best action that could be performed when the environment is as described in Δ . Then imagine we start running our agent. At time t_1 , the agent has generated some database Δ_1 , and begins to apply its rules ρ in order to find which action to perform. Some time later, at time t_2 , it manages to establish $\Delta_1 \vdash_\rho Do(a)$ for some $a \in A$, and so a is the optimal action that the agent could perform at time t_1 . But if the environment has *changed* between t_1 and t_2 , then there is no guarantee that a will *still* be optimal. It could be far from optimal, particularly if much time has elapsed between t_1 and t_2 . If $t_2 - t_1$ is infinitesimal – that is, if decision making is effectively instantaneous – then we could safely disregard this problem. But in fact, we know that reasoning of the kind our logic-based agents use will be anything *but* instantaneous. (If our agent uses classical first-order predicate logic to represent the environment, and its rules are sound and complete, then there is no guarantee that the decision-making procedure will even *terminate*.) An agent is said to enjoy the property of *calculative rationality* if and only if its decision-making apparatus will suggest an action that was optimal *when the decision-making process began*. Calculative rationality is clearly not acceptable in environments that change faster than the agent can make decisions – we shall return to this point later.

One might argue that this problem is an artifact of the pure logic-based approach adopted here. There is an element of truth in this. By moving away from strictly logical representation languages and complete sets of deduction rules, one can build agents that enjoy respectable performance. But one also loses what is arguably the greatest advantage that the logical approach brings: a simple, elegant,

logical semantics.

There are several other problems associated with the logical approach to agency. First, there is the problem of “translating” raw data provided by the agent’s sensors into an internal symbolic form. For many environments, it is not obvious how the mapping from environment to symbolic form might be realized. For example, the problem of transforming an image to a set of declarative statements representing that image has been the object of study in AI for decades, and is still essentially open. Another problem is that actually *representing* properties of dynamic, real-world environments is extremely hard. As an example, representing and reasoning about *temporal information* – how a situation changes over time – turns out to be extraordinarily difficult. Finally, as the simple vacuum world example illustrates, representing even rather simple *procedural* knowledge (i.e., knowledge about “what to do”) in traditional logic can be rather unintuitive and cumbersome.

To summarize, in logic-based approaches to building agents, decision-making is viewed as deduction. An agent’s “program” – that is, its decision-making strategy – is encoded as a logical theory, and the process of selecting an action reduces to a problem of proof. Logic-based approaches are elegant, and have a clean (logical) semantics – wherein lies much of their long-lived appeal. But logic-based approaches have many disadvantages. In particular, the inherent computational complexity of theorem proving makes it questionable whether agents as theorem provers can operate effectively in time-constrained environments. Decision making in such agents is predicated on the assumption of calculative rationality – the assumption that the world will not change in any significant way while the agent is deciding what to do, and that an action that is rational when decision making begins will be rational when it concludes. The problems associated with representing and reasoning about complex, dynamic, possibly physical environments are also essentially unsolved.

3.1.1 Sources and Further Reading

My presentation of logic-based agents is based largely on the discussion of *deliberate agents* presented in [22, Chapter 13], which represents the logic-centric view of AI and agents very well. The discussion is also partly based on [33]. A number of more or less “pure” logical approaches to agent programming have been developed. Well-known examples include the GOLOG system of Reiter, Lespérance, and colleagues [34, 52] (which is based on the *situation calculus* [39]) and the METATEM and Concurrent METATEM programming languages developed by Fisher and colleagues [3, 20] (in which agents are programmed by giving them *temporal logic* specifications of the behavior they should exhibit). Note that these architectures (and the discussion above) assume that if one adopts a logical ap-

proach to agent-building, then this means agents are essentially theorem provers, employing explicit symbolic reasoning (theorem proving) in order to make decisions. But just because we find logic a useful tool for conceptualizing or specifying agents, this does not mean that we must view decision making as logical manipulation. An alternative is to *compile* the logical specification of an agent into a form more amenable to efficient decision making. The difference is rather like the distinction between interpreted and compiled programming languages. The best-known example of this work is the *situated automata* paradigm of Rosen-schein and Kaelbling [54]. A review of the role of logic in intelligent agents may be found in [59]. Finally, for a detailed discussion of calculative rationality and the way that it has affected thinking in AI, see [56].

3.2 Reactive Architectures

Problems with symbolic/logical approaches to building agents led some researchers to question, and ultimately reject, the assumptions upon which such approaches are based. These researchers have argued that minor changes to the symbolic approach, such as weakening the logical representation language, will not be sufficient to build agents that can operate in time-constrained environments: nothing less than a whole new approach is required. In the mid to late 1980s, these researchers began to investigate alternatives to the symbolic AI paradigm. It is difficult to neatly characterize these different approaches, since their advocates are united mainly by a rejection of symbolic AI, rather than by a common manifesto. However, certain themes do recur:

- the rejection of symbolic representations, and of decision making based on syntactic manipulation of such representations;
- the idea that intelligent, rational behavior is seen as innately linked to the *environment* an agent occupies – intelligent behavior is not disembodied, but is a product of the *interaction* the agent maintains with its environment;
- the idea that intelligent behavior *emerges* from the interaction of various simpler behaviors.

Alternative approaches to agency are sometimes referred to as *behavioral* (since a common theme is that of developing and combining individual behaviors), *situated* (since a common theme is that of agents actually situated in some environment, rather than being disembodied from it), and finally – the term used in this chapter – *reactive* (because such systems are often perceived as simply reacting to an environment, without reasoning about it).

3.2.1 The Subsumption Architecture

This section presents a survey of the *subsumption architecture*, which is arguably the best-known reactive agent architecture. It was developed by Rodney Brooks – one of the most vocal and influential critics of the symbolic approach to agency to have emerged in recent years.

There are two defining characteristics of the subsumption architecture. The first is that an agent’s decision making is realized through a set of *task accomplishing behaviors*; each behavior may be thought of as an individual action selection process, which continually takes perceptual input and maps it to an action to perform. Each of these behavior modules is intended to achieve some particular task. In Brooks’s implementation, the behavior modules are finite-state machines. An important point to note is that these task accomplishing modules are assumed to include *no* complex symbolic representations, and are assumed to do *no* symbolic reasoning at all. In many implementations, these behaviors are implemented as rules of the form

situation \longrightarrow action

which simply map perceptual input directly to actions.

The second defining characteristic of the subsumption architecture is that many behaviors can “fire” simultaneously. There must obviously be a mechanism to choose between the different actions selected by these multiple actions. Brooks proposed arranging the modules into a *subsumption hierarchy*, with the behaviors arranged into *layers*. Lower layers in the hierarchy are able to *inhibit* higher layers: the lower a layer is, the higher is its priority. The idea is that higher layers represent more abstract behaviors. For example, one might desire a behavior in a mobile robot for the behavior “avoid obstacles.” It makes sense to give obstacle avoidance a high priority – hence this behavior will typically be encoded in a *low-level* layer, which has a *high* priority.

To illustrate in more detail how the subsumption architecture works, we will show how subsumption architecture agents were built for the following scenario (from [58]):

The objective is to explore a distant planet, more concretely, to collect samples of a particular type of precious rock. The location of the rock samples is not known in advance, but they are typically clustered in certain spots. A number of autonomous vehicles are available that can drive around the planet collecting samples and later reenter the mothership spacecraft to go back to earth. There is no detailed map of the planet available, although it is known that the terrain is full

of obstacles – hills, valleys, etc. – which prevent the vehicles from exchanging any communication.

The problem we are faced with is that of building an agent control architecture for each vehicle, so that they will cooperate to collect rock samples from the planet surface as efficiently as possible. Luc Steels argues that logic-based agents, of the type we described above, are “entirely unrealistic” for this problem [58]. Instead, he proposes a solution using the subsumption architecture.

The solution makes use of two mechanisms introduced by Steels. The first is a *gradient field*. In order that agents can know in which direction the mothership lies, the mothership generates a radio signal. Now this signal will obviously weaken as distance to the source increases – to find the direction of the mothership, an agent need therefore only travel “up the gradient” of signal strength. The signal need not carry any information – it need only exist.

The second mechanism enables agents to communicate with one another. The characteristics of the terrain prevent direct communication (such as message-passing), so Steels adopted an *indirect* communication method. The idea is that agents will carry “radioactive crumbs,” which can be dropped, picked up, and detected by passing robots. Thus if an agent drops some of these crumbs in a particular location, then later, another agent happening upon this location will be able to detect them. This simple mechanism enables a quite sophisticated form of cooperation.

The behavior of an individual agent is then built up from a number of behaviors, as we indicated above. First, we will see how agents can be programmed to *individually* collect samples. We will then see how agents can be programmed to generate a *cooperative* solution.

For individual (non-cooperative) agents, the lowest-level behavior (and hence the behavior with the highest “priority”) is obstacle avoidance. This behavior can be represented in the rule:

if detect an obstacle then change direction. (1.6)

The second behavior ensures that any samples carried by agents are dropped back at the mothership.

if carrying samples and at the base then drop samples. (1.7)

if carrying samples and not at the base then travel up gradient. (1.8)

Behavior (1.8) ensures that agents carrying samples will return to the mothership (by heading toward the origin of the gradient field). The next behavior ensures that agents will collect samples they find.

if detect a sample *then* pick sample up. (1.9)

The final behavior ensures that an agent with “nothing better to do” will explore randomly.

if true *then* move randomly. (1.10)

The precondition of this rule is thus assumed to always fire. These behaviors are arranged into the following hierarchy:

$$(1.6) \prec (1.7) \prec (1.8) \prec (1.9) \prec (1.10)$$

where the left-most behavior is the lowest in the hierarchy (i.e., highest priority), and the right-most behavior is the highest in the hierarchy (i.e., lowest priority).

The subsumption hierarchy for this example ensures that, for example, an agent will *always* turn if any obstacles are detected; if the agent is at the mothership and is carrying samples, then it will *always* drop them if it is not in any immediate danger of crashing, and so on. The “top level” behavior – a random walk – will only ever be carried out if the agent has nothing more urgent to do. It is not difficult to see how this simple set of behaviors will solve the problem: agents will search for samples (ultimately by searching randomly), and when they find them, will return them to the mothership.

If the samples are distributed across the terrain entirely at random, then equipping a large number of robots with these very simple behaviors will work extremely well. But we know from the problem specification, above, that this is not the case: the samples tend to be located in clusters. In this case, it makes sense to have agents *cooperate* with one another in order to find the samples. Thus when one agent finds a large sample, it would be helpful for it to communicate this to the other agents, so they can help it collect the rocks. Unfortunately, we also know from the problem specification that *direct* communication is impossible. Steels developed a simple solution to this problem, partly inspired by the foraging behavior of ants. The idea revolves around an agent creating a “trail” of radioactive crumbs whenever it finds a rock sample. The trail will be created when the agent returns the rock samples to the mothership. If at some later point, another agent comes across this trail, then it need only follow it down the gradient field to locate the source of the rock samples. Some small refinements improve the efficiency of this ingenious scheme still further. First, as an agent follows a trail to the rock sample source, it picks up some of the crumbs it finds, hence making the trail fainter. Secondly, the trail is *only* laid by agents returning to the mothership. Hence if an agent follows the trail out to the source of the nominal rock sample only to find that it contains no samples, it will reduce the trail on the way out, and

will not return with samples to reinforce it. After a few agents have followed the trail to find no sample at the end of it, the trail will in fact have been removed.

The modified behaviors for this example are as follows. Obstacle avoidance (1.6) remains unchanged. However, the two rules determining what to do if carrying a sample are modified as follows.

if carrying samples and at the base then drop samples. (1.11)

*if carrying samples and not at the base
then drop 2 crumbs and travel up gradient.* (1.12)

The behavior (1.12) requires an agent to drop crumbs when returning to base with a sample, thus either reinforcing or creating a trail. The “pick up sample” behavior, (1.9), remains unchanged. However, an additional behavior is required for dealing with crumbs.

if sense crumbs then pick up 1 crumb and travel down gradient. (1.13)

Finally, the random movement behavior (1.10) remains unchanged. These behaviors are then arranged into the following subsumption hierarchy:

$$(1.6) \prec (1.11) \prec (1.12) \prec (1.9) \prec (1.13) \prec (1.10)$$

Steels shows how this simple adjustment achieves near-optimal performance in many situations. Moreover, the solution is *cheap* (the computing power required by each agent is minimal) and *robust* (the loss of a single agent will not affect the overall system significantly).

In summary, there are obvious advantages to reactive approaches such as Brooks’s subsumption architecture: simplicity, economy, computational tractability, robustness against failure, and elegance all make such architectures appealing. But there are some fundamental, unsolved problems, not just with the subsumption architecture but also with other purely reactive architectures:

- If agents do not employ models of their environment, then they must have sufficient information available in their *local* environment for them to determine an acceptable action.
- Since purely reactive agents make decisions based on *local* information, (i.e., information about the agents’ *current* state), it is difficult to see how such decision making could take into account *non-local* information – it must inherently take a “short-term” view.

- A major selling point of purely reactive systems is that overall behavior *emerges* from the interaction of the component behaviors when the agent is placed in its environment. But the very term “emerges” suggests that the relationship between individual behaviors, environment, and overall behavior is not understandable. This necessarily makes it very hard to *engineer* agents to fulfill specific tasks. Ultimately, there is no principled *methodology* for building such agents: one must use a laborious process of experimentation, trial, and error to engineer an agent.
- While effective agents can be generated with small numbers of behaviors (typically less than ten layers), it is *much* harder to build agents that contain many layers. The dynamics of the interactions between the different behaviors become too complex to understand.

3.2.2 Markov Decision Processes

One very active area of development in the reactive agents area over the past decade is the use of *Markov models* [44]. Markov models were originally developed by the Russian mathematician Andrei Markov as models of stochastic processes – that is, dynamic processes whose behaviors are probabilistic. Markov models in their various forms represent one of the fundamental models used in operations research for modeling stochastic processes, and they are now very widely used in AI for environments in which a sequence of decisions must be made over time. In this section, we will very briefly survey the principles of Markov models, and, in particular, we will present the basic concepts and algorithms of *Markov Decision Processes* (MDPs), upon which most work in the area builds.

The basic components of Markov models are as follows. We assume there is an agent in an environment that can be any of a set S of states, and that the agent can modify its environment by performing actions from a set A . A fundamental assumption in Markov models is that the behavior of a system is stochastic, in the sense that the result of performing a particular action in a particular state is not uniquely determined. We write $p(s' | s, a)$ to denote the probability that state s' will result if action $a \in A$ is performed in state s . Thus $p(\cdot \cdot \cdot)$ defines, for every state $s \in S$ and action $a \in A$, a probability distribution over states. Next, MDP models make use of the notion of *reward*. We let $r(a, s) \in \mathbb{R}$ denote the reward obtained from performing action $a \in A$ in state $s \in S$. If the reward is negative, i.e., $r(a, s) < 0$, then the reward can be interpreted as the *cost* of performing the action a in state s . Notice that the behavior of a system (defined by the probability function $p(s | s, a)$) and the rewards obtained by performing actions within it (defined by the reward values $r(a, s)$), depend *solely* on the state in which an action is performed – not on what states or actions occur before or afterward. This property is so central to

Markov models that it is called the *Markov assumption*.

Now, *solving* a Markov decision problem defined by state set S , action set A , probability function p , and reward function r , means finding a *policy* for choosing actions in the MDP. A policy (sometimes called a *decision rule*) is essentially a conditional plan: it defines an action to perform for every possible state. Formally, a policy can be understood as a function $d : S \rightarrow A$. So, given an MDP $\langle S, A, p, r \rangle$, which policy should we choose? Intuitively, we want the one that *maximizes the expected reward that we would obtain should we choose to follow the policy*.

One obvious difficulty is that, as formulated here, MDPs are non-terminating: a policy dictates an action for every state, and so we just carry on choosing actions to perform in each state we find ourselves. So, it may well be that executing a policy will result in infinite expected reward; and how are we to compare two putatively optimal policies when both yield infinite expected rewards? A standard solution is to use a *discount factor*, λ , where $0 \leq \lambda < 1$. The idea is to use λ to discount rewards obtained in the future, so that rewards obtained soon are valued more highly. With this idea, we can now define an approach to finding an optimal policy.

The approach to finding an optimal policy that we describe here is called *value iteration*, and it is one of the two standard approaches to finding optimal policies in MDPs. Finding an optimal policy using value iteration proceeds in two steps. First, we compute the *value function*, $v^* : S \rightarrow \mathbb{R}$, which gives the *value* $v^*(s)$ of every state $s \in S$. The value v^* of state $s \in S$ is *the expected reward that would be obtained from executing the optimal policy in that state*. Now, given the value function v^* , we can then easily “extract” the optimal action from s , in a way that we will describe shortly.

So, first we show how to compute the value function v^* . The algorithm for this is given in Figure 1.4; the idea is to iteratively approximate v^* using two variables v (the “old” approximation to the optimal policy) and v^* (the “new” approximation to the optimal policy). We continue to iterate until the difference between the old and new approximation is sufficiently small.

We have not defined in this algorithm what “sufficiently close” means. A number of answers are possible. A standard approach is to define some convergence threshold ϵ , and to stop when the maximum difference between v and v^* is ϵ . When $\epsilon = 0$, we are requiring exact convergence. Now, it can be shown (we will not do it here) that this algorithm does indeed converge on an optimal value function.

So, after executing this algorithm, we will have in v^* the value of every state, i.e., the expected value that would be obtained by executing the optimal policy from that state. How does this give us the optimal policy itself? Trivially, simply observe that the optimal policy d^* will satisfy the following property:

```

1.  function value_iteration( $S, A, p, r, \lambda$ ) return optimal policy  $v^*$ 
2.      initialize  $v^*$  randomly
3.      repeat
4.           $v := v^*$ 
5.          for each  $s \in S$  do
6.               $v^*(s) := \max_{a \in A} \left( r(a, s) + \lambda \sum_{s' \in S} p(s' | s, a) v(s') \right)$ 
7.          end-for
8.      until  $v$  and  $v^*$  are sufficiently close
9.      return  $v^*$ 
10. end function value_iteration

```

Figure 1.4: The value iteration algorithm for Markov decision processes.

$$d^*(s) = \arg \max_{a \in A} \left(r(s, a) + \lambda \sum_{s' \in S} p(s' | s, a) v^*(s') \right)$$

Now, since at this stage we will have the values for $r(s, a)$, $p(s' | s, a)$, and $v^*(s')$, this equation can be directly applied to find the optimal policy $d^*(s)$.

3.2.3 Sources and Further Reading

Brooks's original paper on the subsumption architecture – the one that started all the fuss – was published as [9]. The description and discussion here is partly based on [14]. This original paper seems to be somewhat less radical than many of his later ones, which include [10, 11, 12]. The version of the subsumption architecture used in this chapter is actually a simplification of that presented by Brooks. The subsumption architecture is probably the best-known reactive architecture around — but there are many others. The collection of papers edited by Pattie Maes [36] contains papers that describe many of these, as does the collection by Agre and Rosenzchein [2]. Other approaches include:

- the *agent network architecture* developed by Pattie Maes [35, 37, 38];
- Nilsson's *teleo reactive programs* [43];
- Rosenzchein and Kaelbling's *situated automata* approach, which is particularly interesting in that it shows how agents can be *specified* in an abstract,

logical framework, and *compiled* into equivalent, but computationally very simple, machines [29, 31, 53, 54];

- Agre and Chapman’s PENG1 system [1];
- Schoppers’ *universal plans* – which are essentially decision trees that can be used to efficiently determine an appropriate action in any situation [57];
- Firby’s *reactive action packages* [18].

Kaelbling [28] gives a good discussion of the issues associated with developing resource-bounded rational agents, and proposes an agent architecture somewhat similar to that developed by Brooks.

Markov decision processes are a huge research topic in computer science and operations research, going much further than the simple framework and algorithms we have described here. The definitive reference to MDPs and their endless variations is [44]; a good introduction to Markov models in AI is [30].

3.3 Belief-Desire-Intention Architectures

In this section, we shall discuss *belief-desire-intention* (BDI) architectures. These architectures have their roots in the philosophical tradition of understanding *practical reasoning* — the process of deciding, moment by moment, which action to perform in the furtherance of our goals.

Practical reasoning involves two important processes: deciding *what* goals we want to achieve, and *how* we are going to achieve these goals. The former process is known as *deliberation*, the latter as *means-ends* reasoning. To gain an understanding of the BDI model, it is worth considering a simple example of practical reasoning. When you leave university with a first degree, you are faced with a decision to make – about what to do with your life. The decision process typically begins by trying to understand what the *options* available to you are. For example, if you gain a good first degree, then one option is that of becoming an academic. (If you fail to obtain a good degree, this option is not available to you.) Another option is entering industry. After generating this set of alternatives, you must *choose between them*, and *commit* to some. These chosen options become *intentions*, which then determine the agent’s actions. Intentions then feed back into the agent’s future practical reasoning. For example, if I decide I want to be an academic, then I should commit to this objective, and devote time and effort to bringing it about.

Intentions play a crucial role in the practical reasoning process. Perhaps the most obvious property of intentions is that they tend to lead to action. If I truly have an intention to become an academic, then you would expect me to *act* on

that intention — to try to achieve it. For example, you might expect me to apply to various PhD programs. You would expect to make a *reasonable attempt* to achieve the intention. Thus you would expect me to carry out some course of action that I believed would best satisfy the intention. Moreover, if a course of action fails to achieve the intention, then you would expect me to *try again* — you would not expect me to simply give up. For example, if my first application for a PhD program is rejected, then you might expect me to apply to alternative universities.

In addition, once I have adopted an intention, then the very fact of having this intention will constrain my future practical reasoning. For example, while I hold some particular intention, I will not entertain options that are inconsistent with that intention. Intending to become an academic, for example, would preclude the option of partying every night: the two are mutually exclusive.

Next, intentions *persist*. If I adopt an intention to become an academic, then I should *persist* with this intention and attempt to achieve it. For if I immediately drop my intentions without devoting resources to achieving them, then I will never achieve anything. However, I should not persist with my intention for too long — if it becomes clear to me that I will *never* become an academic, then it is only rational to drop my intention to do so. Similarly, if the reason for having an intention goes away, then it is rational of me to drop the intention. For example, if I adopted the intention to become an academic because I believed it would be an easy life, but then discover that I would be expected to actually *teach*, then the justification for the intention is no longer present, and I should drop the intention.

Finally, intentions are closely related to beliefs about the future. For example, if I intend to become an academic, then I should believe that I will indeed become an academic. For if I truly believe that I will never be an academic, it would be nonsensical of me to have an intention to become one. Thus if I intend to become an academic, I should at least believe that there is a good chance I will indeed become one.

From this discussion, we can see that intentions play a number of important roles in practical reasoning:

- *Intentions drive means-ends reasoning.*

If I have formed an intention to become an academic, then I will attempt to achieve the intention, which involves, among other things, deciding *how* to achieve it, for example, by applying for a PhD program. Moreover, if one particular course of action fails to achieve an intention, then I will typically attempt others. Thus if I fail to gain a PhD place at one university, I might try another university.

- *Intentions constrain future deliberation.*

If I intend to become an academic, then I will not entertain options that are inconsistent with this intention. For example, a rational agent would not consider being rich as an option while simultaneously intending to be an academic. (While the two are not actually mutually exclusive, the probability of simultaneously achieving both is infinitesimal.)

- *Intentions persist.*

I will not usually give up on my intentions without good reason – they will persist, typically until either I believe I have successfully achieved them, I believe I cannot achieve them, or else because the purpose for the intention is no longer present.

- *Intentions influence beliefs upon which future practical reasoning is based.*

If I adopt the intention to become an academic, then I can plan for the future on the assumption that I *will* be an academic. For if I intend to be an academic while simultaneously believing that I will never be one, then I am being irrational.

A key problem in the design of practical reasoning agents is that of achieving a good *balance* between these different concerns. Specifically, it seems clear that an agent should at times drop some intentions (because it comes to believe that either they will never be achieved, they are achieved, or else because the reason for having the intention is no longer present). It follows that, from time to time, it is worth an agent stopping to *reconsider* its intentions. But reconsideration has a cost – in terms of both time and computational resources. But this presents us with a dilemma:

- an agent that does not stop to reconsider sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them;
- an agent that *constantly* reconsiders its intentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them.

This dilemma is essentially the problem of balancing proactive (goal-directed) and reactive (event-driven) behavior, that we introduced in Section 2.2.

There is clearly a trade-off to be struck between the degree of commitment and reconsideration at work here. The nature of this trade-off was examined by David Kinny and Michael Georgeff in a number of experiments carried out with

a BDI agent framework called dMARS [32]. They investigate how *bold* agents (those that never stop to reconsider) and *cautious* agents (those that are constantly stopping to reconsider) perform in a variety of different environments. The most important parameter in these experiments was the *rate of world change*, γ . The key results of Kinny and Georgeff were as follows.

- If γ is low (i.e., the environment does not change quickly) then bold agents do well compared to cautious ones, because cautious ones waste time reconsidering their commitments while bold agents are busy working towards – and achieving – their goals.
- If γ is high (i.e., the environment changes frequently) then cautious agents tend to outperform bold agents, because they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities.

The lesson is that different types of environments require different types of decision strategies. In static, unchanging environments, purely proactive, goal-directed behavior is adequate. But in more dynamic environments, the ability to react to changes by modifying intentions becomes more important.

The process of practical reasoning in a BDI agent is summarized in Figure 1.5. As this figure illustrates, there are seven main components to a BDI agent:

- a set of current *beliefs*, representing information the agent has about its current environment;
- a *belief revision function* (*brf*), which takes a perceptual input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs;
- an *option generation function* (*options*), which determines the options available to the agent (its desires), on the basis of its current beliefs about its environment and its current *intentions*;
- a set of *current options*, representing possible courses of actions available to the agent;
- a *filter* function (*filter*), which represents the agent's *deliberation* process, and which determines the agent's intentions on the basis of its current beliefs, desires, and intentions;
- a set of current *intentions*, representing the agent's current focus – those states of affairs that it has committed to trying to bring about;

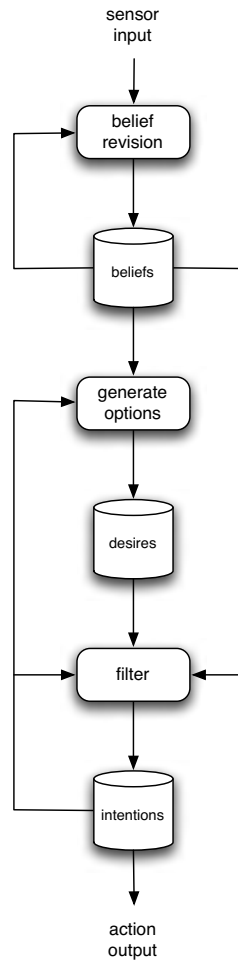


Figure 1.5: Schematic diagram of a generic belief-desire-intention architecture.

- an *action selection function* (*execute*), which determines an action to perform on the basis of current intentions.

It is straightforward to formally define these components. First, let *Bel* be the set of all possible beliefs, *Des* be the set of all possible desires, and *Int* be the set of all possible intentions. For the purposes of this chapter, the content of these sets is not important. (Often, beliefs, desires, and intentions are represented as logical formulae, perhaps of first-order logic.) Whatever the content of these sets, it is worth noting that they should have some notion of *consistency* defined upon them, so that one can answer the question of, for example, whether having an intention to achieve *x* is consistent with the belief that *y*. Representing beliefs,

desires, and intentions as logical formulae permits us to recast such questions as problems of determining whether logical formulae are consistent – a well-known and well-understood problem. The state of a BDI agent at any given moment is, unsurprisingly, a triple (B, D, I) , where $B \subseteq Bel$, $D \subseteq Des$, and $I \subseteq Int$.

If we denote the set of possible percepts that the agent can receive by P , then an agent's belief revision function is a mapping

$$brf : 2^{Bel} \times P \rightarrow 2^{Bel}$$

which on the basis of the current percept and current beliefs determines a new set of beliefs. Belief revision is out of the scope of this chapter (and indeed this book), and so we shall say no more about it here.

The option generation function, *options*, maps a set of beliefs and a set of intentions to a set of desires.

$$options : 2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$$

This function plays several roles. First, it must be responsible for the agent's means-ends reasoning – the process of deciding how to achieve intentions. Thus, once an agent has formed an intention to x , it must subsequently consider options to *achieve* x . These options will be more concrete – less abstract – than x . As some of these options then become intentions themselves, they will also feedback into option generation, resulting in yet more concrete options being generated. We can thus think of a BDI agent's option generation process as one of recursively elaborating a hierarchical plan structure, considering and committing to progressively more specific intentions, until finally it reaches the intentions that correspond to immediately executable actions.

While the main purpose of the *options* function is thus means-ends reasoning, it must in addition satisfy several other constraints. First, it must be *consistent*: any options generated must be consistent with both the agent's current beliefs and current intentions. Secondly, it must be *opportunistic*, in that it should recognize when environmental circumstances change advantageously, to offer the agent new ways of achieving intentions, or the possibility of achieving intentions that were otherwise unachievable.

A BDI agent's deliberation process (deciding *what* to do) is represented in the *filter* function,

$$filter : 2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$$

which updates the agent's intentions on the basis of its previously-held intentions and current beliefs and desires. This function must fulfill two roles. First, it must *drop* any intentions that are no longer achievable, or for which the expected cost of

achieving them exceeds the expected gain associated with successfully achieving them. Second, it should *retain* intentions that are not achieved, and that are still expected to have a positive overall benefit. Finally, it should *adopt* new intentions, either to achieve existing intentions, or to exploit new opportunities.

Notice that we do not expect this function to introduce intentions from nowhere. Thus *filter* should satisfy the following constraint:

$$\forall B \in 2^{Bel}, \forall D \in 2^{Des}, \forall I \in 2^{Int}, filter(B, D, I) \subseteq I \cup D.$$

In other words, current intentions are either previously held intentions or newly adopted options.

The *execute* function is assumed to simply return any executable intentions – one that corresponds to a directly executable action:

$$execute : 2^{Int} \rightarrow A$$

The action selection function *action* of a BDI agent is then a function that takes as input a percept (i.e., some raw sensor data, denoted by p), and returns an action; it is defined by the following pseudo-code.

1. function *action*(p) returns an action
2. begin
3. $B := bnf(B, p)$
4. $D := options(B, I)$
5. $I := filter(B, D, I)$
6. return *execute*(I)
7. end function *action*

Note that representing an agent's intentions as a *set* (i.e., as an unstructured collection) is generally too simplistic in practice. A simple alternative is to associate a *priority* with each intention, indicating its relative importance. Another natural idea is to represent intentions as a *stack*. An intention is pushed on to the stack when it is adopted, and popped when it is either achieved or else not achievable. More abstract intentions will tend to be at the bottom of the stack, with more concrete intentions toward the top.

To summarize, BDI architectures are practical reasoning architectures, in which the process of deciding what to do resembles the kind of practical reasoning that we appear to use in our everyday lives. The basic components of a BDI architecture are data structures representing the beliefs, desires, and intentions of the agent, and functions that represent its deliberation (deciding *what* intentions

to have – i.e., deciding what to do) and means-ends reasoning (deciding how to do it). Intentions play a central role in the BDI model: they provide stability for decision making, and act to focus the agent’s practical reasoning. A major issue in BDI architectures is the problem of striking a *balance* between being committed to and overcommitted to one’s intentions: the deliberation process must be finely tuned to its environment, ensuring that in more dynamic, highly unpredictable domains, it reconsiders its intentions relatively frequently – in more static environments, less frequent reconsideration is necessary.

The BDI model is attractive for several reasons. First, it is intuitive – we all recognize the processes of deciding what to do and then how to do it, and we all have an informal understanding of the notions of belief, desire, and intention. Second, it gives us a clear functional decomposition, which indicates what sorts of subsystems might be required to build an agent. But the main difficulty, as ever, is knowing how to efficiently implement these functions.

3.3.1 Sources and Further Reading

Belief-desire-intention architectures originated in the work of the Rational Agency project at Stanford Research Institute in the mid 1980s. The origins of the model lie in the theory of human practical reasoning developed by the philosopher Michael Bratman [7], which focuses particularly on the role of intentions in practical reasoning. The conceptual framework of the BDI model is described in [8], which also describes a specific BDI agent architecture called IRMA. The description of the BDI model given here (and in particular in Figure 1.5) is adapted from [8]. One of the interesting aspects of the BDI model is that it has been used in one of the most successful agent architectures to date. The Procedural Reasoning System (PRS), originally developed by Michael Georgeff and Amy Lansky [23], has been used to build some of the most exacting agent applications to date, including fault diagnosis for the reaction control system of the space shuttle, and an air traffic management system at Sydney Airport in Australia – overviews of these systems are described in [24]. In the PRS, an agent is equipped with a library of *plans*, which are used to perform means-ends reasoning. Deliberation is achieved by the use of *meta-level plans*, which are able to modify an agent’s intention structure at run-time, in order to change the focus of the agent’s practical reasoning. Beliefs in the PRS are represented as PROLOG-like facts – essentially, as atoms of first-order logic. A number of modern implementations of PRS are available. Perhaps the best-known of these is the AGENTSPEAK language [45], which has been implemented in a freely available distribution known as JASON [6].

The BDI model is also interesting because a great deal of effort has been devoted to formalizing it. In particular, Anand Rao and Michael Georgeff have

developed a range of BDI logics, which they use to axiomatize properties of BDI-based practical reasoning agents [46, 47, 48, 49, 50, 51].

3.4 Layered Architectures

Given the requirement that an agent be capable of reactive and proactive behavior, an obvious decomposition involves creating separate subsystems to deal with these different types of behaviors. This idea leads naturally to a class of architectures in which the various subsystems are arranged into a hierarchy of interacting *layers*. In this section, we will consider some general aspects of layered architectures, and then go on to consider two examples of such architectures: INTERRAP and TOURINGMACHINES.

Typically, there will be at least two layers, to deal with reactive and proactive behaviors, respectively. In principle, there is no reason why there should not be many more layers. However many layers there are, a useful typology for such architectures is by the information and control flows within them. Broadly speaking, we can identify two types of control flow within layered architectures (see Figure 1.6):

- *Horizontal layering*

In horizontally layered architectures (Figure 1.6(a)), the software layers are each directly connected to the sensory input and action output. In effect, each layer itself acts like an agent, producing suggestions as to what action to perform.

- *Vertical layering*

In vertically layered architectures (Figure 1.6(b) and 1.6(c)), sensory input and action output are each dealt with by at most one layer each.

The great advantage of horizontally layered architectures is their conceptual simplicity: if we need an agent to exhibit n different types of behavior, then we implement n different layers. However, because the layers are each in effect competing with one another to generate action suggestions, there is a danger that the *overall* behavior of the agent will not be coherent. In order to ensure that horizontally layered architectures *are* consistent, they generally include a *mediator* function, which makes decisions about which layer has “control” of the agent at any given time. The need for such central control is problematic: it means that the designer must potentially consider all possible interactions between layers. If there are n layers in the architecture, and each layer is capable of suggesting m possible actions, then this means there are m^n such interactions to be considered. This is clearly difficult from a design point of view in any but the most simple system.

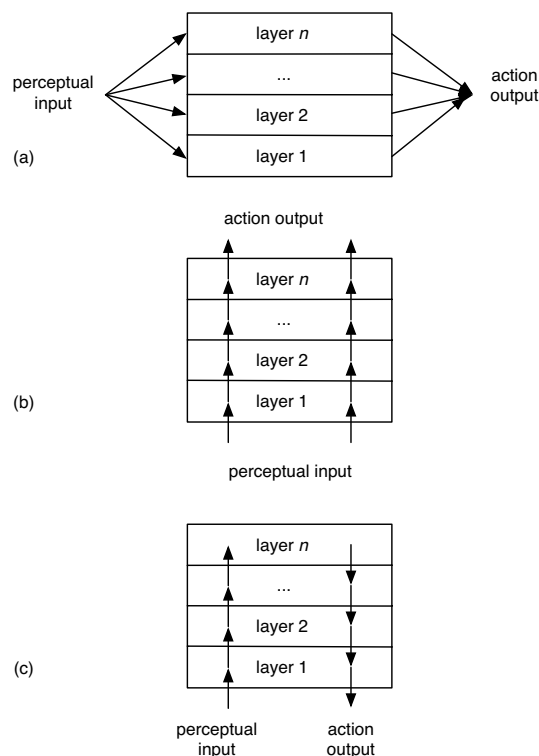


Figure 1.6: Information and control flows in three types of layered agent architectures (Source: [41, p. 263]).

The introduction of a central control system also introduces a *bottleneck* into the agent's decision making.

These problems are partly alleviated in a vertically layered architecture. We can subdivide vertically layered architectures into *one-pass* architectures (Figure 1.6(b)) and *two-pass* architectures (Figure 1.6(c)). In one-pass architectures, control flows sequentially through each layer, until the final layer generates action output. In two-pass architectures, information flows up the architecture (the first pass) and control then flows back down. There are some interesting similarities between the idea of two-pass vertically layered architectures and the way that organizations work, with information flowing up to the highest levels of the organization, and commands then flowing down. In both one-pass and two-pass vertically layered architectures, the complexity of interactions between layers is reduced: since there are $n - 1$ interfaces between n layers, then if each layer is capable of suggesting m actions, there are at most $m^2(n - 1)$ interactions to be considered between layers. This is clearly much simpler than the horizontally layered case. However, this simplicity comes at the cost of some flexibility: in

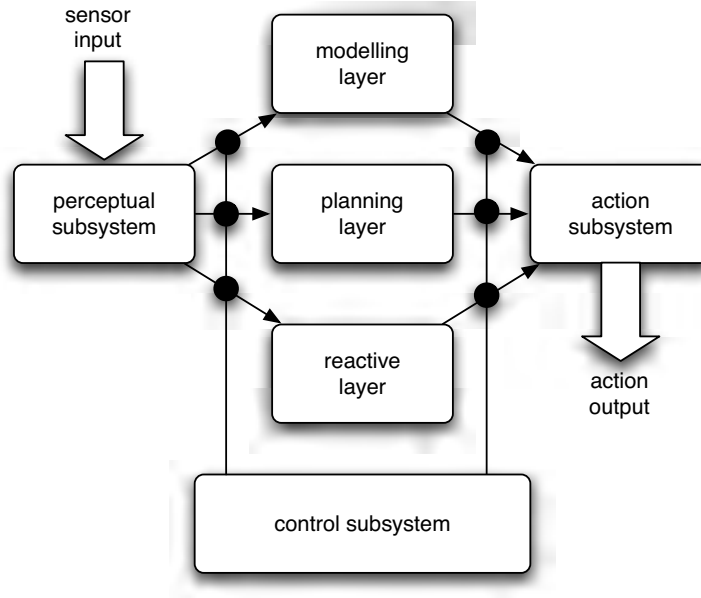


Figure 1.7: TOURINGMACHINES: a horizontally layered agent architecture.

order for a vertically layered architecture to make a decision, control must pass between *each* different layer. This is not fault tolerant: failures in any one layer are likely to have serious consequences for agent performance.

In the remainder of this section, we will consider two examples of layered architectures: Innes Ferguson’s TOURINGMACHINES, and Jörg Müller’s INTERRAP. The former is an example of a horizontally layered architecture; the latter is a (two-pass) vertically layered architecture.

3.4.1 TouringMachines

The TOURINGMACHINES architecture is illustrated in Figure 1.7. As this figure shows, TOURINGMACHINES consists of three *activity producing layers*. That is, each layer continually produces “suggestions” for what actions the agent should perform. The *reactive layer* provides a more or less immediate response to changes that occur in the environment. It is implemented as a set of situation-action rules, like the behaviors in Brooks’s subsumption architecture (Section 3.2). These rules map sensor input directly to effector output. The original demonstration scenario for TOURINGMACHINES was that of autonomous vehicles driving between locations through streets populated by other similar agents. In this scenario, reactive rules typically deal with functions like obstacle avoidance. For example, here is an example of a reactive rule for avoiding the kerb (from [15,

p. 59]):

```
rule-1: kerb-avoidance
  if
    is-in-front(Kerb, Observer) and
    speed(Observer) > 0 and
    separation(Kerb, Observer) < KerbThreshHold
  then
    change-orientation(KerbAvoidanceAngle)
```

Here `change-orientation(...)` is the action suggested if the rule fires. The rules can only make references to the agent's current state – they cannot do any explicit reasoning about the world and on the right-hand side of rules are *actions*, not predicates. Thus if this rule fired, it would not result in any central environment model being updated, but would just result in an action being suggested by the reactive layer.

The TOURINGMACHINES *planning layer* achieves the agent's proactive behavior. Specifically, the planning layer is responsible for the “day-to-day” running of the agent – under normal circumstances, the planning layer will be responsible for deciding what the agent does. However, the planning layer does not do “first-principles” planning. That is, it does not attempt to generate plans from scratch. Rather, the planning layer employs a *library* of plan “skeletons,” called *schemas*. These skeletons are in essence hierarchically structured plans, which the TOURINGMACHINES planning layer elaborates at run-time in order to decide what to do. So, in order to achieve a goal, the planning layer attempts to find a schema in its library that matches that goal. This schema will contain subgoals, which the planning layer elaborates by attempting to find other schemas in its plan library that match these subgoals.

The *modeling* layer represents the various entities in the world (including the agent itself, as well as other agents). The modeling layer thus predicts conflicts between agents, and generates new goals to be achieved in order to resolve these conflicts. These new goals are then posted down to the planning layer, which makes use of its plan library in order to determine how to satisfy them.

The three control layers are embedded within a *control subsystem*, which is effectively responsible for deciding which of the layers should have control over the agent. This control subsystem is implemented as a set of *control rules*. Control rules can either *suppress* sensor information between the control rules and the control layers, or else *sensor* action outputs from the control layers. Here is an example of a sensor rule [17, p. 207]:

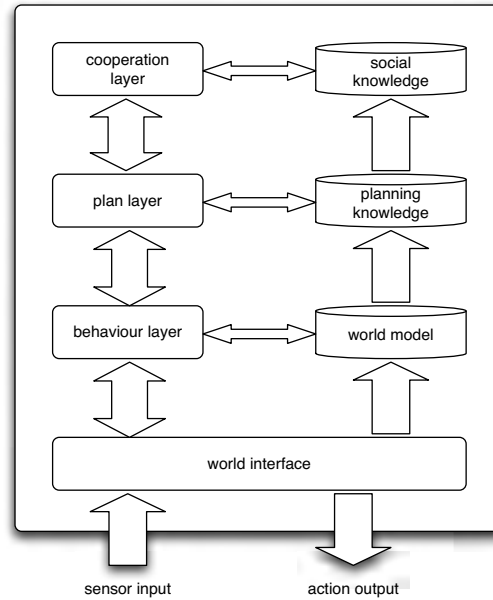


Figure 1.8: INTERRAP: a vertically layered two-pass agent architecture.

```

censor-rule-1:
  if
    entity(obstacle-6) in perception-buffer
  then
    remove-sensory-record(layer-R, entity(obstacle-6))

```

This rule prevents the reactive layer from ever knowing about whether `obstacle-6` has been perceived. The intuition is that although the reactive layer will in general be the most appropriate layer for dealing with obstacle avoidance, there are certain obstacles for which other layers are more appropriate. This rule ensures that the reactive layer never comes to know about these obstacles.

3.4.2 InteRRaP

INTERRAP is an example of a vertically layered two-pass agent architecture – see Figure 1.8.

As Figure 1.8 shows, INTERRAP contains three control layers, as in *TOURING-MACHINES*. Moreover, the purpose of each INTERRAP layer appears to be rather similar to the purpose of each corresponding *TOURINGMACHINES* layer. Thus the lowest (*behavior-based*) layer deals with reactive behavior; the middle (*local planning*) layer deals with everyday planning to achieve the agent’s goals, and the

uppermost (*cooperative planning*) layer deals with social interactions. Each layer has associated with it a *knowledge base*, i.e., a representation of the world appropriate for that layer. These different knowledge bases represent the agent and its environment at different levels of abstraction. Thus the highest level knowledge base represents the plans and actions of other agents in the environment; the middle-level knowledge base represents the plans and actions of the agent itself; and the lowest level knowledge base represents “raw” information about the environment. The explicit introduction of these knowledge bases distinguishes TOURINGMACHINES from INTERRAP.

The way the different layers in INTERRAP conspire to produce behavior is also quite different from TOURINGMACHINES. The main difference is in the way the layers interact with the environment. In TOURINGMACHINES, each layer was directly coupled to perceptual input and action output. This necessitated the introduction of a supervisory control framework, to deal with conflicts or problems between layers. In INTERRAP, layers interact with *each other* to achieve the same end. The two main types of interaction between layers are *bottom-up activation* and *top-down execution*. Bottom-up activation occurs when a lower layer passes control to a higher layer because it is not *competent* to deal with the current situation. Top-down execution occurs when a higher layer makes use of the facilities provided by a lower layer to achieve one of its goals. The basic flow of control in INTERRAP begins when perceptual input arrives at the lowest layer in the architecture. If the reactive layer can deal with this input, then it will do so; otherwise, bottom-up activation will occur, and control will be passed to the local planning layer. If the local planning layer can handle the situation, then it will do so, typically by making use of top-down execution. Otherwise, it will use bottom-up activation to pass control to the highest layer. In this way, control in INTERRAP will flow from the lowest layer to higher layers of the architecture, and then back down again.

The internals of each layer are not important for the purposes of this chapter. However, it is worth noting that each layer implements two general functions. The first of these is a *situation recognition and goal activation* function. This function acts rather like the *options* function in a BDI architecture (see Section 3.3). It maps a knowledge base (one of the three layers) and current goals to a new set of goals. The second function is responsible for *planning and scheduling* – it is responsible for selecting which plans to execute, based on the current plans, goals, and knowledge base of that layer.

Layered architectures are currently the most popular general class of agent architecture available. Layering represents a natural decomposition of functionality: it is easy to see how reactive, proactive, social behavior can be generated by the reactive, proactive, and social layers in an architecture. The main problem with

layered architectures is that while they are arguably a *pragmatic* solution, they lack the conceptual and semantic clarity of unlayered approaches. In particular, while logic-based approaches have a clear logical semantics, it is difficult to see how such a semantics could be devised for a layered architecture. Another issue is that of interactions between layers. If each layer is an independent activity producing process (as in *TOURINGMACHINES*), then it is necessary to consider all possible ways that the layers can interact with one another. This problem is partly alleviated in two-pass vertically layered architectures such as *INTERRAP*.

3.4.3 Sources and Further Reading

The introductory discussion of layered architectures given here draws heavily upon [41, pp. 262–264]. The best reference to *TOURINGMACHINES* is [15]; more accessible references include [16, 17]. The definitive reference to *INTERRAP* is [40], although [19] is also a useful reference. Other examples of layered architectures include the subsumption architecture [9] (see also Section 3.2), and the 3T architecture [4].

4 Conclusions

I hope that after reading this chapter, you understand what agents are and why they are considered to be an important area of research and development. The requirement for systems that can operate autonomously is very common. The requirement for systems capable of *flexible* autonomous action, in the sense that I have described in this chapter, is similarly common. This leads me to conclude that intelligent agents have the potential to play a significant role in the future of software engineering. Intelligent agent research is about the theory, design, construction, and application of such systems. This chapter has focused on the design of intelligent agents. It has presented a high-level, abstract view of intelligent agents, and described the sort of properties that one would expect such an agent to enjoy. It went on to show how this view of an agent could be refined into various different types of agent architectures — purely logical agents, purely reactive/behavioral agents, BDI agents, and layered agent architectures.

5 Exercises

1. **Level 1** Give other examples of agents (not necessarily intelligent) that you know of. For each, define as precisely as possible:

- (a) the environment that the agent occupies (physical, software, ...), the states that this environment can be in, and whether the environment is: accessible or inaccessible; deterministic or non-deterministic; episodic or non-episodic; static or dynamic; discrete or continuous.
 - (b) the action repertoire available to the agent, and any preconditions associated with these actions.
 - (c) the goal, or design objectives, of the agent – what it is intended to achieve.
2. **Level 2** The following few questions refer to the vacuum world example. Give the full definition (using pseudo-code if desired) of the *new* function, which defines the predicates to add to the agent's database.
 3. **Level 2** Complete the vacuum world example by filling in the missing rules. How intuitive do you think the solution is? How elegant is it? How compact is it?
 4. **Level 2** Try using your favorite (imperative) programming language to code a solution to the basic vacuum world example. How do you think it compares to the logical solution? What does this tell you about trying to encode essentially *procedural* knowledge (i.e., knowledge about what action to perform) as purely logical rules?
 5. **Level 2** If you are familiar with PROLOG, try encoding the vacuum world example in this language and running it with randomly placed dirt. Make use of the `assert` and `retract` meta-level predicates provided by PROLOG to simplify your system (allowing the program itself to achieve much of the operation of the *next* function).
 6. **Level 2** Develop a solution to the vacuum world example using the subsumption architecture. How does it compare to the logic-based example?
 7. **Level 2** Try scaling the vacuum world up to a 10×10 grid size. Approximately how many rules would you need to encode this enlarged example, using the approach presented above? Try to generalize the rules, encoding a more general decision-making mechanism.
 8. **Level 3** Suppose that the vacuum world could also contain *obstacles*, which the agent needs to avoid. (Imagine it is equipped with a sensor to detect such obstacles.) Try to adapt the example to deal with obstacle detection and avoidance. Again, compare a logic-based solution to one implemented in a traditional (imperative) programming language.

9. **Level 3** Suppose the agent's sphere of perception in the vacuum world is enlarged, so that it can see the *whole* of its world, and see *exactly* where the dirt lay. In this case, it would be possible to generate an *optimal* decision-making algorithm – one which cleared up the dirt in the smallest time possible. Try and think of such general algorithms, and try to code them both in first-order logic and a more traditional programming language. Investigate the effectiveness of these algorithms when there is the possibility of *noise* in the perceptual input the agent receives (i.e., there is a non-zero probability that the perceptual information is wrong) and try to develop decision-making algorithms that are robust in the presence of such noise. How do such algorithms perform as the level of perception is reduced?
10. **Level 3** Try developing a solution to the “distant planet exploration” example (see page 21) using the logic-based approach. How does it compare to the reactive solution?
11. **Level 3** In the programming language of your choice, implement the “distant planet exploration” example using the subsumption architecture. (To do this, you may find it useful to implement a simple subsumption architecture “shell” for programming different behaviors.) Investigate the performance of the two approaches described, and see if you can do better.
12. **Level 3** Using the simulator implemented for the preceding question, see what happens as you increase the number of agents. Eventually, you should see that overcrowding leads to a suboptimal solution – agents spend too much time getting out of each other's way to get any work done. Try to get around this problem by allowing agents to pass samples to each other, thus implementing *chains*. (See the description in [14, p. 305].)
13. **Level 3** Read about traditional *control theory*, and compare the problems and techniques of control theory to what we are trying to accomplish in building intelligent agents. How are the techniques and problems of traditional control theory similar to those of intelligent agent work, and how do they differ?
14. **Level 4** One advantage of the logic-based approach to building agents is that the logic-based architecture is *generic*: first-order logic turns out to be extremely powerful and useful for expressing a range of different properties. Thus it turns out to be possible to use the logic-based architecture to *encode* a range of other architectures. For this exercise, you should attempt to use first-order logic to encode the different architectures (reactive, BDI, layered) described in this chapter. (You will probably need to read the

original references to be able to do this.) Once completed, you will have a logical *theory* of the architecture, which will serve both as a formal specification of the architecture, and also as a precise mathematical model of it, amenable to proof. Once you have your logically-specified architecture, try to *animate* it by mapping your logical theory of it into, say, the PROLOG programming language. What compromises do you have to make? Does it seem worthwhile trying to directly program the system in logic, or would it be simpler to implement your system in a more pragmatic programming language (such as JAVA)?

References

- [1] P. Agre and D. Chapman. PENG! : An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272, Seattle, WA, 1987.
- [2] P. E. Agre and S. J. Rosenschein, editors. *Computational Theories of Interaction and Agency*. The MIT Press: Cambridge, MA, 1996.
- [3] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A framework for programming in temporal logic. In *REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (LNCS Volume 430)*, pages 94–129. Springer-Verlag: Berlin, Germany, June 1989.
- [4] R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 187–202. Springer-Verlag: Berlin, Germany, 1996.
- [5] G. Booch. *Object-Oriented Analysis and Design (second edition)*. Addison-Wesley: Reading, MA, 1994.
- [6] R. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.
- [7] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA, 1987.
- [8] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
- [9] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

- [10] R. A. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–15. The MIT Press: Cambridge, MA, 1990.
- [11] R. A. Brooks. Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991.
- [12] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
- [13] O. Etzioni. Intelligence without robots. *AI Magazine*, 14(4), December 1993.
- [14] J. Ferber. Reactive distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 287–317. John Wiley & Sons, 1996.
- [15] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Clare Hall, University of Cambridge, UK, November 1992. (Also available as Technical Report No. 273, University of Cambridge Computer Laboratory).
- [16] I. A. Ferguson. Towards an architecture for adaptive, rational, mobile agents. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 249–262. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.
- [17] I. A. Ferguson. Integrated control and coordinated behaviour: A case for agent models. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 203–218. Springer-Verlag: Berlin, Germany, January 1995.
- [18] J. A. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 202–206, Milan, Italy, 1987.
- [19] K. Fischer, J. P. Müller, and M. Pischel. A pragmatic BDI architecture. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 203–218. Springer-Verlag: Berlin, Germany, 1996.
- [20] M. Fisher. A survey of Concurrent METATEM — the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag: Berlin, Germany, July 1994.

- [21] L. Gasser and J. P. Briot. Object-based concurrent programming and DAI. In N. M. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–108. Kluwer Academic Publishers: Dordrecht, The Netherlands, 1992.
- [22] M. R. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1987.
- [23] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.
- [24] M. P. Georgeff and A. S. Rao. A profile of the Australian AI Institute. *IEEE Expert*, 11(6):89–92, December 1996.
- [25] F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors. *Building Expert Systems*. Addison-Wesley: Reading, MA, 1983.
- [26] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley: Reading, MA, 1986.
- [27] N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek, and L. Z. Varga. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control. *IEEE Expert*, 11(6):60–88, December 1996.
- [28] L. P. Kaelbling. An architecture for intelligent reactive systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions & Plans — Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
- [29] L. P. Kaelbling. A situated automata approach to the design of embedded agents. *SIGART Bulletin*, 2(4):85–88, 1991.
- [30] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- [31] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In P. Maes, editor, *Designing Autonomous Agents*, pages 35–48. The MIT Press: Cambridge, MA, 1990.
- [32] D. Kinny and M. Georgeff. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 82–88, Sydney, Australia, 1991.
- [33] K. Konolige. *A Deduction Model of Belief*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1986.

- [34] Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Berlin, Germany, 1996.
- [35] P. Maes. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 991–997, Detroit, MI, 1989.
- [36] P. Maes, editor. *Designing Autonomous Agents*. The MIT Press: Cambridge, MA, 1990.
- [37] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. The MIT Press: Cambridge, MA, 1990.
- [38] P. Maes. The agent network architecture (ANA). *SIGART Bulletin*, 2(4):115–120, 1991.
- [39] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
- [40] J. P. Müller. *The Design of Intelligent Agents (LNAI Volume 1177)*. Springer-Verlag: Berlin, Germany, 1997.
- [41] J. P. Müller, M. Pischel, and M. Thiel. Modelling reactive behaviour in vertically layered agent architectures. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 261–276. Springer-Verlag: Berlin, Germany, January 1995.
- [42] J. P. Müller, M. Wooldridge, and N. R. Jennings, editors. *Intelligent Agents III (LNAI Volume 1193)*. Springer-Verlag: Berlin, Germany, 1997.
- [43] N. J. Nilsson. Towards agent programs with circuit semantics. Technical Report STAN-CS-92-1412, Computer Science Department, Stanford University, Stanford, CA 94305, January 1992.
- [44] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, 1994.
- [45] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038)*, pages 42–55. Springer-Verlag: Berlin, Germany, 1996.

- [46] A. S. Rao. Decision procedures for propositional linear-time Belief-Desire-Intention logics. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 33–48. Springer-Verlag: Berlin, Germany, 1996.
- [47] A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear time and branching time intention logics. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 498–504, Sydney, Australia, 1991.
- [48] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
- [49] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.
- [50] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.
- [51] A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.
- [52] R. Reiter. *Knowledge in Action*. The MIT Press: Cambridge, MA, 2001.
- [53] S. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
- [54] S. J. Rosenschein and L. P. Kaelbling. A situated view of representation and control. In P. E. Agre and S. J. Rosenschein, editors, *Computational Theories of Interaction and Agency*, pages 515–540. The MIT Press: Cambridge, MA, 1996.
- [55] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.
- [56] S. Russell and D. Subramanian. Provably bounded-optimal agents. *Journal of AI Research*, 2:575–609, 1995.

-
- [57] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy, 1987.
- [58] L. Steels. Cooperation between distributed agents through self organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 175–196. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
- [59] M. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, February 1997.
- [60] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

Chapter 2

Multiagent Organizations

Virginia Dignum and Julian Padget

1 Introduction

The previous chapter discusses the design of intelligent agents. Each agent is an individual entity capable of independent action. However, many applications require the interaction between several individuals in order to realize a certain goal. Think, for instance, of a logistics system that coordinates transport and storage of different goods belonging to different owners, using different transportation forms. Another example is a network of sensors that monitors traffic in a busy intersection. If we think of each sensor as an intelligent agent, then it is clear that they should be able to coordinate their activities. Such systems, composed of several agents, are called *Multiagent Systems* (MAS).

This chapter builds on the concept of agent put forward in Chapter 1, in which the agent is seen as situated in an environment in which it may sense and upon which it may act, and consequently a multiagent system is seen as a collection of agents that are concurrently sensing and acting on an environment. The premise of multiagent systems is that an agent can be more effective in the context of others because it can concentrate on tasks within its competence, delegate other tasks, and use its ability to communicate, coordinate, and negotiate to achieve its goals. But how can a collection of agents achieve the desired level of mutual coordination? Moreover, how can MAS account for global goals, which are not necessarily assumed by any of the agents?

A multiagent system often cannot be fully described by the sum of all the descriptions of the agents in it. Not only that not all problems can be easily described

in terms of individual mental states, but also, in many cases, situations are better described based on activities and constraints that characterize the externally observable behavior of the whole population. Think, for example, of resources that are collectively owned or shared between or among communities, such as a shared Internet connection. If each agent follows its own goal of having as much and as frequent access to the Internet as possible, soon the quality of the connection of each will be very low because they all will be limiting bandwidth to each other. This example demonstrates the need to somehow “organize” these agents so that all benefit (possible solutions are to divide access time equally between them, or to use a pay-per-minute system, or to make a rooster, etc.). Sometimes, such “organization” will emerge from the interactions between the agents, but in many cases it is an entity in itself which has its own goals, that regulates the agents in the environment.

Since their origin in the 1980s, Multiagent Systems (MAS) have often been defined as organizations or societies of agents, i.e., as a set of agents that interact together to coordinate their behavior and often cooperate to achieve some collective goal [28]. The term *agent organization* has become commonplace within the MAS community, but is often used to refer to different, even incompatible, concepts. In short, some take organization as the process of organizing a set of individuals, whereas others see organization as an entity in itself, with its own requirements and objectives. These differences are in large part due to the diverse worldviews and backgrounds of different research fields, namely sociology and organization theory (OT), on the one hand, and distributed artificial intelligence (DAI), on the other hand. From a sociological perspective, agent organization is specified independently of its participants and relates the structure of a (complex) system to its externally observable global behavior. The artificial intelligence view on MAS is mostly directed to the study of the mental state of individual agents and their relation to the overall behavior of the system.

The main focus of this chapter is on the idea of organization as an entity itself, which is different from the agents in it. Taking an example from human organizations, think for instance of a university. As an entity, it clearly has its own goals (e.g., being a place of learning), its plans (e.g., admitting students or producing research), and its design (e.g., departments, positions, and activities). However, in itself, the university cannot do anything! In order to achieve its goals, the university is dependent on the individuals it will attract to fulfill its positions (i.e., lecturers, students, and so on). On the other hand, an agent with the goal of getting a degree in computer science is dependent on the existence of a university that awards it. So, both agents and organizations are dependent on each other.

This chapter will show that organizations are an essential aspect of multiagent systems, because they can complement the concept of agents in that they allow

for simplified agent models through reducing uncertainty. This chapter discusses comprehensive frameworks to build multiagent organizations, based on notions from human societies, in which we identify roles, assign responsibilities, and specify permissions – among other things. We will discuss structural and institutional approaches to multiagent organizations, where the former is mostly concerned with the specification and enactment of organization goals, and the latter sees organizations as regulative instruments for interaction.

The remainder of this chapter is organized as follows. In Section 2 we set out the broader context of multiagent systems and organizations, identifying the drivers for bringing organizational issues into MAS; describe the different views on organization; and introduce the conference management scenario used throughout this chapter to illustrate different concepts. In Section 3 we examine the elements and key properties of organizations, concluding with an example of organizational modeling using the OperA framework. This perspective is complemented in Section 4 by a presentation of institutions and their formal underpinnings, followed by an example of institutional modeling using InstAL. In Section 5 we move to the agent perspective to consider how organizations appear from the position of agents. Finally, in Section 6 we examine motivations and mechanisms for organizational change.

2 Background

In this section, we describe the evolution of the organization perspective in MAS, and discuss how different sources of inspiration have lead to different approaches to organization in MAS.

2.1 From Intelligent Agents to Multiagent Systems

A multiagent system is a system composed of multiple interacting intelligent agents that interact to solve problems that are beyond the individual capabilities or knowledge of each individual. In [68], MAS are characterized as systems where (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous. That is, MAS can be seen as *distributed problem solvers* (DPS) [26] used to solve problems that are difficult or impossible to solve by an individual agent or a monolithic system. In those situations, a central controller is either not feasible, or one wants to make good use of an existing distribution of resources. A good example is a sensor network, which consists of multiple processing units, each with local sensor capabilities, limited processing power, limited power supply, and limited communication bandwidth.

Despite these limitations, sensor networks are able to provide some global services.

Research in MAS is concerned with the modeling, analysis, and construction of a collection of possibly pre-existing, autonomous agents that interact with each other and their environments. Agents are considered to be autonomous entities, such as software programs or robots. In MAS, the study of such systems goes beyond the study of individual intelligence to consider, in addition, problem solving that has social components. Interactions in MAS can be either cooperative or selfish. That is, the agents can share a common goal (e.g., an ant colony), or they can pursue their own interests (as in a free market economy). In cooperative situations, agents collaborate to achieve a common goal, shared between the agents or, alternatively, the goal of a central designer who is designing the various agents [63]. Interaction between selfish agents usually uses coordination techniques based on auctions or other resource sharing mechanisms. The definition of coordination mechanisms between different agent types is a major part of the MAS, resulting in implementations where social aspects, goals, and behaviors are part of the architecture of the specific agents [55].

From an engineering perspective, the development of MAS is not straightforward. Even though it is commonly accepted that “some way of structuring the society is typically needed to reduce the system’s complexity, to increase the system’s efficiency, and to more accurately model the problem being tackled” [39], in many situations, individuals do not share nor necessarily pursue the same aims and requirements as the global system or society to which they belong. In these cases, the view of coordination and control needs to be expanded to consider not only an agent-centric perspective but also takes a societal focus. However, many approaches assume a predefined agent type or class when designing MAS. A disadvantage is that systems are then closed for agents that are not able to use the same type of coordination and behavior, and that all global characteristics and requirements are implemented in the individual agents and not outside them.

The pursuit of open solutions to MAS, which are independent of the architectures of a particular set of agents, has led to the concept of organization-based approaches to MAS. These can be broadly divided into two main categories: *structural* and *institutional*. Structural approaches see organizations mainly as ways to provide the means for coordination, which enable the achievement of global goals. Institutions are mechanisms used to regulate social action by defining and upholding norms. According to Ostrom, institutions are rules-in-use that structure organizations and activities [52].

So far, we have discussed organizations, primarily from a design perspective, as a mechanism that can be crafted to enable coordination between agents. A complementary view of organizations stems from emergence, that starts from the

assumption that MAS require no real structure or patterns and the outcomes of interaction are unpredictable. From an emergent perspective, MAS are viewed as a population of (simple) agents interacting locally with one another and with their environment. In these systems, organization is not designed but is taken as an externally observable result of the collective behavior of agents. Communication is often based on modification of the environment (stigmergy). There is no centralized control structure dictating how individual agents should behave, meaning that local interactions between such agents are expected to lead to the emergence of global behavior. Emergent agent systems are mostly used in the domains of social simulation, adaptive planning, logistics, and artificial life.

In this chapter, we mostly follow the design view on organization, but will also discuss the emergence view whenever relevant.

2.2 From Multiagent Systems to Multiagent Organizations

An important differentiation in organizational approaches is that of “focus,” leading to the distinction between *agent-centric* approaches which implicitly encode organizational aspects at the agent level and *organization-centric* approaches which provide an explicit, referenceable representation of organizational components.

Multiagent systems, as outlined in Section 2.1, offer a view in which agents communicate, negotiate, and collaborate directly with one another. The organizational structure exists within the individual agents’ state, and their propensity to organize is typically predetermined by their encoded behaviors. The attraction of this approach is that the agents themselves encapsulate the organization, but this also means the organization has no representation because it is only implicitly observable through the agents’ interactions. However, because the organization is distributed and hard-wired, it is hard to maintain, revise, and scale up. Furthermore, such solutions often lead to systems with high entry barriers – because third-party agents must somehow have in themselves the knowledge of how to interact with existing agents correctly and maintain the integrity of the system.

The response is to refactor the system to build an explicit representation of the organization, external to the agents, which opens the door to open systems, because the agents and, in particular, new entrants to the system can then reason about the organizational structure, goals, and policies, if presented in a suitable form.

In organization-centric approaches, organizations are, in general, conceptualized in terms of their structure, that is, the pattern of information and control relations that exist among agents and the distribution of problem-solving capabilities among them. An organization structure provides a framework for agent interactions through the definition of roles, behavior expectations, and authority

relations. In cooperative problem solving, a structure gives each agent a high-level view of how the group solves problems. Coordination is often achieved by communication between agents (using an agent communication infrastructure such as FIPA), but interaction through the environment is also common, particularly in systems where agents are very simple (e.g., in swarm systems, agents interact implicitly through the use of resources in the environment). Chapter 3 explores agent communication in much greater depth.

Organization structure provides scope for interactions, enables strength in numbers, reduces or manages uncertainty, and addresses whether there is need for redundancy. At the same time, organizations can also adversely affect computational or communication overhead, reduce overall flexibility or reactivity, and add an additional layer of complexity to the system. This requires careful thought on the design of organization structures.

Organization-oriented approaches to MAS are well-suited to understanding and designing MAS in ways that extend and complement traditional agent-centric approaches. Multiagent organizations can be understood as complex entities where a multitude of agents interact, within a structured environment aimed at some global purpose. This is of particular importance for MAS in complex, dynamic, distributed domains.

In the following, the view of an organization as a basic explicit component of MAS is elaborated further, starting from its grounds in organizational and social sciences to its concrete use in MAS engineering frameworks.

2.3 Sources of Inspiration

In the last decade, the field of multiagent systems has increasingly applied ideas, paradigms, and theories from research in human organizations as a means to understand and model distributed complex domains. In particular, research in organization theory, institutional analysis, and rationality theory has been crucial for the development of the organizational views now being used in MAS.

2.3.1 Organization as Structure

Organization theory has for many decades investigated the issue of organizational structure. An organizational structure has essentially two objectives [25]: first, it facilitates the flow of information within the organization in order to reduce the uncertainty of decision making; secondly, the structure of the organization should combine organizational behavior across the parts of the organization so it is coordinated. In deciding what kind of organization structure to use, decision makers need to understand the characteristics of the environment they are in and the demands of this environment in terms of information and coordination requirements.

Organizational Form	Machine Bureaucracy	Professional Organization	Entrepreneurial Startup	Adhocracy
Complexity	Low (simple)	High (complex)	Low (simple)	High (complex)
Pace of Change	Low (stable)	Low (stable)	High (dynamic)	High (dynamic)
Coordination Mechanism	Standardize procedures and outputs	Standardize skills and norms	Direct supervision and control	Mutual adjustment of ad-hoc teams

Table 2.1: Organizational forms according to Mintzberg.

Organizational design therefore concerns the allocation of resources and people to a specified mission or purpose and the structuring of these resources to achieve the mission [29]. Given that there are no exact recipes to construct the optimal organization, the evaluation of design and determination of its appropriateness – given the organization’s aims and constraints – are main issues in organization theory.

Notable in organization theory is the work of Mintzberg on the classification of organizational structures [47]. According to Mintzberg, environmental variety is determined by both environmental complexity and the pace of change. He identifies four types of organizational forms, which are associated with the four possible combinations of complexity and change. Each of the four forms of organization depend on fundamentally different coordination mechanisms. Table 2.1 summarizes Mintzberg’s taxonomy of organizations.

Inspired by and extending upon the work of Mintzberg, researchers in organizational theory have proposed a growing number of classification systems for organization structures, such as bureaucratic, matrix, virtual enterprise, network, boundary-less, conglomerate, alliance, among many others. However, often, definitions and naming of organizational forms are unclear, and the classification of a specific organization into one of the proposed classes is not trivial, often resulting in a hybrid classification.

Design strategies determine the focus of the design process. Some organizations place a higher priority on efficiency, preferring designs that minimize the costs of producing goods or services. Others emphasize effectiveness, focusing on generating revenues or seizing leading-edge innovation in the marketplace [9].

From the perspective of developing MAS, one of the main contributions of organization theory is the idea that models for organizational design should enable the separation of collective from individual concerns. This is achieved by providing abstract representations of interactions, environments, objectives on the one hand, and of the participating entities, on the other hand, which enables the analysis of their partial contributions to the overall performance of the organization. Horling and Lesser have analyzed different organizational forms commonly

used in MAS, such as hierarchies, coalitions, teams, federations, markets, and societies. In [37], they give some insight into how they can be used and generated, and compare their strengths and weaknesses. The use of organizational structures in MAS is explored further in Section 3.

2.3.2 Organization as Institution

Social sciences look at organization from the perspective of social behavior, that is, behavior directed towards society, or taking place between members of the same group or species. While strategic management and organizational theory, as discussed in Section 2.3.1, study the *macro* behavior of whole organizations and industries, how they adapt, and the strategies, structures, and contingencies that guide them – in the social sciences, the interest on organizations is mostly geared to the study of *micro* organizational behavior, which focuses on individual and group dynamics in an organizational setting.

Social activity concerns interpersonal processes and structures related to interaction between people. According to Max Weber, “an action is *social* if the acting individual takes account of the behavior of others and is thereby oriented in its course.” Whenever people interact, many factors come into play. In this sense, organizing can be seen as the collective attempt to manage a common resource, which brings with it the problems of “coping with free-riding, solving commitment problems, arranging for the supply of new institutions, and monitoring individual compliance with sets of rules.” [51].

In sociology, behavioral expectations within a society or group are defined as *norms*. A norm is a more or less general rule of conduct within a group or society, which constitutes a link between the abstract values (goals) and concrete behavior that is considered to serve one or more of those goals. As such, norms are the explicit, or implicit, rules that a group uses to determine appropriate and inappropriate values, beliefs, attitudes, and behaviors [19].

According to Ostrom, institutions are rules-in-use that structure organizations and activities [52]. Institutions are essential to building trust and represent significant investment of time and effort to increase results and reduce social costs. This complements the view introduced by North and commonly used in economics, which defines institutions as “the rules of the game in a society, or more formally, the humanly devised constraints that shape social interaction” [49]. Institutions operate in an environment consisting of individuals and other institutions that affect each other.

Norms are one of the ways to express the criteria for rightness and wrongness, correctness and incorrectness, that characterize, for example, formulae for greeting in social relationships. Norms are shared by a community, and may have no individual representations [36]. In most cases, work on norm compliance as-

sumes norms to be implemented and enforced by an institution. This perspective both sees the institution as active and as an entity (firm, government, company), which is the warrant of those conventions [49]. Clearly, this conflicts with the social sciences definition of institution as a set of rules and conventions and presents a source of potential confusion if not disambiguated.

In multiagent systems, institutions are used to regulate interactions where agents establish commitments and to facilitate that these commitments are upheld. Such institutions are devices, external to the agents, which facilitate agent interactions; that is, according to Simon's design view, institutions are coordination artifacts that constitute an interface between the internal, rational decision-making capabilities of agents and the social effect of their interactions [65].

Work on norms and institutions in multiagent systems (MAS) has resulted in broadly two different approaches [20]: a *regimented* view of norms [27, 54, 59] in which norms are viewed as constraints, and a *regulated* view [1, 12, 33] in which norm enforcement is explicit and agents' motivations play an important role in compliance decisions. When regimenting norms, all agent actions leading to a violation of those norms are impossible. That is, the design of the system makes it impossible to perform forbidden actions – think of gates at the metro station that prevent entrance without a ticket. From an engineering perspective, these approaches are easier to implement, but they seriously limit agent autonomy. Regulated approaches require both the establishment of institutions – together with their representative agents that can monitor and enforce norm compliance – and the presence of normative agents that are able to reason about the effect of norms on goal achievement.

We will further discuss the use of institutions in MAS in Section 4.

2.3.3 Organization as Agent

From a holistic perspective, organizations can be seen as active entities (typically socio-technological systems) that act in an environment. As a system, “organizations are economic units with a production function that transforms inputs into outputs, with efficiency realized through the choice of optimal factor proportions” [78]. Systems theory sees an organization as more than the sum of its various parts, and, in particular, organizational behavior as dependent on the interrelationships of its parts [58].

As an agent, an organization can affect and be affected by its environment and by other agents with whom it interacts. This is the view of agency theory, i.e., the principal-agent paradigm, as it is known in economics, which deals with the difficulties that arise under conditions of incomplete and asymmetric information, when one party (the principal) delegates work to another (the agent), who performs that work. Classical economics views the firm as a single decision-unit

engaged in maximizing profits. It ignores the possibility of conflict between owners, managers, and employees. Agency theory is concerned with two issues that can occur in agency relationships. The first occurs when (a) the desires or goals of the principal and agent conflict and (b) it is difficult or expensive for the principal to verify what the agent is actually doing. The problem here is that the principal cannot verify that the agent has behaved appropriately. The second issue is the problem of risk sharing, which arises when the principal and agent have different attitudes towards risk.

2.4 Autonomy and Regulation

In MAS, agents are assumed to be autonomous entities, pursuing their own individual goals based on their own beliefs and capabilities. From this perspective, global behavior emerges from individual interactions and therefore the final behavior of the whole system cannot be easily predicted, managed, or specified externally. However, in complex critical environments, the requirements, expectations, and overall behavior of the global system must be taken into account and structural characteristics of the domain have to be incorporated. As such, multi-agent organizations must consider not only the goals and characteristics of individuals, but also such organizational characteristics as stability over time, predictability, and commitment to predefined aims and strategies. These are not reducible to individual issues, and therefore must exist independently and externally to agent design.

The organization perspective on MAS recognizes that the modeling of interaction in MAS cannot simply rely on an agent's own architectures and capabilities. This implies that organizational design cannot be assumed to be the result of autonomous interaction, and, thus, organization must be pre-established. Organizational approaches arise from the idea that the effective engineering of MAS needs high-level, agent-independent concepts and abstractions that explicitly define the organization in which agents live [79]. These are the rules and global objectives that govern the activity of an enterprise, group, organization, or nation, which are specified externally and imposed on the participants. Just as MAS cannot assume that agents will coordinate their actions, organizational models cannot assume that participating agents will act according to the global needs and expectations. This creates a dichotomy between regulation and autonomy, which characterizes multi-agent organizations [75]. As such, multiagent organization specifications need to describe mechanisms designed to systematize, defend, and recommend right and wrong behavior, which can inspire trust into the agents that will join them.

Agent organizations reflect the idea that interactions occur not just by accident but aim at achieving some desired global goals. That is, there are goals external to each individual participant (or agent) that must be reached through their inter-

action. Desired behavior of an organization is therefore external to the participants and must be guaranteed by the organizational structure. However, assuming open environments where neither the internal architecture nor the actual aims of the agents can be verified, such guarantees on desired global behavior should be achieved without relying on the design of agents nor compromising the agents' autonomy [73].

A consequence of the considerations mentioned above is that agent organizations assume organizational structure to be determined by design, independently of the participants. From an organizational perspective, the main function of an individual agent is the enactment of a role that contributes to the global aims of the organization. From this perspective, organizational goals determine agent roles and interaction norms. Agents are then seen as the actors that perform role(s) described by the organization design.

However, the very notion of agent autonomy refers to the ability of individual agents to determine their own actions, plans, and beliefs. From the agent's perspective, its own capabilities and aims determine the reasons and the specific way an agent will enact its role(s), and the behavior of individual agents is motivated from their own goals and capabilities [16, 76]. That is, agents bring in their own ways into the society, in that they will follow their own goals and motivations and will bring in their own ways of doing things to the system. In other words, the actual behavior of the society emerges from the goal-pursuing behavior of the individual agents within the constraints set by the organization.

In summary, there is a clear need for multiagent frameworks that combine and use the potential of a group of agents for the realization of the objectives of the whole, without ignoring the individual aims and "personalities" of the autonomous participant agents. That is, in order to represent interactions between agents in such an open context, organizational framework must meet the following requirements [76]:

Internal autonomy: interaction and structure of the organization must be represented independently from the internal design of the agents.

Collaboration autonomy: activity and interaction in the organization must be specified without completely fixing in advance all interaction possibilities.

The first requirement relates to the fact that since an open organization allows the participation of multiple heterogeneous entities – the number, characteristics, and architecture of which are unknown to the designer – the design of the organization cannot be dependent on their individual designs. The second requirement highlights the fundamental tension between the goals of the organization and the autonomy of the participating entities. On the one hand, the more detail about

interactions provided by the organization design, the more requirements can be checked and guaranteed at design time. This allows us, for example, to ensure that certain rules are always followed. On the other hand, there are good reasons to allow the agents some degree of freedom, basically to enable their autonomy to choose their own way of achieving collaboration, and, as such, increase flexibility and adaptability.

Finally, different degrees of abstraction on organization specification have consequences to the level of autonomy required from each agent to accomplish organizational objectives. Intuitively, the more detailed specification given, the less alternative ways (and thus less autonomy) are available for the agents to achieve the organizational objectives. On the other hand, abstract organization models require more autonomy and reasoning capabilities from the agents, as they will need to be able to interpret the specification and decide on how to coordinate with others in each situation. This, however, provides higher flexibility of operation. It is then a design decision how to better regulate agent autonomy by deciding on the level of abstractness or concreteness of the organization description.

2.5 Example Scenario

In order to further describe multiagent organizations and the concepts introduced above, we will use as an illustration throughout this chapter the scenario of the management of an international conference (taken from [79]), shown in Figure 2.1.

3 Multiagent Organizations

In this section, we present in more detail the structural approach to multiagent organizations, as introduced in Section 2.3.1.

Organization structures define the formal lines of communication, allocation of information processing tasks, distribution of decision-making authority, and the provision of incentives. That is, organizations describe objectives, roles, interactions, and rules in an environment without considering the particular characteristics of the individuals involved. Organizational objectives are not necessarily shared by any of the individual participants, but can only be achieved through their combined action. In order to achieve its goals, it is thus necessary that an organization employs the relevant agents, and structures their interactions and responsibilities such that organizational objectives can be realized. The performance of an organization is therefore determined both by its interaction structures, and by the individual characteristics of the participating agents.

CONFERENCE MANAGEMENT SYSTEM

Setting up and running a conference is a multiphase process involving several individuals and groups, including organizing instance, authors, chairs, program committee (PC) members, and participants. Tasks and activities are either individual or require the coordination/cooperation of a set of roles: for instance, the PC chair depends on PC members to achieve the reviewing process and on the authors to get submissions; the organizer depends on session chairs to manage sessions; presenters depend on the organizer to have a suitable location, and so forth.

Agents associated with the persons involved in the process play roles in this organization, and therefore their behavior can be influenced by those persons. For example, an author could attempt to review his or her own paper or a PC member could try to deal with fewer papers than he or she should. As such, the organization must define the holding norms of behavior, which can, for instance, include the following: (a) papers must be submitted before the deadline; (b) reviewers must submit their evaluations on time; (c) all papers must be written in English; (d) reviewing must follow a single-blind (or double-blind) process.

During the submission phase, authors of submitted papers need to be informed that their papers have been received and they need to be assigned a submission number. Once the submission deadline has passed, the PC has to handle the review of the papers – contacting potential referees and asking them to review a number of the papers. After a while, reviews are expected to come in and be used to decide about the acceptance/rejection of the submissions. Authors need to be notified of these decisions, and in the event of acceptance, must be asked to produce the camera-ready version of their revised papers. Finally, the publisher has to collect the camera-ready versions from the authors and print the whole proceedings.

Figure 2.1: Conference management system.

Research on agent organizations applies normative, social, and organizational ideas coming from human societies into electronic distributed computational mechanisms for the design and analysis of MAS. Using organizational concepts as first-class modeling constructs [46] allows for a natural specification of open systems, and can describe both emergent and designed organizations. Just as in

human organizations, agent organizations describe how agents interact with each other and with the environment.

Implicit in the definition of organizations as instruments of purpose are the ideas that organizations have goals, or objectives, to be realized and, therefore, the shape, size, and characteristics of the organization affect its behavior [37]. Objectives of an organization are achieved through the action of the individuals in the organization, which means that an organization should make sure to employ the relevant actors, so that it can “enforce” the possibility of making its desires happen. Note that here an explicit distinction is made between the organization position, or role, and the actor, or agent. In this way, separation of concerns is possible according to the requirements described in Section 2.4. Furthermore, one of the main reasons for creating organizations is efficiency, that is, to provide the means for coordination that enables the achievement of global goals in an efficient manner. This means that the actors in the organization need to coordinate their activities in order to efficiently achieve those objectives.

3.1 Organization Concepts

Organization implies the existence of several coordinating entities that need to coordinate to realize some goal, which none individually has the capacity to achieve. As such, organization also implies the need for rules indicating how parts must be put together (i.e., the organizational structure relating roles to each other). Organizational structure can be defined as “what persists when components or individuals enter or leave an organization, i.e., the relationships that make an aggregate of elements a whole” [28].

The structure of a multiagent organization is the collection of roles, relationships, and authority structures that governs its behavior [37]. Roles describe the capabilities, objectives, rights, and obligations of different parties. By knowing one’s role, individuals can define expectations and establish plans. Organizational theory and sociology research have for a long time studied structure, dynamics, and behavior of human organizations and human interactions, based on the concept of role and relationships between roles.

A concrete organization is one possible instantiation of an organizational structure, where roles are filled in by specific individuals (which can be software agents, humans, or other systems). Social structure is thus an independent construct that describes and enables interaction between agents in a system. A social structure may be explicitly implemented in the form of a social artifact existing independently of the implementations of the agents, may be included in the implementations of the individual agents, or may exist only intangibly, in the form of the behavior patterns exhibited by the collective of agents during interaction [44].

Inspired by [67], components of organization can be classified into three broad classes. The first are *(task) environmental factors*, which include the components and features of the task (such as size, time constraints, uncertainty), the available resources, and the external conditions under which an organization operates. The second are the *structural factors*, which describe the components and features of the structure itself (such as roles, dependencies, constraints, norms, and regulations). The third class of factors are *agent factors*, which describe the characteristics of the individual agents concerning task capability, intelligence (including decision making and reasoning capabilities), social awareness, etc. To sum up, the three main aspects that must be represented in any model aimed at understanding or specifying organizational performance or behavior are:

1. **Environment** is the space in which organizations exist. This space is not completely controllable by the organization, and therefore results of activity cannot always be guaranteed. Environment can also include the description of tasks and resources (such as size and frequency), and is characterized by properties such as volatility, scarcity, unpredictability, and (task) complexity.
2. **Individuals or agents** are the acting, reasoning entities in the organization, which have the capability (partially) to control the state of some element in the environment. Agents are defined by their capabilities – typically describing their learning, communication, reasoning, and decision-making skills.
3. **Structure** describes the features of the organization, such as objectives, roles, relationships, and strategy. The roles and relationships holding in the organization determine control, coordination, and power relations. Differentiating dimensions of structure are size, degree of centralization, and formalization.

The ontological relations between these concepts, according to [24], are captured in Figure 2.2. Most organization models adopt similar classifications, but put different emphasis on the various parts of this ontology. For an overview, we refer to [15].

3.2 Example of Organization Modeling: The OperA Framework

Models to design organizations must enable the explicit representation of structural and strategic concerns and their adaptation to environmental changes in a way that is independent of the behavior of the agents.

In this section, we present an example of an organization modeling language, the OperA framework. Other well-known models include Tropos [8],

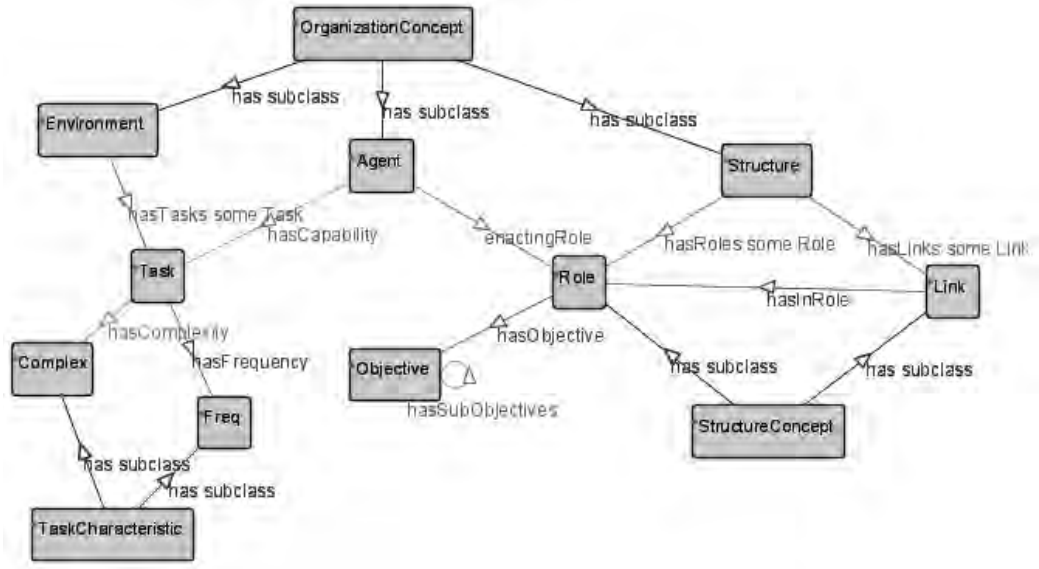


Figure 2.2: An excerpt of the organization ontology.

MOISE [38], or ORG4MAS [41], among others.

The OperA framework [21] consists of an organization modeling language and a supporting toolset for specification and analysis of organizations. OperA proposes an expressive language for defining organizations, distinguishing explicitly between the organizational aims, and the agents who act in it. That is, OperA enables the specification of organizational structures, requirements, and objectives, independently from agent choice criteria, which allows participants to have the freedom to act according to their own capabilities and demands. At the organization level, an OperA model describes the aims and concerns of the organization with respect to the social system. These are described as an organization's externally observable *objectives*, that is, the desired states of affairs for the organization. OperA supports the deployment of organizations in dynamic and unpredictable settings, and brings forth critical issues concerning the design, implementation, and validation of their behavior [34, 56, 76], guided by the following principles.

- Provide sufficient representation of the global organization requirements so that the overall system complies with its norms.
- Provide enough flexibility to accommodate heterogeneous participants so that agents can act according to their own characteristics and requirements.

OperA provides the means to represent concepts and relationships in a domain that is rich enough to cover the necessary contexts of agent interaction while keeping in mind the relevance of those concepts for the global aims of the system. The design

and validation of OperA organization models (OMs) can be done using the OperettA toolset [3]. OperettA is a combination of tools based on the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF), integrated into a single editor. Developed as an Eclipse plug-in, OperettA is fully open source and follows the model-driven engineering (MDE) principles of tool development. OperettA is available open source at <http://www.operettatool.nl>.

The OperA framework consists of three interrelated models. The **Organization Model** (OM) is the result of the observation and analysis of the domain and describes the desired behavior of the organization, as determined by the organizational stakeholders in terms of objectives, norms, roles, interactions, and ontologies. The OM is described in more detail in the remainder of this section, using as an example the conference organization scenario as in Section 2.5.

The OM provides the overall organization design that fulfills the stakeholders' requirements. Objectives of an organization are achieved through the action of agents, which means that, at each moment, an organization should employ the relevant agents that can make its objectives happen. However, the OM does not allow for specifying the individual agents. The **Social Model** (SM) maps organizational roles to (existing) agents and describes agreements concerning the role enactment and other conditions in enactment contracts. Finally, the **Interaction Model** (IM) describes the run-time interactions between role-enacting agents. The overall development process is depicted in Figure 2.3.

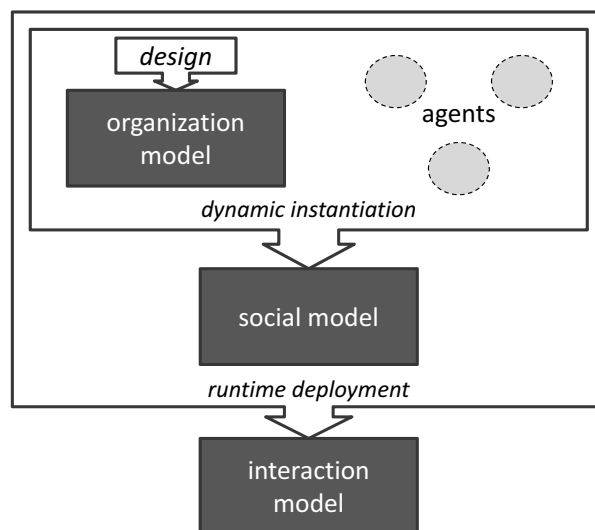


Figure 2.3: The OperA development process.

<i>Id</i>	PC_member
<i>Objectives</i>	paper_reviewed(Paper,Report)
<i>Sub-objectives</i>	{read(P), report_written(P, Rep), review_received(Org, P, Rep)}
<i>Rights</i>	access_confmanagement_system(<i>me</i>)
<i>Norms & Rules</i>	PC_member OBLIGED understand_english PC_member OBLIGED review_paper BEFORE deadline IF paper_by_colleague THEN PC_member FORBIDDEN review_paper

Table 2.2: *PC member* role description.

In this chapter, we focus on the OM that specifies the structure and global characteristics of a domain from an organizational perspective, e.g., how a conference should be organized, its program, submissions, etc. That is, the OM describes the means to achieve global objectives. Components of the OM are the *social* and *interaction structures*, in which global goals are specified in terms of roles and interactions. Moreover, organization specification should include the description of concepts holding in the domain, and of expected or required behaviors. Therefore, these structures should be linked with the norms, defined in *normative structure*, and with the ontologies and communication languages defined in the *communication structure*.

3.2.1 The Social Structure

The social structure describes the roles and dependencies holding in the organization. It consists of a list of role definitions, *Roles* (including their objectives, rights and requirements), a list of role groups' definitions, *Groups*, and a *Role Dependency's* graph. Examples of roles in the conference scenario are PC member, program chair, author, etc.

Global objectives form the basis for the definition of the objectives of roles. From the organization perspective, role descriptions should identify the activities and services necessary to achieve the organizational objectives and also to make it possible to abstract from the individuals that will eventually perform the role. From the agent perspective, roles specify the expectations of the society with respect to the agent's activity in the society. In OperA, the definition of a role consists of an identifier, a set of role objectives, possibly sets of sub-objectives per objective, a set of role rights, a set of norms, and the type of role. An example of a role description for a PC member in the conference scenario is depicted in Table 2.2.

Groups provide the means to collectively refer to a set of roles and are used to specify norms that hold for all roles in the group. Groups are defined by means of an identifier, a non-empty set of roles, and group norms. An example of a group

in the conference scenario is the organizing team, consisting of the roles *program chair*, *local organizer*, and *general chair*.

The distribution of objectives in roles is defined by means of the *role hierarchy*. Different criteria can guide the definition of *role hierarchy*. In particular, a role can be refined by decomposing it into sub-roles that, together, fulfill the objectives of the given role.

This refinement of roles defines *role dependencies*. A dependency graph represents the dependency relations between roles. Nodes in the graph are roles in the society. Arcs are labeled with the objectives for which the parent role depends on the child role. Part of the dependency graph for the conference society is displayed in Figure 2.4.

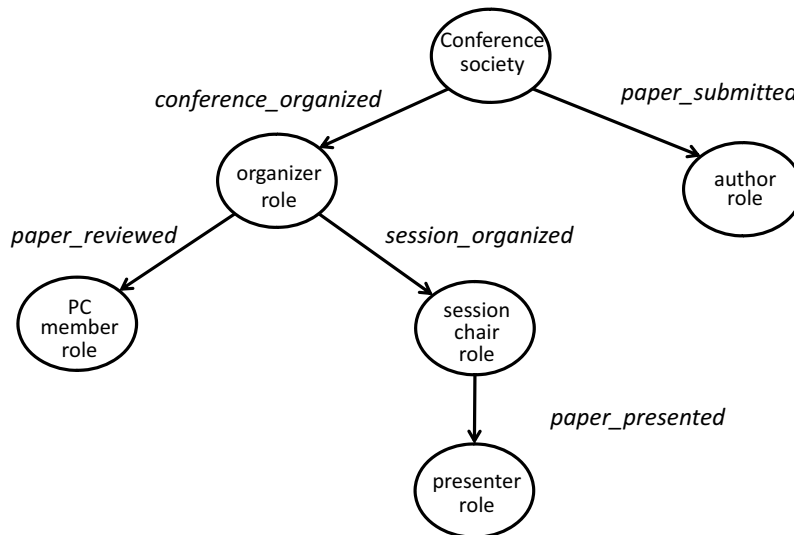


Figure 2.4: Role dependencies in a conference.

For example, the arc between nodes PC-Chair and PC-member represents the dependency between *PC-Chair* and *PC-member* concerning *paper-reviewed* ($PC_Chair \succeq_{paper_reviewed} PC_Member$). The way objective g of r_1 in a dependency relation $r_1 \succeq_g r_2$ is actually passed to r_2 depends on the coordination type of the society, defined in the architectural templates. In OperA, three types of role dependencies are identified: *bidding*, *request*, and *delegation*. These dependency types result in three different interaction possibilities:

Bidding defines market, or auction-like interactions, where the dependent (initiator) of the dependency asks for proposals from the dependees. Typically, the best proposal is selected for the achievement of the objective.

<i>Scene</i>	Review Process
<i>Roles</i>	Program-Chair (1), PC-member(2)
<i>Results</i>	$r_1 = \forall P \in \text{Papers: reviews_done}(P, \text{rev1}, \text{rev2})$
<i>Interaction Pattern</i>	PATTERN(r_1): <i>see Figure 2.5</i>
<i>Norms & Rules</i>	Program-Chair PERMITTED assign_papers IF paper_assigned THEN PC_member OBLIGED review_paper BEFORE deadline

Table 2.3: Script for the *review process* scene.

Request leads to networks, where roles interact cooperatively toward the achievement of an objective;

Delegation gives rise to hierarchies, where the dependent of the dependency delegates the responsibility of the achievement of the objective to the dependees (i.e., subordinates).

3.2.2 The Interaction Structure

Interaction is structured as a set of meaningful scenes that follow predefined scene scripts. Examples of scenes are the “registration” of participants in a conference, which involves a registrar and a participant, or “paper review” involving program committee members and the PC chair. A *scene script* describes the players (roles), desired results, and the norms regulating the interaction. The results of an interaction scene are achieved by the joint activity of the participating roles, through the realization of (sub-)objectives of those roles. A scene script establishes also the desired *interaction patterns* between roles, that is, a desired combination of the (sub-)objectives of the roles. Table 2.3 gives an example of a scene script for the review process involving two PC members and a PC chair.

Interaction scene descriptions are declarative, indicating the global aims of the interaction rather than describing exact activities in details. Interaction patterns can be more or less restrictive, which will give the agent enacting the role more or less freedom to decide how to achieve the role objectives and interpret its norms. Following the ideas of [42, 66], we call such expressions *landmarks*, defined as conjunctions of logical expressions that are true in a state. Landmarks combined with a partial ordering to indicate the order in which the landmarks are to be achieved are called a *landmark pattern*. Figure 2.5 shows the landmark pattern for the *review process*. Several different specific actions can bring about the same state, that is, landmark patterns actually represent families of actual interaction protocols. The use of landmarks to describe activity enables the actors to choose the best applicable actions, according to their own goals and capabilities. The ordering relation between scenes is given in the *interaction structure* (see Fig-

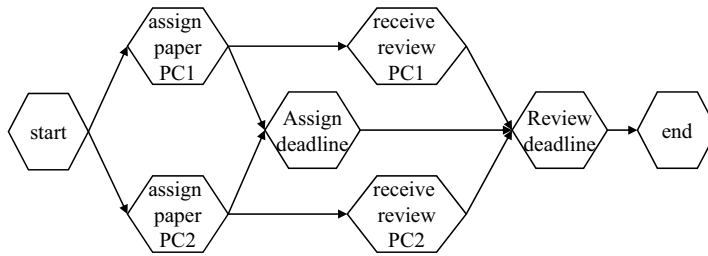


Figure 2.5: Landmark pattern for *review process*.

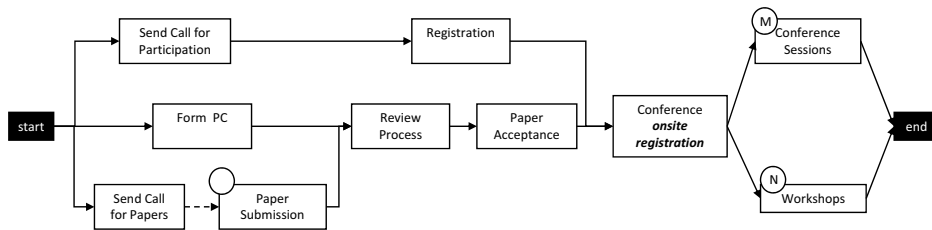


Figure 2.6: Interaction structure in the conference scenario.

ure 2.6). In this diagram, *transitions* describe a partial ordering of the scenes, plus eventual synchronization constraints. Note that, at run-time, several scenes can be happening at the same time and one agent can participate in different scenes simultaneously. Transitions also describe the conditions for the creation of a new instance of the scene, and specify the maximum number of scene instances that are allowed simultaneously. Furthermore, the enactment of a role in a scene may have consequences in following scenes. Role *evolution relations* describe the constraints that hold for the role-enacting agents as they move from scene to scene; for example, in the transition between paper acceptance and conference registration, authors will become participants.

3.2.3 The Normative Structure

At the highest level of abstraction, norms are the *values* of a society, in the sense that they define the concepts that are used to determine the value or utility of situations. For the conference organization scenario, the desire to share information and uphold scientific quality can be seen as values. However, values do not specify *how*, *when*, or in *which* conditions individuals should behave appropriately in any given social setup. In OperA, these aspects are defined in the normative structure.

In OperA, norms are specified using a deontic logic that is temporal, relativized (in terms of roles and groups), and conditional [23]. For instance, the following norm might hold: “*The authors should submit their contributions before the submission deadline*” – which can be formalized as, for example, $O_{author}(submit(paper) \leq Submission_deadline)$.

Furthermore, in order to check norms and act on possible violations of the norms by the agents within an organization, abstract norms have to be translated into actions and concepts that can be handled within such organizations. To do so, the definition of the abstract norms are iteratively concretized into more concrete norms, and then translated into specific rules, violations, and sanctions. Concrete norms are related to abstract norms through a mapping function, based on the “counts-as” operator as developed in [2]. Norms are further discussed in Section 4.

3.2.4 The Communication Structure

Communication mechanisms include both the representation of domain knowledge (*what* we are talking about) and protocols for communication (*how* we are talking). Both content and protocol have different meanings at the different levels of abstraction. For example, while at the abstract level one might talk of *disseminate knowledge*, such action will probably not be available to agents acting at the implementation level, where such abstract objective will be translated into concrete actions, such as *publish proceedings*. Specification of communication content is usually realized using ontologies, which are shared conceptualizations of the terms and predicates in a domain. Agent communication languages (ACLs) are the usual means in MAS to describe communicative actions. ACLs are wrapper languages in the sense that they abstract from the content of communication.

In OperA, the communication structure describes both the content and the language for communication. The content aspects of communication, or domain knowledge, are specified by *domain ontologies*; and *communication acts* define the language for communication, including the performatives and the protocols.

4 Institutions

The concept of institution – and its realization in multiagent systems – has largely been inspired by the economic [49] and social [36, 51] perspectives, as discussed in Section 2.3.2. In contrast, the concept of organization has drawn more on the literature of organizational structure, management, and business [47]. While the terminology, emphasis, and tools are different, there is substantial overlap in goals – indeed institutions can be seen to underpin organizations, and it is these connections that we aim to illustrate in this section.

Institutions are closely tied to the concept of norm – in both its implicit, social manifestation and its explicit, legal form – and reiterating North [49], they constitute “the rules of the game in a society, or more formally, the humanly devised constraints that shape social interaction.” Harré and Secord [36] emphasize the importance of roles in institutions as “that part of the act-action structure produced by the subset of the rules followed by a *particular category of individual*,” and state that “Role is a *normative* concept, focusing on what is proper for a person in a particular category to do.” These views are also echoed by Ostrom [52], who describes institutions as “the prescriptions that humans use to organize all forms of repetitive and structured interaction ... at all scales” and, just as importantly, observes that individuals “face choices regarding the actions and strategies they take, leading to consequences for themselves and for others.” All of which are underlined by the dictionary [53] definitions:

INSTITUTION An established law, custom, usage, practice, organization, or other element in the political or social life of a people; a regulative principle or convention subservient to the needs of an organized community or the general ends of civilization.

NORM A standard or pattern of social behavior that is accepted in or expected of a group.

This has led to the view in multiagent systems research [12, 54, 59, 74] that an institution is a set of norms, while still encompassing the rich variety of usage surrounding the term “norm.”

In this section, we first address the relationship between institutions and organizations, then examine how individual actions affect institutional state through the concept of conventional generation. Consequently it becomes possible to recognize particular configurations of the institution, in order to monitor both individual and collective action, including norm violation and subsequent application of sanction. We conclude with an illustration of one approach to normative modeling, taking the conference scenario, using the institutional action language InstAL.

4.1 Organizations, Institutions, and Norms

Institutions facilitate and enforce the normative character of organizations by describing exchange mechanisms, specifying coordination structures, determining interaction and communication forms within the organization, connecting organizational and individual perspectives, and, above all, making explicit the social norms governing behavior, *external* to the agents.

The relationship between organizations, institutions, and norms is captured in Figure 2.7, where the major cycle connects norms, institutions, and organizations. We explain the connections starting from norms:

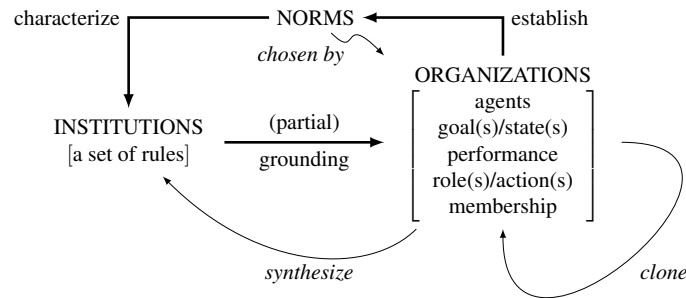


Figure 2.7: Relating institutions, norms, and organizations.

- A particular set of norms characterize an institution, giving it a kind of signature, which enables us to recognize a university, a bank, a club, or a company, for example. Those norms also provide the agent with a representation of the “rules of the game” – how the agent processes those rules may vary widely, as covered in Section 5.
- An organization is typically constructed from a recognized set of norms – an institution – and fixes aspects of them to meet its organizational goals, taking account of the attributes listed in the figure, such as performance, roles, and membership. Agents then animate the organization, playing roles and taking actions to achieve individual and organizational goals.
- The agents participating in an organization are the enablers of change, and may choose to refine, revise, replace, or even create norms in response to changes in requirements, changes in the environment in which the organization is situated, or simply changes in the society of agents that the organization serves. Hence we get the establishment of new norms, contributing to the pool by which institutions and organizations are defined. Change can arise bottom-up, initiated by the actors in an organization, where through the application of recognized (institutional) decision-making procedures the norms and organizational properties can be amended: this typifies incremental change. It can also be imposed, possibly even externally, requiring the wholesale revision of the normative and organizational structure. These aspects are discussed further in Section 6.

While the above explains the major relationships between the three elements, there are also other feedback and evolutionary processes:

- Organizations – through the actions of their participants – have some control over which rules they choose for which game from the set of norms, even to the extent of creating disruptive organizational models that act as “game changers.”
- Consequently, successful (new) organizational models may be cloned and

refined, and in due course may lead to the synthesis of new institutions.

In practice, an agent is likely to be subject to the governance of more than one institution, certainly concurrently, perhaps even simultaneously – consider for example a job submitted from one country to a cloud computing facility in another and the question of which legislation governs the security of which stages of the computation. It would be surprising, given such a scenario, if the norms of one institution do not sometimes conflict with another: in the worst case, an agent may take a norm-compliant action in one institution, only to violate a norm in another, or vice versa, so that whichever action the agent takes, it may suffer sanctions. This may seem a pathological case, but neither can it be ruled out: given that institutions may be developed independently of one another, without knowing how they may be combined in the future, such conflicts are inevitable and is the reason why agents need to be able to find out the potential consequences of actions before taking them.

4.2 Events and States

Approaches to the specification of organizational processes fall broadly into one of two categories: state-based or event-based. The two approaches offer alternative views of the behavior of a system, which can be abstractly characterized by a trace comprising events *and* states, where an event causes the transition from one (system) state to the next. Neither events nor states are sufficient alone: a sequence of events does not inform us about the system state, whereas a sequence of states does not inform us how the system got where it is.

In pursuit of the objective expressed in “On social laws for artificial agent societies [...]” identified in [62] as “[...] laws which guarantee the successful co-existence of multiple programs and programmers,” the set of norms comprising an institution ought to be:

- capable of describing correct as well as incorrect action,
- obligations acquired through correct action, and
- sanctions applied for incorrect action

while maintaining a record of the institutional state.

This conceptual statement of requirements is captured in the upper part of Figure 2.8: there is a (partially) observable environment, where the actions of agents are events (e_i) that bring about changes in the environmental state, but additionally *some* events (e_i) are recognized and mapped to institutional events (e'_j), which bring about changes to the institutional state, causing the addition and deletion of institutional facts. Hence, we can write down a simple abstract model of institutions (lower part of Figure 2.8), which highlights the trace that characterizes the interaction of a single actor with an institution – the *institutionally situated*

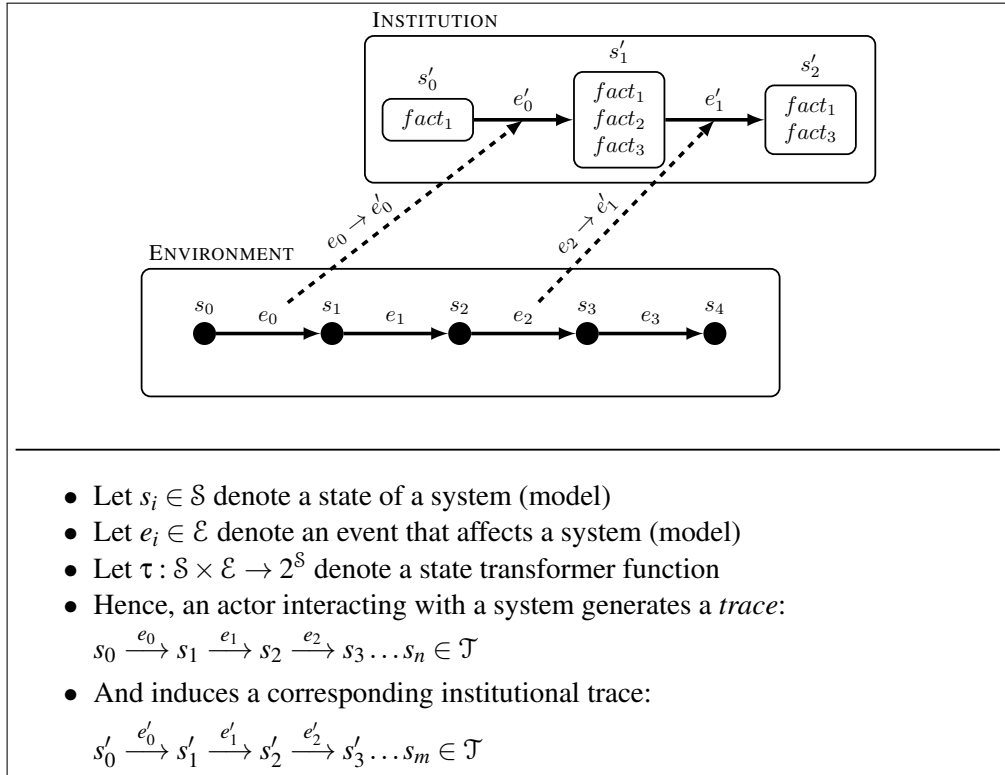


Figure 2.8: An event-based view of institutions.

agent – and the (institutional) state transformer function that takes a set of institutional facts and an event and produces the consequent institutional state. If the event is of no relevance for the institution, the institutional state does not change. If it is relevant, the state does change. For example, the program chair may only close submission if it was previously opened. The notion of interpreting an action in the physical world as an institutional action is called *conventional generation*, deriving from speech acts [60], action theory [32], and the unification of these ideas [18], as reflected in the upper part of Figure 2.8.

The attraction of an event-based approach is that it offers a fine-grained view of the action, enabling a (relatively) forensic focus on details and ordering that may be critical to system state evolution. In contrast, the attraction of a situation-based approach is that it abstracts away from the detail of individual events and orderings (that may be irrelevant), enabling a focus on significant phases in the evolution of the system state. Both are feasible within a common formal and computational framework, as described in [43].

The point of such a formalization, from an organizational modeling perspec-

tive, is to support the construction of computational models of dynamic systems, so that it becomes possible to ask (of a model) not only “where are we?” but also “how did we get here?” – along with the subsidiary questions of “where can we go?” and “how do we get there?” With this formal context in place, and corresponding computational mechanisms, it becomes possible to make the connection between the formalization of events and states and how multiagent systems may use such models.

4.3 Obligations, Permission, and Power

In this section we examine (institutional) constraints on action and the expression of those constraints in the form of: (i) obligations, which denote an obligation on some principal to bring about a particular (institutional) state of affairs, after which the obligation is discharged (ii) permissions, which denote whether an action is correct for some principal in some state of the institution, and (iii) powers, following [40], which denote whether an action by some principal “counts-as” some institutional action, hence bringing about a new institutional state. To put this in terms of Figure 2.8, attention is focused on what happens on the dashed lines between environment and institution.

Obligations are the consequence of some action, but obligations are institutional, not environmental artifacts, and so first the action must be recognized as relevant to the institution ($e_i \rightarrow e'_j$) in the current state, and only then, if the action is permitted, might an obligation be added to the institutional facts. A subsequent action (mapped to an institutional event) may then bring about a state of affairs that satisfies the obligation, so that it may be removed from the institutional state. In the case of the conference scenario, a reviewer acquires the obligation to deliver a review, when he or she is assigned an article. The obligation is discharged when the review is submitted. The obligation may also be contingent on another event, such as when the program chair says reviews are due.

Permissions are largely self-explanatory. If an action is not permitted for some principal in some state, then the action is in violation of the institution. Thus, a permission is also an institutional artifact. Sometimes violations matter, such as the program chair closing submission before the announced deadline; other times they can be ignored, if the effect is benign.

Power is an important property in modeling, because it determines whether an action has any institutional effect. To be more precise, it determines whether an *institutional* action has an institutional effect, because although power can be granted and rescinded with respect to an institutional action, such operations are meaningless for environmental actions. Thus, as with obligations, it is first necessary to recognize whether the action is relevant in the current state, then determine

if the institutional action is permitted and empowered for that principal in the current state. Power is represented in the institutional state, associating an action and some principal, and may be added and removed consequent to institutional actions. In the conference scenario, the program chair is empowered to open and close submission, for example, whereas a program committee member is not. But the empowerment to close submission may only be added after submission is opened.

4.4 Example of Institutional Modeling: InstAL

We start this section by giving a brief introduction to the formal model (see Figure 2.9) behind InstAL, explaining how it relates to the principles of institutions covered in Sections 4.2 and 4.3. This is followed by an overview of the InstAL language, through annotated examples of fragments of the conference scenario model. InstAL is translated into a computational model based on Answer Set Programming [30].

4.4.1 The Formal Model

Data in the formal model comprises events of various kinds (exogenous and institutional) and fluents,¹ also of various kinds (power, permission, obligation), corresponding to the concepts discussed above, from which the institutional state is constituted.

The transformer function comes in two parts: the generation relation, denoted \mathcal{G} , and the consequence relation \mathcal{C} , again reflecting the concepts discussed above and the consequences arising from their interaction. \mathcal{G} is responsible for recognizing relevant exogenous events and turning them into institutional events, but also ensuring that all the institutional events that ensue from an institutional event are generated. \mathcal{C} is responsible for the addition and deletion of fluents in the institutional state, arising from all the events identified by \mathcal{G} . Hence, τ (from Figure 2.8) is the application of \mathcal{C} to the events arising from the transitive closure of \mathcal{G} . For a detailed description of the computational model see [13].

4.4.2 The Conference Scenario

Given a scenario, the task is to translate the natural language description, and other expressions of the requirements, into a specification. The start of the process is very similar to that employed in object-oriented modeling: analyze the description of physical world or *exogenous* events that are of significance to the conference

¹A term whose presence in the institutional state indicates it is true, and absence implies falsity.

Normative system	$\mathcal{N} := \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$
Events, comprising exogenous, (normative) actions and (normative) violations	$\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{inst}$ with $\mathcal{E}_{inst} = \mathcal{E}_{act} \cup \mathcal{E}_{viol}$
Normative facts (fluents): power, permission, obligations, and domain-specific facts	$\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$
Generation relation: maps state and event to a set of events	$\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$
State formula: the set of positive and negative fluents comprising the current normative state	$\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$
Consequence relation: maps state and event to a pair (additions, deletions) of sets of fluents	$\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ where $C(X, e) = (\mathcal{C}^\uparrow(\varphi, e), \mathcal{C}^\downarrow(\varphi, e))$ where (ii) $\mathcal{C}^\uparrow(\varphi, e)$ initiates a fluent (ii) $\mathcal{C}^\downarrow(\varphi, e)$ terminates a fluent
The initial set of fluents	Δ

Figure 2.9: An event-based formal model of institutions.

institution and subsequently to institutional events. For example, there is the opening of submission, the closing, the start of reviewing, the end of reviewing, and so forth. For the sake of the example, we assume the following declarations:

- Person: frank, gerhard, julian, virginia
- Paper: paper01
- Review: review02

where Person, Paper, and Review are types, as shown in Figure 2.10.

All of the physical world events bring about institutional events and begin to lay the foundations of the model. Once such a set of events has been identified, two issues become apparent: (i) that order matters: opening comes before closing, submission comes before review, etc., and (ii) that the identity of actors matters: actors play roles and only certain roles should cause certain events. For example, only the conference chair can declare that the conference is open for submissions; and likewise, paper assignments can only be made by the chair, and then only subject to certain conditions, such as the reviewer not being a listed author of the paper. Institutions make explicit this separation of concerns and enable reasoning about such matters. The keys to dealing with the issues described above are the twin concepts of permission and power. Physical world events are always empowered, but within the institution, power can be given and taken away in order

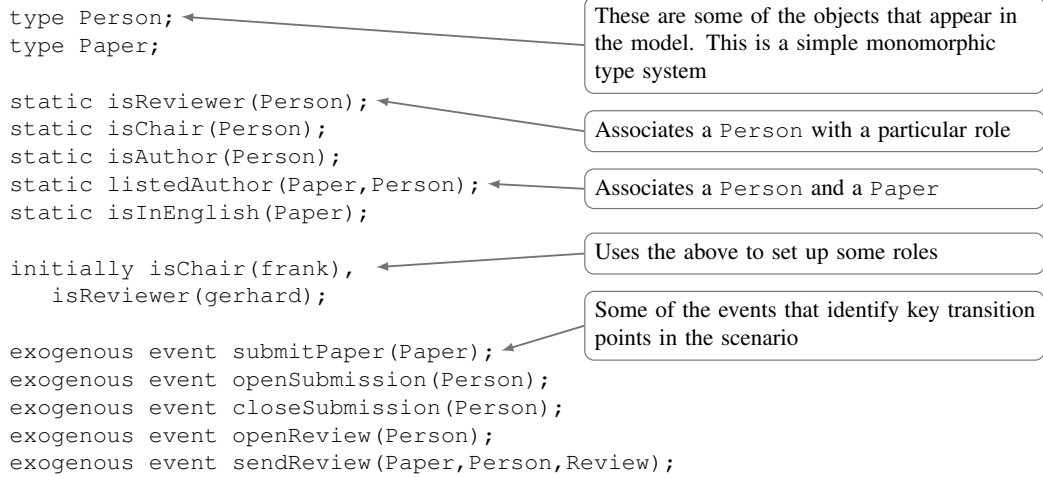


Figure 2.10: Conference types, predicates, and events.

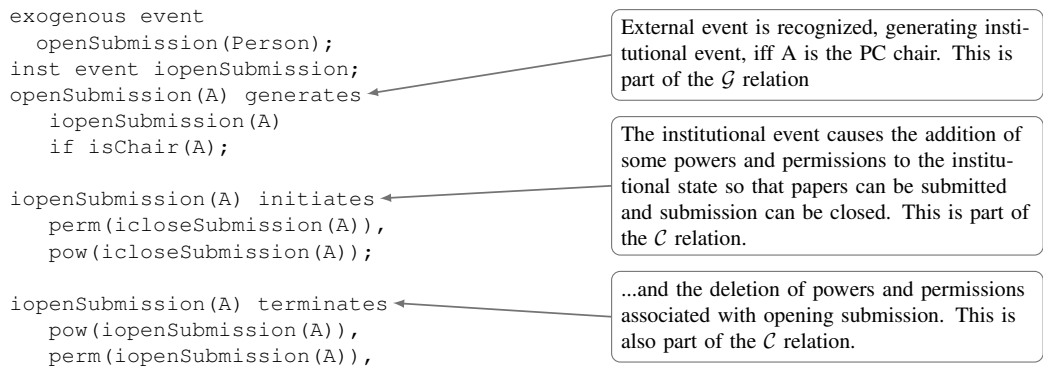


Figure 2.11: Institutional events changing institutional state.

to ensure that events have (or not) their intended institutional consequences; thus, for example, closing submission might only be empowered after the opening of submission. The PC chair can close submission because he or she has the permission, but the PC chair only has the power if sufficient time has elapsed since the opening, and he or she may only do it (say) once, as shown by the fragment in Figure 2.11.

Obligations are used to express that certain events have to take place, e.g., the review requires that a review is delivered before the review period closes. To emphasize that the evolution of the model state is controlled by external events, no dates and the like are encoded in the model. Instead we use exogenous events that act as deadlines, i.e., the obligation has to be satisfied before the deadline occurs. This deadline event can be generated by an agent acting as a timekeeper, as above, where the program chair declares the review period closed, so any reviews not sent

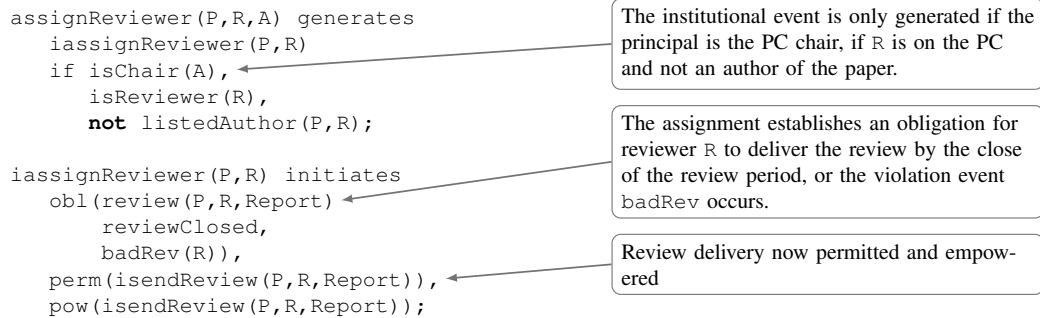


Figure 2.12: Reviewer assignment.

trigger badRev violations (see Figure 2.12).

Dedicated violation events can be introduced to indicate not only that an event has taken place without permission but also that an obligation was not fulfilled or that an undesirable event has taken place, given the current state of affairs. The domain fluents are used to describe the non-institutional state of the world. They keep track of what has happened in the system, such as a paper was submitted or a reviewer was assigned, and they record details about a participant.

An InstAL institutional specification can be used in two ways: (i) to explore the *static* properties of a set of normative rules, such as reachability of certain states or the existence of traces containing particular events, using synthetic sequence of external events, and (ii) to explore the *emergent* properties of those rules, in conjunction with an external source of events, which could be derived from the physical world, or a simulation, or some combination.

The static properties are expressed through the generation of answer sets corresponding to all the possible traces that could occur, subject to the ordering constraints determined by external events. An impression of how this can be visualized appears in Figure 2.13. The time instants are denoted $i_{j-1} \dots i_{m+1}$, corresponding not to real time, but to the occurrence of events of significance to the institutional model. The institutional state corresponding to a time instant appears below it, with additions arising from the last exogenous event in bold typeface. The arc between each instant is labeled with the exogenous event (at the top) that caused the transition, followed by any consequent events, as determined by the transitive closure of \mathcal{G} . The fragment here illustrates the registration of a paper, its submission, close of submission, reviewer assignment, and review submission.

The emergent properties of an institution are best explored initially by coupling the institutional model with a simulation, and most likely an agent-based simulation, since this permits experimentation with diverse individual behaviors through a range of population mixes and subsequent statistical analysis for the significance of effects observed. For a detailed description of the process of ex-

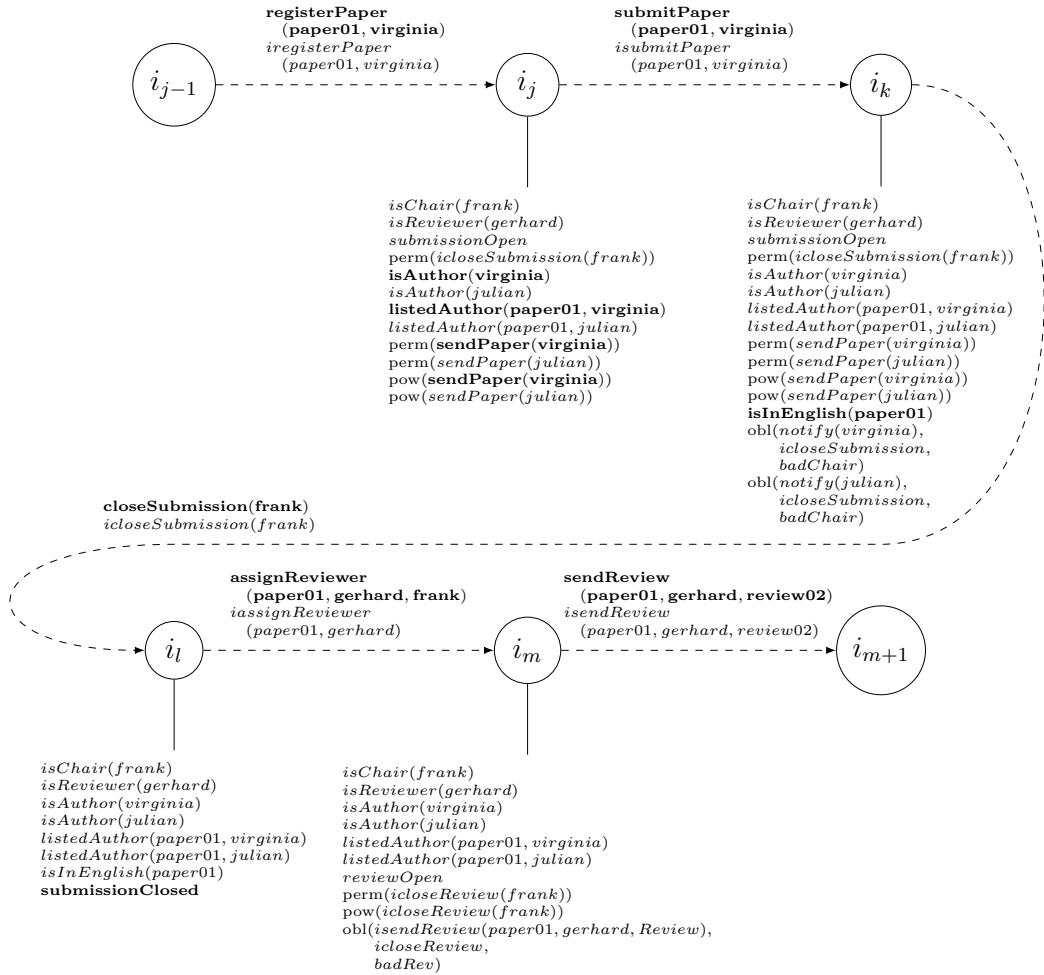


Figure 2.13: A visualization of a possible answer set trace.

amination of emergent properties and a fully-worked case study, see [6].

5 Agents in Organizations

An important challenge in agent organizations is the specification of mechanisms through which agents can evaluate the characteristics and objectives of organizational roles, in order to decide about participation. In particular, an agent has to reason about whether it wants to play a role and whether it has the capabilities to behave as the role requires [4]. This is a complex issue and an open area of research. To counter this problem, in many situations, agents are designed from scratch so that their behavior complies with that of the organization. In

such systems, the organizational model is often implicit in the specification of the agents. However, comprehensive approaches to organizations cannot assume that all agents are known at design time, but require organization-aware agents, that is, agents that are able to reason about their own objectives and desires and thus decide and negotiate their participation in an organization [71].

By specifying the way interactions can occur in an environment, multiagent organizations are able to specify global organizational goals independently of the design of the agents (cf. Section 2.4). A role description, as provided by an organization model, identifies a “position” to be filled by a player [50], which contributes to some part of the organizational objectives and interaction rules. By considering role descriptions, global goals can be verified independently of the agents that will act in the system. From the perspective of the organization it often does not matter whether agent A or agent B takes a role, as long as they both have sufficient capabilities. However, the ways in which each agent, A or B, will enact the role will probably differ, leading to different global results. This is because agents are assumed to have their own goals, which may be different from those of the organization, and will use their own reasoning capabilities to decide on the enactment of one or another organizational role, and to determine which protocol available to them is the most appropriate to achieve the objectives of the organizational positions assigned to them. The ability to dynamically bind different players to roles gives the organization a degree of adaptability in meeting changing goals and environments [57].

Existing approaches for programming role enactment focus mainly on the process of role enactment through communication with the organization [5], and on the result of role enactment (for example, the adoption of the objectives of a role as agents’ own goals) [16, 17]. In [16], compatibility between agent goals and those of the role is investigated and taken as a prerequisite for enacting the role. Moreover, these approaches assume that (1) organizational specification is explicit and available to the agents, and (2) an agent is able to interpret that specification and reason about whether it has the required capabilities to play a role in order to decide on participating. However, given the heterogeneity of agents in open environments, such level of organization-awareness cannot be assumed for all agents.

Role-enacting agents must be able to perform the assigned role(s). These capabilities include [14]:

- the execution of the functions defined by the role or imposed by role relationships, including the ability to use resources available to the role.
- the ability to communicate, as a proxy for its role, with players of other roles.
- the ability to reason about which of its plans and activities can be used to achieve role objectives.

A possible way to deal with these issues, as proposed in [72], is to equip agents with an interface to the organization (called a governor). This interface prevents any action not allowed by the role definition and therefore ensures organizational norms are met. However, it is not flexible enough to incorporate different agents, enacting styles, capabilities, and requirements. It actually makes the actual agent “invisible” to the society and only its enactment of the role behavior is apparent in the organization. Moreover, the interface determines exactly which actions are allowed, while differences between agents are invisible to the organization.

From the perspective of an agent, the role description provides a more or less abstract definition of the organizational knowledge and skills required to perform the role adequately. Depending on the level of detail of an organization specification, more or less interpretation is required from agents. Detailed organization models support agent designers to develop agents whose behavior complies with the behavior described by the role(s) they will take up in the society. However, such a solution is not applicable to open systems, where it is assumed that heterogeneous agents are designed to run independently of the organization.

From the perspective of the organization, the concerns are the effect of the attitudes of agents toward the performance of roles. Agent literature discusses extensively different types of social attitudes of agents: selfish, altruistic, honest, dishonest, etc. [11, 21, 45, 48, 64]. Different agents result in different role performances, because the way an agent will plan its goals, which is dependent on its social attitude, influences the realization of its role objectives and the fulfillment of the role norms. For instance, some agents will only attempt to achieve the goals of their adopted roles and forget their own private goals, while others will only attempt to achieve the goals of the role after all their own goals have been satisfied. Furthermore, the relations between agent plans and role objectives, and of agent goals and role sub-objectives must be considered, as well as the influence of the role norms on the behavior of agents.

The participation of agents in an organization assumes that there is some benefit to be gained, preferably both by the agent and by the organization. Depending on how the agent will enact its role(s), different behaviours can be distinguished [16]. Agents that follow a social enactment strategy will attempt first to realize their organizational objectives (obtained from the roles they enact) before it will consider its own goals. In selfish enactment strategies, the situation is reversed. Many other situations are also possible. In the same way, the effect of agent plans on role objectives, and of role objectives on agent goals leads to different types of role enactment strategies [64], in which either the role or the individual plans can be enriched. Organizational norms also affect the behavior of agents in the organization, as those can limit or alter the achievement of individual goals.

In summary, most agent organization models assume that the effective engi-

neering of MAS must be based on the independence between organizational models specifying collective structures and agent architectures specifying individuals. However, in order to allow agents to join organizations, one must be able to specify what is expected of those agents, and engage in a process of admission during which the requirements and aims of both the organizations and the agent are evaluated. Assuming that agents are capable of reasoning about role enactment, we discussed what the consequences are for the organization of different enactment styles.

6 Evolution of Organizations

One of the main reasons for having organizations is to achieve stability. However, organizations and their environments are never static. They change, disappear, or grow. Agents can migrate, organizational objectives can change, or the environment can evolve, all of which require adaptation of organizations. Reorganization is the response to two different stimuli: a reaction to (local) changes in the environment, and a means to implement modified overall intentions or strategies.

Multiagent organization models must therefore not only enable the adaptation of individual agents, but also be able to adapt organizations' structures dynamically in response to changes in the environment. Depending on the type of organization and on the perceived impact of the changes in the environment, adaptation can be achieved by behavioral changes at the agent level, modification of interaction agreements, or the adoption of a new social structure.

Organizational evolution is the process by which organizations change and adapt over time to meet new requirements and changes in the deployment environment. There is a growing recognition that in organizations, a combination of regulation and autonomy is often necessary. Most human organizations follow ordered structures and stated goals, but develop an informal structure that reflects the spontaneous efforts of individuals and subgroups to control and adapt to the environment. The autonomy to develop informal structures is indispensable to the process of organizational control and stability [61]. Also in the area of MAS, Wellman noted in 1993 that *"combining individual rationality with laws of social interaction provides perhaps the most natural approach to [...] distributed computations"* [77]. However, although each change may itself be justified, often, emergent patterns became the norm, and with time, part of the structures and rules fixed in the organization. Figure 2.14 shows this cycle, common in human interactions, moving from explicit rules to implicit practical structures, and back.

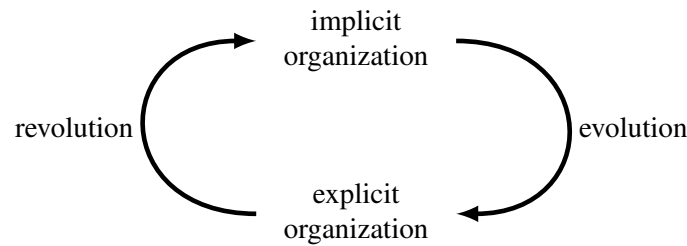


Figure 2.14: Implicit and explicit organizations.

6.1 Organizational Adaptation

In management and organization theory, most studies aim at understanding and manipulating the performance, or behavior, of an organization in its environment. Performance of the organization can be seen as the measure by which its objectives are achieved at a certain moment. Because environments evolve, performance will change. Many organizational studies are therefore concerned with the evaluation of performance, identifying triggers for change and determining the influence of environmental change on the organizational performance, and indicating directions to improve performance.

Reorganization is in a sense at odds with a main objective of creating organizations, that of achieving stability. The issue is then to ascertain under which conditions it is better to reorganize, knowing that organizational stability will be (momentarily) diminished, and when to maintain stability [22]. In order to answer this question, it is necessary to measure the utility of an organization. Reorganization is desirable if it leads to increased utility of the system. That is, the reorganized organization should perform better in some sense than the original situation. In general terms, organizational utility can be defined as the measure in which organizational objectives are met. Moreover, organizational utility depends also on the *cost* of a possible reorganization. That is, any function to measure organization utility must take into account both the success of a given structure, and the cost of any change needed to achieve that structure from the current situation [31].

So and Durfee state that task-environmental factors, together with the structure and behavior of the organization, influence the performance of the organization [67]. That is, the way agents, tasks, and decision-making capabilities are organized influences the performance of the system. In dynamic environments, it is therefore necessary to enable an organization to adapt to change in the task-environment. Organization theory research has also shown that when an organization is able to adapt its structure to environmental changes, its performance will improve [10]. Formal models for organizations should therefore be able to

represent and analyze organizational change.

The concept of *dynamic adaptation* refers to modifications in the structure and behavior of a system, such as adding, removing, or substituting components, done while the system is running and without bringing it down [69]. Dynamic adaptation demands that systems can evaluate their own “health” (i.e., success and other utility parameters) and take action to preserve or recover it by performing suitable integration and reconfiguration actions. Reorganization of organizations should therefore describe both situations in which the operational behavior of the organization changes, due to admission or departure of agents, as well as situations in which the social structure of the society changes, that is, roles, relationships, norms, or interactions change.

6.2 Emergent Organizations

Much research on multiagent organizations takes a design perspective in the sense that an external entity or a small group of decision makers within a larger community determines the nature, structure, and objectives of the organization based on requirements from a set of stakeholders. In contrast, bottom-up, or self-organization, which is suitable in situations where there is a complex interaction between resources and stakeholders, tends to lead to quite different – and novel – solutions.

The kind of scenarios that have received the most attention as part of the study of bottom-up organizations are those that feature so-called common-pool resources (CPRs), often known as the “tragedy of the commons” [35]. Several illustrative cases, and the innovative solutions people have developed, are analyzed by Ostrom [51] and lead to the topic of this section, the institutional analysis and development framework (IAD) [52]. In contrast to conventional approaches to the analysis of agent interaction, which tend to focus on game-theoretic models or utility-driven decision making in an isolated context, IAD embraces the complexity of real-world situations in which there are both many variables and contexts within contexts. The starting point for IAD is the *action arena*, which comprises the participants and a particular action situation. Thus an arena identifies some physical or conceptual space that situates the action, while an episode for analysis is populated by certain players and circumstances that are determined by (identified) exogenous variables. The interactions of the players are subject to rules – some of which may be known, others yet to be discovered – leading to outcomes, which may then be evaluated to assess the performance of the system. As Ostrom [52] makes clear, her subjects of study are *human* solutions to *human* organizational problems and the work is firmly rooted in the social sciences. We draw attention to it here for the following reasons: (i) the methodology of IAD provides useful lessons and a starting place when thinking about unconventional

solutions for “messy” common-pool resource-like problems in both pure agent and human-agent systems, (ii) because the solutions emerge from the desires of the participants, rather than being imposed externally, and because they typically exhibit high stability and resilience, it is interesting to see how these ideas may contribute to the largely unexplored subject of emergent normative frameworks for virtual environments, and (iii) the approach emphasizes the “discovery” of opposite solutions that take a range of factors into account and discourages the re-use of off-the-shelf solutions.

7 Conclusions

This chapter gives an organization-oriented perspective on multiagent systems. Assuming MAS to be an organization, or society of agents, makes it possible to describe the set of agents interacting to coordinate their behavior as well as the cooperation requirements that can lead to the achievement of some collective goal [28]. The stance taken in this chapter is that the conceptualization of multiagent systems is better served by an explicit separation of concerns about organizational and individual issues. Whereas the individual aspect should concern the mental states and capabilities of an agent, organizational issues can better describe activities and constraints that characterize the externally observable behavior of a whole agent population. This chapter further presents the two most common perspectives on organization, that of organization as structure (cf. Section 3) and that of organization as institution (cf. Section 4).

An important ongoing research issue is that of *organization-aware agents*, as discussed in Section 5 [70]. Agents who want to enter and play roles in an organization are expected to understand and reason about the organizational specification, if they are to operate effectively and flexibly in the organization. The broader aim of this line of research is the development of languages and techniques for programming organization-aware agents. Such agents should be able to reason about role enactment, about whether they want to play a role and whether they have the capabilities to behave as the role requires.

Another open research issue concerns the interaction between human and artificial agents in organizations. What happens when human and artificial agents interact in organizations? Such cooperation is increasingly happening, mostly in situations where reaction speed is important (such as emergency response), where knowledge is diffuse, where a high level of connectivity is necessary, or where operation in constantly changing environments is needed. As yet, the reach and consequences of coordinated activity between people and artificial agents working in close and continuous interaction is not well understood. Planning technologies for intelligent systems often take an *autonomy-centered* approach, with represen-

tations, mechanisms, and algorithms that have been designed to accept a set of goals, and to generate and execute a complete plan in the most efficient and sound fashion possible. The *teamwork-centered* autonomy approach takes as a premise that people are working in parallel alongside one or more autonomous systems, and hence adopts the stance that the processes of understanding, problem solving, and task execution are necessarily incremental, subject to negotiation, and forever tentative [7]. That is, autonomy in teams requires close alignment to the current work of other team members and the perception of the team's goals.

Another major open issue is that of organizational adaptation and evolution. Reorganization is needed in order to enable systems to enforce or adapt to changes in the environment. This issue has been discussed by many researchers in both organizational theory and distributed systems, resulting mostly in domain-oriented empiric solutions. The lack, in most cases, of a formal basis makes it difficult to develop theories about reorganization, prevents the comparison of approaches and results, and makes it difficult to adapt models to other domains or situations.

The view of agent organizations presented in this chapter posits that agent organizations demand (i) the integration of organizational and individual perspectives, (ii) the dynamic adaptation of models to organizational and environmental changes, and (iii) rely significantly on the notions of openness and heterogeneity in MAS. Practical applications of agents to organizational modeling are being widely developed but formal theories are needed to describe interaction and organizational structure. Furthermore, it is necessary to get a closer look at the applicability of insights and theories from organization sciences to the development of agent organizations. There is a need for a theoretic model to describe organizations and their environments that enables the formal analysis of the fit between organizational design and environment characteristics. This enables the a priori comparison of designs and their consequences and therefore supports the decision-making process on the choice of design.

Acknowledgments

We are grateful to Huib Aldewereld, Tina Balke, Frank Dignum, and Marina De Vos for their assistance in writing this chapter.

8 Exercises

1. **Level I** You have been asked to design an organization model for an on-line bookstore. The system must be able to handle both selling and buying books by individuals as well as acting as front-end for a bookstore. Take

into account the interests and requirements of the different stakeholders.

- (a) What are the organizational roles?
 - (b) What objectives can you identify for each role?
 - (c) What are their dependencies?
 - (d) Design a possible interaction structure for this organization. Are there any scenes that can be active at the same time? Which interaction scenes must follow from others?
 - (e) What are the roles that can participate in each scene?
2. **Level 1** Extend the abstract model of agent, institutions, and environments to handle:
 - (a) Multiple agents.
 - (b) Multiple institutions.
 - (c) Multiple agents and multiple institutions.
3. **Level 2** Design fragments of an institution in InstAL,² or an organization in OperA, to control “rover” agents to explore a simulated planetary environment, in order to locate resources, and then return them to a base. Assume a toroidal planet to avoid edge cases. Initial conditions are that: (a) rover agents start at a base whose location is established at random, (b) each agent has a limited amount of energy, and (c) resources are scattered throughout the environment. Varying degrees of difficulty are possible as set out below, but in each case the objective is to return as many resources to the base as possible. Agents that run out of energy can no longer function. When all agents run out of energy, or all resources have been collected, the scenario is complete. Ordered by difficulty, the scenarios are:
 - (a) A single agent with one resource location. High energy limit.
 - (b) A single agent with several resource locations. High energy limit.
 - (c) Three agents with many available resources. Medium energy limit.
 - (d) Three agents with sparse resources. Medium energy limit.
 - (e) Ten agents with sparse resources. Low energy limit.
4. **Level 2** Develop additional fragments of the InstAL model of the conference scenario to demonstrate some (or all!) of the following additional features, such as:
 - (a) Handle papers without reviews after the end of the review period.
 - (b) Extend the review conflict mechanism to account for colleagues.
 - (c) Extend the review mechanism with reviewer bidding preferences.

²More information and tools can be downloaded from <http://agents.cs.bath.ac.uk/instal>.

- (d) Extend the review mechanism to support multiple reviews of a paper.
 - (e) Handle the processing of decisions: this will require a wider decision scale than just accept/reject.
5. **Level 3** Download and install the OperettA modeling environment available at <http://www.operettatool.nl/>. Implement the solution of the exercise above in this environment.
- (a) Execute the verification process by right clicking the OM element in the tree view and selecting *validate*. (i) Do you find any error? (ii) What do the warnings mean?
 - (b) For the Book Selling scene, define a Book_Selling landmark pattern. (i) What are the roles that can participate in the scene? (ii) How does the scene work? (iii) Can a buyer buy more than one book within the scene? (iv) Can an order be canceled after the payment is done? What happens then?
 - (c) Now modify the organization to include the possibility of selling a book by auction. What extra roles, objectives, dependencies, and scenes are needed? Run the verification process to check the changes.
 - (d) Define norms of behavior for this auction process.
6. **Level 4** Institutional and organizational change is a complicated exercise. There are mechanisms for incremental change, using predefined collective decision-making procedures and argumentation to propose and enact changes to normative frameworks, which can be imported from and formalized from the physical world. But externally imposed change, where there is significant change in norms and structure, to the extent that a previously compliant act may not be non-compliant, is largely unconsidered. A particularly challenging issue is the identification and validation of appropriate transitional arrangements to avoid a sequence of actions that start compliant but end as violations.
7. **Level 4** Game theory and normative frameworks appear incompatible: one encourages utility maximization, assuming such a function can be defined, while the other encourages the satisfaction of individual and collective goals through obligations and prohibitions. The question is, to what extent can these models be unified, or how can one approach be expressed, or even synthesized, from the other, allowing the tools that have been developed for each to be used to analyze the other?
8. **Level 4** An agent may simultaneously be subject to the governance of more than one set of institutional rules, and given that one institution is most likely not designed to take the other into account, this will inevitably lead

to conflict at some point. Three levels of compatibility appear to be possible: (i) compatible, where no actions lead to violations in either institution (ii) partially compatible, where some actions lead to violations and others do not, and (iii) incompatible, where a compliant action in one institution leads to violation in the other or vice versa, or even *and* vice versa. This raises numerous questions, such as: How to analyze institutions for all possible traces with respect to compatibility? Are there traces that make partially compatible institutions compatible in practice? Are there rule changes that can make institutions compatible? How do those changes affect the abstract norms the institutions are designed to maintain?

References

- [1] Huib Aldewereld. *Autonomy vs. Conformity – An Institutional Perspective on Norms and Protocols*. PhD thesis, Universiteit Utrecht, 2007.
- [2] Huib Aldewereld, Sergio Alvarez-Napagao, Frank Dignum, and Javier Vázquez-Salceda. Engineering social reality with inheritance relations. In *Proceedings of the 10th International Workshop on Engineering Societies in the Agents World X*, ESAW '09, pages 116–131, Berlin, Heidelberg, 2009. Springer-Verlag.
- [3] Huib Aldewereld and Virginia Dignum. OperettA: Organization-oriented development environment. In *Languages, Methodologies and Development Tools for Multi-agent Systems (LADS2010@Mallow)*, LNCS 6822, pages 1–19, 2010.
- [4] Huib Aldewereld, Virginia Dignum, Catholijn Jonker, and M. van Riemsdijk. Agreeing on role adoption in open organisations. *KI - Künstliche Intelligenz*, pages 1–9, 2011. 10.1007/s13218-011-0152-5.
- [5] Matteo Baldoni, Guido Boella, Valerio Genovese, Roberto Grenna, and Leendert Torre. How to program organizations and roles in the JADE framework. In *Proceedings of the 6th German Conference on Multiagent System Technologies*, MATES '08, pages 25–36, Berlin, Heidelberg, 2008. Springer-Verlag.
- [6] Tina Balke. *Towards the Governance of Open Distributed Systems: A Case Study in Wireless Mobile Grids*. PhD thesis, University of Bayreuth, November 2011. Available via <http://opus.ub.uni-bayreuth.de/volltexte/2011/929/>. Retrieved 20120109. Also available as ISBN-13: 978-1466420090, published by Createspace.
- [7] Jeffrey M. Bradshaw, Paul J. Feltoovich, Matthew Johnson, Maggie R. Breedy, Larry Bunch, Thomas C. Eskridge, Hyuckchul Jung, James Lott, Andrzej Uszok, and Jurriaan van Diggelen. From tools to teammates: Joint activity in human-agent-robot

- teams. In Masaaki Kurosu, editor, *HCI (10)*, volume 5619 of *Lecture Notes in Computer Science*, pages 935–944. Springer, 2009.
- [8] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8:203–236, 2004.
- [9] R. Burton, G. DeSanctis, and B. Obel. *Organizational Design: A Step-by-Step Approach*. Cambridge University Press, 2006.
- [10] K. Carley. Organizational adaptation. *Annals of Operations Research*, 75:25–47, 1997.
- [11] C. Castelfranchi. Commitments: From individual intensions to groups and organizations. In V. Lesser, editor, *Proc. ICMAS’95*, pages 41–48, 1995.
- [12] Owen Cliffe. *Specifying and Analysing Institutions in Multi-Agent Systems Using Answer Set Programming*. PhD thesis, University of Bath, 2007.
- [13] Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and reasoning about multiple institutions. In *Coin*, volume 4386 of *LNAI*, pages 67–85. Springer Berlin / Heidelberg, 2007.
- [14] A. Colman and J. Han. Roles, players and adaptive organisations. *Applied Ontology: An Interdisciplinary Journal of Ontological Analysis and Conceptual Modeling*, 2(2):105–126, 2007.
- [15] L. Coutinho, J. Sichman, and O. Boissier. Modelling dimensions for agent organizations. In V. Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference, 2009.
- [16] M. Dastani, V. Dignum, and F. Dignum. Role assignment in open agent societies. In *AAMAS-03*. ACM Press, July 2003.
- [17] Mehdi Dastani, M. Birna van Riemsdijk, Joris Hulstijn, Frank Dignum, and J-J. Meyer. Enacting and deacting roles in agent programming. In *Proceedings of the 5th International Workshop on Agent-Oriented Software Engineering (AOSE2004)*, volume LNCS 3382. Springer, 2004.
- [18] Steven Davis. Speech acts and action theory. *Journal of Pragmatics*, 8(4):469 – 487, 1984.
- [19] Francien Dechesne and Virginia Dignum. No smoking here: Compliance differences between deontic and social norms. In *Proceedings of AAMAS 2011*. IFAAMAS.org, 2011.

- [20] F. Dignum. Autonomous agents with norms. *AI & Law*, 7(12):69–79, 1999.
- [21] V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. SIKS Dissertation Series 2004-1. Utrecht University, 2004. PhD Thesis.
- [22] V. Dignum, F. Dignum, and L. Sonenberg. Towards dynamic organization of agent societies. In G. Vouros, editor, *Workshop on Coordination in Emergent Agent Societies, ECAI 2004*, pages 70–78, 2004.
- [23] V. Dignum, J.J. Meyer, F. Dignum, and H. Weigand. Formal specification of interaction in agent societies. In *Formal Approaches to Agent-Based Systems (FAABS)*, volume F2699 of *LNAI*. Springer, 2003.
- [24] Virginia Dignum. Ontology support for agent-based simulation of organizations. *International Journal on Multiagent and Grid Systems*, 6:191–208, April 2010.
- [25] R. Duncan. What is the right organizational structure: Decision tree analysis provides the answer. *Organizational Dynamics*, Winter:59–80, 1979.
- [26] E. Durfee and J. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *Proc. 13th Int. Distributed Artificial Intelligence Workshop*, pages 94–104, 1994.
- [27] Marc Esteva. *Electronic Institutions: From Specification to Development*. PhD thesis, Technical University of Catalonia, 2003.
- [28] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *AOSE: Agent-Oriented Software Engineering IV*, volume 2935 of *LNCS*. Springer-Verlag, 2003.
- [29] J. Galbraith. *Organization Design*. Addison-Wesley, 1977.
- [30] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
- [31] N. Glasser and P. Morignot. The reorganization of societies of autonomous agents. In *MAAMAW*, pages 98–111, 1997.
- [32] A.I. Goldman. *A Theory of Human Action*. Prentice-Hall, 1970.
- [33] Davide Grossi. *Designing Invisible Handcuffs: Formal Investigations in Institutions and Organizations for Multiagent Systems*. PhD thesis, Universiteit Utrecht, 2007.
- [34] Davide Grossi, Frank Dignum, Mehdi Dastani, and Lambèr Royakkers. Foundations of organizational structures in multiagent systems. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 690–697, New York, NY, USA, 2005. ACM.

- [35] G. Hardin. The tragedy of the commons. *Science*, 162:1243–8, 1968.
- [36] R. Harré and P.F. Secord. *The Explanation of Social Behaviour*. Blackwells, 1972. ISBN 0-631-14220-7.
- [37] Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2004.
- [38] J. Hübner, J. Sichman, and O. Boissier. S-moise+: A middleware for developing organised multi-agent systems. In O. Boissier et al., editor, *COIN I*, volume 3913 of *LNAI*, pages 64–78. Springer, 2006.
- [39] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. *Artificial Intelligence*, 117:277–296, 2000.
- [40] Andrew J. I. Jones and Marek Sergot. A formal characterisation of institutionalised power. *Logic Journal of IGPL*, 4(3):427–443, 1996.
- [41] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: Giving the power back to the agents. In *Proceedings of the 2007 International Conference on Coordination, Organizations, Institutions, and Norms in Agent Systems III*, COIN’07, pages 171–186. Springer, 2008.
- [42] S. Kumar, M. Huber, P. Cohen, and D. McGee. Towards a formalism for conversation protocols using joint intention theory. *Computational Intelligence Journal*, 18(2), 2002.
- [43] H.J. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):159–178, 1998. Retrieved 20110728 from <http://www.ep.liu.se/ej/etai/1998/005/>.
- [44] R. Malyankar. A pattern template for intelligent agent systems. In *Agents’99 Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain*, 1999.
- [45] Maria Miceli, Amedo Cesta, and Paola Rizzo. Distributed artificial intelligence from a socio-cognitive standpoint: Looking at reasons for interaction. *AI & Society*, 9:287–320, 1995.
- [46] S. Miles, M. Joy, and M. Luck. Towards a methodology for coordination mechanism selection in open systems. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World III*, LNAI 2577. Springer-Verlag, 2003.
- [47] H. Mintzberg. *Structures in Fives: Designing Effective Organizations*. Prentice Hall, 1993.

- [48] Abdulla M. Mohamed and Michael N. Huhns. Benevolent agents in multiagent systems. *Multi-Agent Systems, International Conference on*, pages 04–19, 2000.
- [49] D.C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.
- [50] J. Odell, M. Nodine, and R. Levy. A metamodel for agents, roles, and groups. In J. Odell, P. Giorgini, and J. Müller, editors, *AOSE IV*, LNCS, forthcoming. Springer, 2005.
- [51] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, Cambridge., 1990.
- [52] Elinor Ostrom. *Understanding Institutional Diversity*. Princeton University Press, 2005.
- [53] Oxford English Dictionary. <http://www.oed.com>, retrieved March 2011.
- [54] P. Noriega. *Agent Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona, 1997.
- [55] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems*. Wiley, 2004.
- [56] L. Penserini, D. Grossi, F. Dignum, V. Dignum, and H. Aldewereld. Evaluating organizational configurations. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2009)*, 2009.
- [57] Loris Penserini, Virginia Dignum, Athanasios Staikopoulos, Huib Aldewereld, and Frank Dignum. Balancing organizational regulation and agent autonomy: An MDE-based approach. In Huib Aldewereld, Virginia Dignum, and Gauthier Picard, editors, *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 197–212. Springer Berlin / Heidelberg, 2009.
- [58] A. Rice, editor. *The Enterprise and Its Environment: A System Theory of Management Organization*. Routledge, 2001.
- [59] J.A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autònoma de Barcelona, 2001.
- [60] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
- [61] P. Selznick. *TVA and the Grass Roots: A Study of Politics and Organization*. University of California Press, 1953.
- [62] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: off-line design. *Artif. Intell.*, 73(1-2):231–252, 1995.

- [63] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, Cambridge, UK, 2009.
- [64] J. Sichman and R. Conte. On personal and role mental attitude: A preliminary dependency-based analysis. In *Advances in AI*, LNAI 1515. Springer, 1998.
- [65] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, USA, 3rd edition, 1996.
- [66] I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory. In *Proc. ICMAS-98*, pages 269–276. IEEE Press, 1998.
- [67] Y. So and E. Durfee. Designing organizations for computational agents. In K. Carley, M.J. Pritula, and L. Gasser, editors, *Simulating Organizations*, pages 47–64, 1998.
- [68] K. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.
- [69] G. Valetto, G. Kaiser, and Gaurav S. Kc. A mobile agent approach to process-based dynamic adaptation of complex software systems. In *8th European Workshop on Software Process Technology*, pages 102–116, 2001.
- [70] M. Birna van Riemsdijk, Koen V. Hindriks, and Catholijn M. Jonker. Programming organization-aware agents: A research agenda. In *Proceedings of the Tenth International Workshop on Engineering Societies in the Agents' World (ESAW'09)*, volume 5881 of *LNAI*, pages 98–112. Springer, 2009.
- [71] M.B. van Riemsdijk, K. Hindriks, and C.M. Jonker. Programming organization-aware agents. In Huib Aldewereld, Virginia Dignum, and Gauthier Picard, editors, *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 98–112. Springer Berlin / Heidelberg, 2009.
- [72] W. Vasconcelos, J. Sabater, C. Sierra, and J. Querol. Skeleton-based agent development for electronic institutions. In *Proceedings of AAMAS-02, First International Conference on Autonomous Agents and Multi-Agent Systems*, pages 696–703. ACM Press, July 2003.
- [73] J. Vázquez-Salceda. *The Role of Norms and Electronic Institutions in Multi-Agent Systems*. Birkhuser Verlag AG, 2004.
- [74] Javier Vázquez-Salceda. *The Role of Norms and Electronic Institutions in Multi-Agent Systems Applied to Complex Domains. The HARMONIA Framework*. PhD thesis, Universitat Politècnica de Catalunya, 2003.
- [75] Javier Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *JAAMAS*, 11(3):307–360, 2005.

- [76] H. Weigand and V. Dignum. I am autonomous, you are autonomous. In M. Nickles, M. Rovatsos, and G. Weiss, editors, *Agents and Computational Autonomy*, volume 2969 of *LNCS*, pages 227–236. Springer, 2004.
- [77] M. Wellman. A market-oriented programming environment and its application to distributed multi-commodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.
- [78] O. E. Williamson. Why law, economics, and organization? *Annual Review of Law and Social Science*, 1:369–396, 2005.
- [79] Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 407–422. Springer Berlin / Heidelberg, 2001.

Part II

Communication

Chapter 3

Agent Communication

Amit K. Chopra and Munindar P. Singh

1 Introduction

Multiagent systems are distributed systems. Engineering a multiagent system means rigorously specifying the communications among the agents by way of interaction protocols. What makes specifying the protocols for agent interaction especially interesting and challenging is that agents are *autonomous* and *heterogeneous* entities. These properties of agents have profound implications on the nature of protocol specifications. As we shall see, protocols for multiagent systems turn out to be fundamentally different from those for other kinds of distributed systems such as computer networks and distributed databases.

We conceptualize all distributed systems in architectural terms – as consisting of components and connectors between the components. The components of the Internet are all nodes with IP addresses. The main connector is the Internet protocol, which routes packets between the nodes. The components of the web are the clients (such as browsers) and servers and the connector is the HTTP protocol. The components in a distributed database are the client databases and the coordinator and a connector is the two-phase commit protocol. We can discern a pattern here: the connectors are nothing but the interaction protocols among the components. Further, we can associate protocols with the application it facilitates. For example, the Internet protocol facilitates routing; HTTP facilitates access to a distributed database of resources; and the two-phase commit protocol facilitates distributed transactions.

The same applies for multiagent systems except that the components are autonomous and heterogeneous agents, and applications are typically higher-level – for example, auctions, banking, shipping, and so on. Each application would have its own set of requirements and therefore we would normally find different protocols for each application. Below, the term *traditional distributed systems* refers to non-multiagent distributed systems such as the Internet, the web, and so on.

The importance of protocols is not lost upon industry. Communities of practice are increasingly interested in specifying standard protocols for their respective domains. RosettaNet [40] (e-business), TWIST [53] (foreign exchange transactions), GDSN [33] (supply chains), and HITSP [34] and HL7 [31] (health care) are just a few examples.

Our objectives in this chapter are to help the reader develop a clear sense of the conceptual underpinnings of agent communication and to help the reader learn to apply the concepts to the extent possible using available software. The chapter is broadly structured according to the following sub-objectives.

Requirements for protocol specifications The inherently open nature of multi-agent systems places certain requirements on protocol specifications. Meeting these requirements is the key to designing good protocols.

Protocol specification approaches There are many diverse approaches for specifying protocols. We evaluate some approaches widely practiced in software engineering and some historically significant ones from artificial intelligence. We also study an approach that is particularly promising.

Directions in agent communication research The last fifteen years have seen some exciting developments in agent communication. However, many practical concerns remain to be addressed. We discuss these briefly.

1.1 Autonomy and Its Implications

Protocols are modular, potentially reusable specifications of interactions between two or more components. The interactions are specified in terms of the *messages* the components exchange. To promote reusability, a protocol is specified abstractly with reference to the *roles* that the interacting components may adopt. A protocol is designed with a certain *application* in mind. An *enactment* refers to an execution of the protocol by the components.

In distributed systems, the chief concern is *how can distributed components work together effectively?* In other words, how can we ensure their *interoperation*? In engineering terms, protocols are the key to interoperation. The idea is that as long as components are individually *conformant*, that is, follow their respective

roles in the protocol, they will be able to work together no matter how they are implemented. Interoperation makes great engineering sense because it means that the components are loosely coupled with each other; that is, we can potentially replace a component by another conformant one and the modified system would continue to function. You would have noticed that web browsers and servers often advertise the versions of the HTTP standard with which they are conformant.

The same concepts and concerns apply to multiagent systems. However, agents are not ordinary components. They are components that are autonomous and heterogeneous. Below, we discuss exactly what we mean by these terms, and how autonomy and heterogeneity naturally lead to requirements for agent interaction protocols that go beyond protocols for traditional distributed systems.

Each agent is an autonomous entity in the sense that it itself is a domain of control: other agents have no direct control over its actions (including its communications). For instance, consider online auctions as they are conducted on websites such as eBay. Sellers, bidders, and auctioneers are all agents, and none of them exercises any control over the others. If an auctioneer had control over bidders, then (if it chose to) it could force any of the bidders to bid any amount by simply invoking the appropriate method. Such a setting would lack any resemblance to real life.

There is a subtle tension between the idea of a protocol and autonomy. With protocols, we seek to somehow constrain the interaction among agents so that they would be interoperable. Autonomy means that the agents are free to interact as they please (more precisely, each agent acts according to the rationale of its principal). From this observation follows our first requirement. *We must design protocols so that they do not overconstrain an agent's interactions.*

In traditional distributed systems, interoperation is achieved via low-level coordination. The protocols there would specify the flow of messages between the participants. In the case of the two-phase commit protocol, the controller coordinates the commit outcome of a distributed transaction. In the first phase, a controller component collects votes from individual databases about whether they are each ready to commit their respective subtransactions. If they unanimously respond positively, the controller, in the second phase, instructs each to commit its respective subtransaction; otherwise, it instructs each to abort its subtransaction.

The above discussion of autonomy implies the following.

The irrelevance of intelligence Contrast the notion of agent autonomy discussed above with the one where autonomy is interpreted as the ability of an agent to perform high-level reasoning (intelligent agents) or as the degree to which an agent can operate without the supervision of its principal (autonomic agents). Consider that you want to automate your purchases on the web. On the one hand, you can design a simple bidding agent that takes input

from you about the things you want, the maximum prices you are willing to pay, and the reputation thresholds of the sellers and auctioneers you are willing to deal with. On the other hand, you can design a sophisticated bidding agent that mines your communications to discover the items you desire and what you are willing to pay for them and can figure out on its own which auctions to bid in on your behalf. From the agent communication perspective, however, the latter's sophistication does not matter – they are both autonomous agents.

Logical versus physical distribution Because of their autonomy, agents are the logical units of distribution: they can neither be aggregated nor decomposed into processes. Whenever an application involves two or more agents, there simply is no recourse but to consider their interactions. Constructs such as processes, by contrast, are physical units of distribution. The choice of whether an application is implemented as a single process or multiple ones is often driven by physical considerations such as geographical distribution, throughput, redundancy, number of available processors and cores, and so on. An agent itself may be implemented via multiple physical units of distribution; that choice, however, is immaterial from a multiagent systems perspective.

Heterogeneity refers to the diversity of agent implementations. The software engineering approach for accommodating heterogeneity is to make public the interdependencies among the components. A component can then be implemented based on what it depends on other components for (what it assumes) and what others depend on it for (what it guarantees) without concern for how the others are implemented. The same approach applies to agents. The specification of the interdependencies is essentially a protocol.

In traditional distributed systems, to accommodate heterogeneity, it is enough that protocols specify the schemas of the messages exchanged as well as their legal flows, that is, their ordering and occurrence. However, such a specification is inadequate for multiagent systems, wherein accommodating heterogeneity entails also specifying the semantics of the interaction. As an example, consider the finite state machine in Figure 3.1. It specifies the part of a purchase protocol that deals with making offers. This protocol involves two roles: buyer (b) and seller (s). The transitions are labeled with the messages. First, the seller sends an offer to the buyer. The buyer may then accept or reject the offer. After the buyer accepts, the seller may send an updated offer.

There is, however, an important element of the specification that is missing from this protocol. That element is what the messages mean in the real world. Making an offer in many settings would count as making a public, in other words,

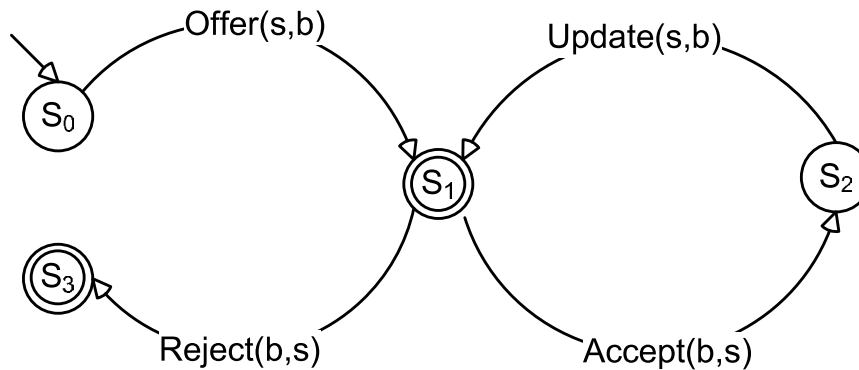


Figure 3.1: Updating an offer.

social commitment (more on social commitments later). Thus when the seller offers some book to the buyer for some price, it would mean that the seller is socially committed to the buyer for the offer. Consequently, updating an offer, for instance, by raising the price of the book, counts as updating the commitment. Specifically, it means that the old commitment is canceled and in its place a new one is created. Clearly, a protocol that specifies only the flow of messages, such as the one in Figure 3.1, does not capture such subtleties of meaning.

If the meanings of messages are not public, that would potentially make the agent non-interoperable. For example, this would happen if the buyer interprets the seller's offer as a commitment, but the seller does not. Their interaction would potentially break down. Accommodating semantic heterogeneity presupposes that we make the meanings of messages public as part of the protocol specification.

In practice, many multiagent protocols are specified as flows without reference to the message meanings. And they seem to work fairly well. In such cases, the designers of the agents agree off-line on how to interpret and process the messages and build this interpretation into the agents, thereby tightly coupling the agents.

1.2 Criteria for Evaluation

Communication has been studied in software engineering, distributed systems, and distributed artificial intelligence. Consequently, there are many approaches for specifying protocols. Later in the chapter, we discuss the major classes of approaches. Let us now motivate broad criteria by which to evaluate each approach.

Software engineering Ideally, protocols should be specified in terms of high-level abstractions that appeal to their stakeholders. In other words, protocol specifications should not be far removed from the expression of stakeholder

requirements. Protocol specifications should be modifiable, easily understandable, and composable. Further, they should promote loose coupling among agents.

Flexibility Agents should be able to enact protocols flexibly. Flexibility is especially important in dynamic settings where agents may come and go, and exceptions and opportunities may arise. Ideally, protocol specifications should constrain agents no more than is necessary to ensure correctness, where correctness is understood in connection with the application domain of interest.

Compliance checking An important standard of correctness is compliance. Checking an agent's compliance with a protocol means determining if the agent is following the protocol. To make such a determination presupposes both that a protocol is precise and that its standard of correctness is based on information that is accessible to the agents involved.

2 Conceptual Foundations of Communication in MAS

2.1 Communicative Acts

An important theme in the study of communication is *speech act theory*, better known as *communicative act theory*, since it has little specific connection with spoken communication. The main insight behind communicative act theory, due to the philosopher of language, John Austin, is that communication is a form of action. Specifically, we can think of communicative acts as those where “saying makes it so.” For example, when a judge declares a couple married, the judge is not merely reporting on some privately or publicly known fact; instead, the judge is bringing the fact into existence. The same may be said for a soccer umpire who ejects a player from the game. The umpire is not merely stating that the player is not allowed on the field for the duration of the game; the umpire is causing the player's permission to enter the field during the current game to be withdrawn. The judge and the umpire rely upon lower-level means to carry out the communicative acts. The judge may merely speak in public or sign a marriage certificate and affix his or her seal on it. The umpire may flash a red card at the player and speak out the player's jersey number. The physical means exist and information is transferred but what makes the communication a true communication is the convention in place in the given setting. Informally, we can think of the judge as saying, “I declare this couple man and wife” and the umpire as saying, “I declare this player as ejected from the game.”

Austin argued that all communications could be phrased in the above declarative form through the use of appropriate *performative* verbs. Thus a simple *informative* such as “the shipment will arrive on Wednesday” can be treated as if it were “I inform you that the shipment will arrive on Wednesday.” A *directive* such as “send me the goods” can be treated as if it were “I request that you send me the goods” or “I demand that you send me the goods” or other such variations. A *commissive* such as “I’ll pay you \$5” can be treated as if it were “I promise that I’ll pay you \$5.”

The above stylized construction has an important ramification for us as students of multiagent systems. It emphasizes that although what is being informed, requested, or promised may or may not be within the control of the informer, requester, or promiser, the fact that the agent chooses to inform, request, or promise another agent is entirely within its control. The above construction thus coheres with our multiagent systems thinking about autonomy and reflects the essence of the autonomous nature of communication as we explained above.

The above stylized construction has another more practical and arguably more nefarious ramification. Specifically, this is the idea that we can use the performative verb in the above to identify the main purpose or *illocutionary point* of a communication, separately from the propositional content of the communication. The underlying intuition is that the same propositional content could be coupled with different illocutionary points to instantiate distinct communicative acts. In computer science terms, the illocutionary points map to message types, and may be thought of as being the value of a message header. Following the shipment example above, we would associate the proposition “the shipment will arrive on Wednesday” with different message types, for example, *inform*, *request*, and *query*.

2.2 Agent Communication Primitives

As a result of the naturalness of the above mapping from illocutionary points to message types, it has been customary in agent communication languages to specify a small number of specialized message types as primitives. Having message types appears reasonable, but a pitfall lurks in this thinking. Because the literature describes a few broad-brush illocutionary points, existing approaches reflect the assumption that only a small number of primitives is adequate. They account for the meaning of each of these primitives. The above assumption proves erroneous because the applications of multiagent systems are manifold. In each application, the meanings that we need can be potentially distinct from the others. Thus the official meaning supplied by the agent communication language is insufficient, and developers end up adopting additional ad hoc meanings, which they hard-code into their agents. As a result, the agents become tightly coupled with each other.

Such coupling makes it difficult to change a multiagent system dynamically, by swapping out one agent for another as it were. Thus the potential benefit of using an agent communication language is lost.

In response to the above challenges, the newer approaches dispense with a fixed set of primitives based on illocutionary points. Instead, they provide an underlying set of abstractions that can be used to provide a formal semantics for any domain-specific primitives that a multiagent system may need. In other words, each domain is different, but there is an underlying logic-based representation in which the meanings of the terms used in the domain may be expressed.

For business applications, today, commitments are the key abstractions employed in the underlying representation. For example, in the stock-trading domain, we would see primitives such as *request stock quote* and *provide stock quote*. And, in the electronic commerce domain, we would see primitives such as *quote price*, *quote delivery charges*, and so on. The semantics of the primitives would be expressed in commitments. Notice that even apparently similar primitives may end up with completely different meanings, reflecting the needs and practices of the applicable domains. For example, in typical practice, a price quote is an offer to sell, meaning that the seller becomes committed to providing the specified item at the quoted price. In contrast, in typical practice, a stock quote carries no such connotation of an offer to sell – all it means is that the quoted price is the price at which the previous transaction was completed on the specified stock symbol, not that the brokerage who provided the quote is offering to sell you the stock for the quoted price. As you can well imagine, the meanings can easily be made more subtle and involved to capture the nuances of practical application scenarios.

Therefore, in a nutshell, it appears misguided to have a few (about a dozen or so) primitives with their unique definitions, hoping that they would cover all practical variations. For the above reason, we suggest that you read the literature on the primitives motivated from the illocutionary points, merely as showing illustrative examples – possibly even as important patterns but definitely not as an adequate basis for building a multiagent system for an arbitrary application.

3 Traditional Software Engineering Approaches

We referred above to low-level distributed computing protocols as a way to explain architectures in general. We argued that we need to consider multiagent systems and high-level protocols as a way to specify architectures that yield interoperability at a level closer to application needs. However, traditional software engineering arguably addresses the challenges of interoperability too. Would it

be possible to adopt software engineering techniques as a basis for dealing with agent communication?

The above view has received a significant amount of attention in the literature. Partly because of the apparent simplicity of traditional techniques and largely because of their familiarity to researchers and practitioners alike, the traditional techniques continue to garner much interest in the agents community.

The traditional techniques leave the formulation of the message syntax open – a message could be any document and in common practice is an XML document. And they disregard the application meaning of the messages involved. Instead, these techniques focus on the operational details of communication, mostly concentrating on the occurrence and ordering of messages.

Thus a protocol may be specified in terms of a finite-state machine (FSM), which describes its states and legal transitions from a centralized perspective. Formally, this may be done in a variety of ways, including state machines [8, 58], Petri nets [18], statecharts [24], UML sequence diagrams [35], process algebras such as the pi-calculus [9], and logic-based or declarative approaches [47, 54]. All of these approaches specify a set of message occurrences and orderings that are deemed to capture the protocol being specified. We discuss a few of these below.

The above-mentioned traditional representations have the advantage of there being a number of formal tools for verifying and even validating specifications written in those representations. Thus a protocol designer would be able to determine if a protocol in question would satisfy useful properties such as termination. Implementing the endpoints or agents to satisfy such specifications is generally quite straightforward. Checking compliance with the specification is also conceptually straightforward. As long as the messages observed respect the ordering and occurrence constraints given by a protocol, the enactment is correct with respect to the protocol; otherwise, an enactment is not correct.

However, the value of such tools is diminished by the fact that in the traditional representations there is no clear way to describe the meanings of the interactions. In other words, these approaches lack an independent application-centric standard of correctness. For example, let us suppose that a protocol happens to specify that a merchant ships the goods to the customer and then the customer pays. Here, if the customer happens to pay first, that would be a violation of the protocol. In informal terms, we should not care. It should be the customer's internal decision whether to pay first. If the customer does (taking the risk of paying first or losing bank interest on the money paid), that is the customer's prerogative. However, given the traditional, operational specification, any such deviation from the stated protocol is equally unacceptable. Notice that it may in fact be in the customer's interest to pay first, for example, to include the expense in the current year's tax deductions. But we have no way of knowing that.

Instead, if the protocol could be specified in terms of the meanings of the communications involved, we would naturally express the intuition that all we expect is that the customer eventually pays or that the customer pays no later than some other crucial event. If the customer fails to pay, that would be a violation. But if the customer pays early, so much the better.

3.1 Choreographies

The service-oriented computing literature includes studies of the notion of a *choreography*. A choreography is a specification of the message flow among the participants. Typically, a choreography is specified in terms of *roles* rather than the participants themselves. Involving roles promotes reusability of the choreography specification. Participants *adopt* roles, that is, bind to the roles, in the choreography.

A choreography is a description of an interaction from a shared or, more properly, a *neutral* perspective. In this manner, a choreography is distinguished from a specification of a *workflow*, wherein one party drives all of the other parties. The latter approach is called an *orchestration* in the services literature.

An advantage of adopting a neutral perspective, as in a choreography, is that it better applies in settings where the participants retain their autonomy: thus it is important to state what each might expect from the others and what each might offer to the others. Doing so promotes loose coupling of the components: centralized approaches could in principle be equally loosely coupled but there is a tendency associated with the power wielded by the central party to make the other partners fit its mold. Also, the existence of the central party and the resulting regimentation of interactions leads to implicit dependencies and thus tight coupling among the parties.

A neutral perspective yields a further advantage that the overall computation becomes naturally distributed and a single party is not involved in mediating all information flows. A choreography is thus a way of specifying and building distributed systems that among the conventional approaches most closely agrees with the multiagent systems' way of thinking. But important distinctions remain, which we discuss below.

WS-CDL [57] and ebBP [25] are the leading industry supported choreography standardization efforts. WS-CDL specifies choreographies as message exchanges among partners. WS-CDL is based on the pi-calculus, so it has a formal operational semantics. However, WS-CDL does not satisfy important criteria for an agent communication formalism. First, WS-CDL lacks a theory of the meanings of the message exchanges. Second, when two or more messages are performed within a given WS-CDL choreography, they are handled sequentially by default, as in an MSC. Third, WS-CDL places into a choreography actions that

would be private to an agent, such as what it should do upon receiving a message. Fourth, for nested choreographies, WS-CDL relies upon local decision making by an agent, such as whether to forward a request received in one choreography to another [50].

3.2 Sequence Diagrams

The most natural way to specify a protocol is through a message sequence chart (MSC), formalized as part of UML as sequence diagrams [28]. The roles of a protocol correspond to the lifelines of an MSC; each edge connecting two lifelines indicates a message from a sender to a receiver. Time flows downward by convention and the ordering of the messages is apparent from the chart. MSCs support primitives for grouping messages into blocks. Additional primitives include alternatives, parallel blocks, or iterative blocks. Although we do not use MSCs extensively, they provide a simple way to specify agent communication protocols.

FIPA (Foundation of Intelligent Physical Agents) is a standards body, now part of the IEEE Computer Society, which has formulated agent communication standards. FIPA defines a number of interaction protocols. These protocols involve messages of the standard types in FIPA. Each FIPA protocol specifies the possible ordering and occurrence constraints on messages as a UML sequence diagram supplemented with some informal documentation.

Figure 3.2 shows the FIPA request interaction protocol in FIPA's variant of the UML sequence diagram notation [26]. This protocol involves two roles, an INITIATOR and a PARTICIPANT. The INITIATOR sends a *request* to the PARTICIPANT, who either responds with a *refuse* or an *agree*. In the latter case, it follows up with a detailed response, which could be a *failure*, an *inform-done*, or an *inform-result*. The PARTICIPANT may omit the *agree* message unless the INITIATOR asked for a notification.

The FIPA request protocol deals with the operational details of when certain messages may or must be sent. It does not address the meanings of the messages themselves. Thus it is perfectly conventional in this regard. Where it deviates from traditional distributed computing is in the semantics it assigns to the messages themselves, which we return to below. However, the benefit of having a protocol is apparent even in this simple example: it identifies the roles and their mutual expectations and thus decouples the implementations of the associated agents from one another.

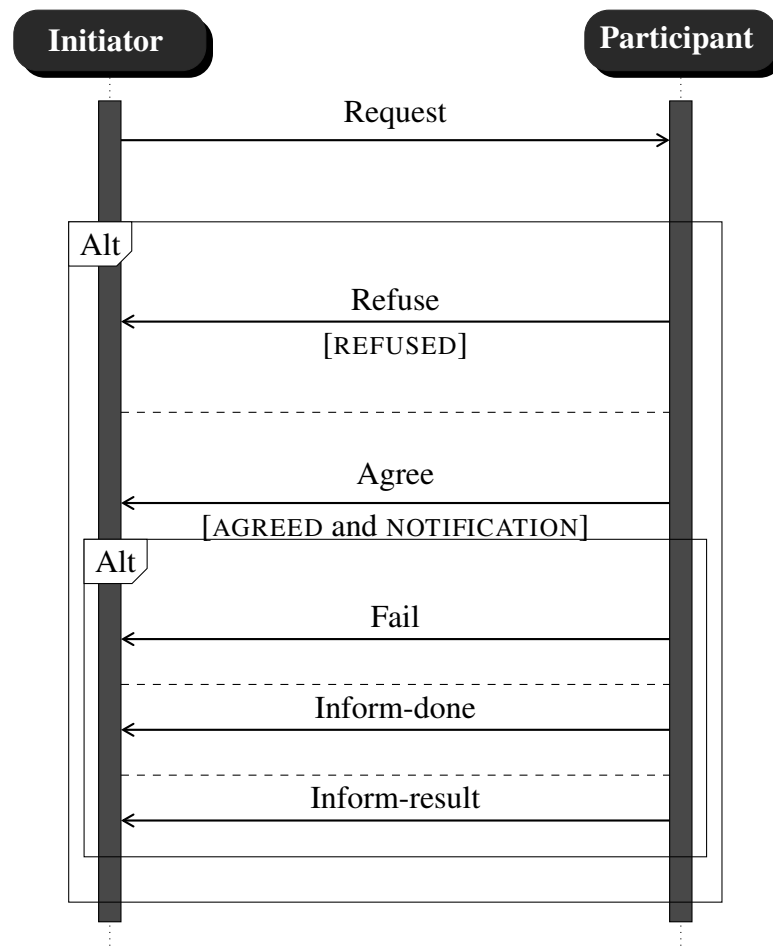


Figure 3.2: FIPA request interaction protocol, from the FIPA specification [26], expressed as a UML sequence diagram.

3.3 State Machines

Figure 3.3 shows a state machine between two roles, merchant (mer) and customer (cus) as a state machine. The transitions are labeled with messages; the prefix mer, cus indicates a message from the merchant to the customer, and cus, mer indicates a message from the customer to the merchant. This state machine supports two executions. One execution represents the scenario where the customer rejects the merchant's offer. The other execution represents the scenario where the customer accepts the offer, following which the merchant and the customer exchange the item and the payment for the item. In the spirit of a state machine, Figure 3.3 does not reflect the internal policies based upon which the customer accepts an offer.

Consider the state machine in Figure 3.4. The dotted paths indicate two addi-

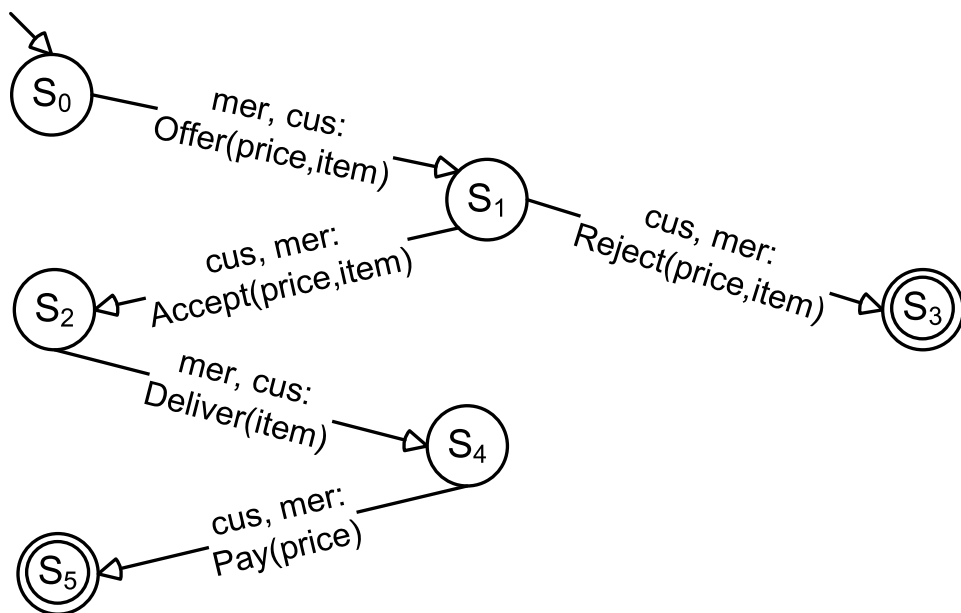


Figure 3.3: A protocol specified as a state machine.

tional executions that are not supported by the state machine in Figure 3.3. The executions depict the scenarios where the customer sends the payment upon receiving an offer and after sending an accept, respectively. These additional executions are just as sensible as the original ones. However, in the context of the state machine in Figure 3.3, these executions are trivially *noncompliant*. The reason is that checking compliance with choreographies is purely syntactical – the messages have to flow between the participants exactly as prescribed. Clearly, this curbs the participants’ autonomy and flexibility.

We can attempt to ameliorate the situation by producing ever larger FSMs that include more and more paths. However, doing so complicates the implementation of agents and the task of comprehending and maintaining protocols, while not supporting any real run-time flexibility. Further, any selection of paths will remain arbitrary.

3.4 Evaluation with Respect to MAS

Traditional software engineering approaches for specifying protocols are operational in nature. Instead of specifying the meaning of a communication, they specify the flow of information among agents. The lack of meaning leads to the following observations about protocols produced following traditional approaches.

Software engineering Because the protocols specify the set of possible enact-

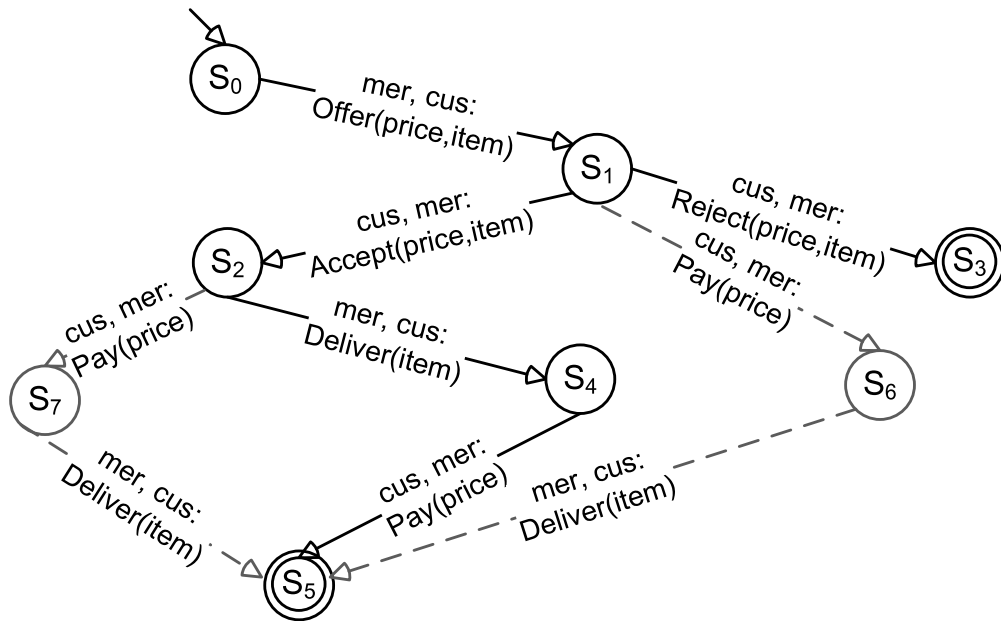


Figure 3.4: An alternative, more flexible state machine.

ments at a low level of abstraction, any but the most trivial are difficult to design and maintain. It is difficult to map the business requirements of stakeholders to the protocols produced.

Flexibility Agents have little flexibility at runtime; the protocols essentially dictate agent skeletons. Any deviation from a protocol by an agent, no matter how sensible from a business perspective, is a violation. Further, to enable interoperation, the protocols are specified so that they produce lock-step synchronization among agents, which also limits flexibility.

Compliance Checking an agent's compliance with the protocol is easy: computationally, it is akin to verifying whether a string is accepted by an FSM. However, that ease comes at the expense of flexibility.

4 Traditional AI Approaches

The traditional AI approaches to agent communication begin from the opposite extreme. These approaches presume that the agents are constructed based on cognitive concepts, especially, beliefs, goals, and intentions. Then they specify the communication of such agents in terms of how the communication relates to their cognitive representations.

The AI approaches came from two related starting points, which has greatly affected how they were shaped. The first starting point was of human-computer interaction broadly and natural language understanding specifically. The latter includes the themes of discourse understanding from text or speech, and speech understanding. What these approaches had in common was that they were geared toward developing a tool that would assist a user in obtaining information from a database or performing simple transactions such as booking a train ticket. A key functionality of such tools was to infer what task the user needed to perform and to help the user accordingly. These tools maintained a user model and were configured with a domain model upon which they reasoned via heuristics to determine how best to respond to their user's request, and potentially to anticipate the user's request.

Such a tool was obviously cooperative: its *raison d'être* was to assist its user and failure to be cooperative would be simply unacceptable. Further, it was an appropriate engineering assumption that the user was cooperative as well. That is, the tool could be based on the idea that the user was not purposefully misleading it, because a user would gain nothing in normal circumstances by lying about its needs and obtaining useless responses in return.

As the tools became more proactive they began to be thought of as agents. Further, in some cases the agents of different users could communicate with one another, not only with their users. The agents would maintain their models of their users and others based on the communications exchanged. They could make strong inferences regarding the beliefs and intentions of one another, and act and communicate accordingly. These approaches worked for their target setting. To AI researchers, the approaches these agents used for communicating with users and other agents appeared to be applicable for agent communication in general.

The second body of work in AI that related to agent communication came from the idea of building distributed knowledge-based systems (really just expert systems with an ability to communicate with each other). The idea was that each agent would include a reasoner and a knowledge representation and communication was merely a means to share such knowledge. Here, too, we see the same two assumptions as for the human interaction work. First, that the member agents were constructed with the same knowledge representations. Second, that the agents were largely cooperative with each other.

4.1 KQML

Agent communication languages began to emerge in the 1980s. These were usually specific to the projects in which they arose, and typically relied on the specific internal representations used within the agents in those projects.

Somewhat along the same lines, but with some improved generality, arose the Knowledge Query and Manipulation Language or KQML. KQML was created by the DARPA Knowledge Sharing Effort, and was meant to be an adjunct to the other work on knowledge representation technologies, such as ontologies. KQML sought to take advantage of a knowledge representation based on the construct of a knowledge base, such as had become prevalent in the 1980s. Instead of a specific internal representation, KQML assumes that each agent maintains a knowledge base described in terms of knowledge (more accurately, belief) assertions.

KQML proposed a small number of important primitives, such as *query* and *tell*. The idea was that each primitive could be given a semantics based on the effect it had on the knowledge bases of the communicating agents. Specifically, an agent would send a *tell* for some content only if it believed the content, that is, the content belonged in its knowledge base. And, an agent who received a *tell* for some content would insert that content into its knowledge base, that is, it would begin believing what it was told.

Even though KQML uses knowledge as a layer of abstraction over the detailed data structures of the internal implementation of agents, it turns out to be overly restricted in several ways. The main assumption of KQML is that the communicating agents are cooperative and designed by the same designers. Thus the designers would make sure that an agent sent a message, such as a *tell*, only under the correct circumstances and an agent who received such a message could immediately accept its contents. When the agents are autonomous, they may generate spurious messages – and not necessarily due to malice.

KQML did not provide a clear basis for agent designers to choose which of the message types to use and how to specify their contents. As a result, designers all too often resolved to using a single message type, typically *tell*, with all meanings encoded (usually in some ad hoc manner) in the contents of the messages. That is, the approach is to use different *tell* messages with arbitrary expressions placed within the contents of the messages.

The above challenges complicated interoperability so that it was in general difficult if not impossible for agents developed by different teams to be able to successfully communicate with one another.

4.2 FIPA ACL

We discussed the FIPA interaction protocols in Section 3.2. FIPA has also produced the FIPA ACL, one of the motivations behind which was to address the challenges with KQML. A goal for the FIPA ACL (Agent Communication Language) was to specify a definitive syntax through which interoperability among agents created by different developers could be facilitated. In addition, to ensure interoperability, the FIPA ACL also specified the semantics of the primitives.

Like the KQML semantics, the FIPA ACL semantics is mentalist, although it has a stronger basis in logic. The FIPA ACL semantics is based on a formalization of the cognitive concepts such as the beliefs and intentions of agents.

Beliefs and intentions are suitable abstractions for designing and implementing agents. However, they are highly unsuitable as a basis for an agent communication language. A communication language supports the interoperation of two or more agents. Thus it must provide a basis for one agent to compute an abstraction of the local state of another agent. The cognitive concepts provide no such basis in a general way. They lead to the internal implementations of the interacting agents to be coupled with each other. The main reason for this is that the cognitive concepts are definitionally internal to an agent. For example, consider the case where a merchant tells a customer that a shipment will arrive on Wednesday. When the shipment fails to arrive on Wednesday, would it be any consolation to the customer that the merchant sincerely believed that it was going to? The merchant could equally well have been lying. The customer would never know without an audit of the merchant's databases. In certain legal situations, such audits can be performed but they are far from the norm in business encounters.

One might hope that it would be possible to infer the beliefs and intentions of another party, but it is easy to see with some additional reflection that no unique characterization of the beliefs and intentions of an agent is possible. In the above example, maybe the merchant had a sincere but false belief; or, maybe the merchant did not have the belief it reported; or, maybe the merchant was simply unsure but decided to report a belief because the merchant also had an intention to consummate a deal with the customer.

It is true that if one developer implements all the interacting agents correctly, the developer can be assured that an agent would send a particular message only in a particular internal state (set of beliefs and intentions). However such a multi-agent system would be logically centralized and would be of severely limited value.

It is worth pointing out that the FIPA specifications have ended up with a split personality. FIPA provides the semiformal specification of an agent management system, which underlies the well-regarded JADE system [7]. FIPA also provides definitions for several interaction protocols (discussed in Section 3.2), which are also useful and used in practice, despite their limitations. FIPA provides a formal semantics for agent communication primitives based on cognitive concepts, which gives a veneer of rigor, but is never used in multiagent systems.

4.3 Evaluation with Respect to MAS

The traditional AI approaches are mentalist, which render them of limited value for multiagent systems.

Software engineering The AI approaches offer high-level abstractions, which is a positive. However, because the abstractions are mentalist, the approaches cannot be applied to the design of multiagent systems except in the restricted case where one developer designs all the agents (as explained above). Further, recall the discussion from Section 2.2 regarding the unsuitability of a small set of primitives. Both KQML and FIPA suffer from this problem.

Flexibility The flexibility of agents is severely curtailed because of restrictions on when agents can send particular communications.

Compliance It is impossible for an observer to verify the cognitive state of an agent. Hence verifying agent compliance (for example, if the agent has the requisite cognitive state for sending a particular message) is impossible.

5 Commitment-Based Multiagent Approaches

In contrast with the operational approaches, commitment protocols give primacy to the *business meanings* of service engagements, which are captured through the participants' *commitments* to one another [60], [11, 22, 52, 56], [20]. Computationally, each participant is modeled as an *agent*; interacting agents carry out a service engagement by creating and manipulating commitments to one another.

5.1 Commitments

A commitment is an expression of the form

$$C(\textit{debtor}, \textit{creditor}, \textit{antecedent}, \textit{consequent}) ,$$

where *debtor* and *creditor* are agents, and *antecedent* and *consequent* are propositions. A commitment $C(x, y, r, u)$ means that x is committed to y that if r holds, then it will bring about u . If r holds, then $C(x, y, r, u)$ is *detached*, and the commitment $C(x, y, \top, u)$ holds (\top being the constant for truth). If u holds, then the commitment is *discharged* and does not hold any longer. All commitments are *conditional*; an unconditional commitment is merely a special case where the antecedent equals \top . Examples 3.1–3.3 illustrate these concepts. In the examples below, EBook is a bookseller, and Alice is a customer.)

Example 3.1 (*Commitment*) $C(\textit{EBook}, \textit{Alice}, \$12, \textit{BNW})$ means that EBook commits to Alice that if she pays \$12, then EBook will send her the book Brave New World.

Example 3.2 (*Detach*) If Alice makes the payment, that is, if \$12 holds, then $C(EBook, Alice, \$12, BNW)$ is detached. In other words, $C(EBook, Alice, \$12, BNW) \wedge \$12 \Rightarrow C(EBook, Alice, \top, BNW)$.

Example 3.3 (*Discharge*) If EBook sends the book, that is, if BNW holds, then both $C(EBook, Alice, \$12, BNW)$ and $C(EBook, Alice, \top, BNW)$ are discharged. In other words, $BNW \Rightarrow \neg C(EBook, Alice, \$12, BNW) \wedge \neg C(EBook, Alice, \top, BNW)$.

Importantly, commitments can be manipulated, which supports flexibility. The commitment operations [45] are listed below; CREATE, CANCEL, and RELEASE are two-party operations, whereas DELEGATE and ASSIGN are three-party operations.

- $CREATE(x, y, r, u)$ is performed by x , and it causes $C(x, y, r, u)$ to hold.
- $CANCEL(x, y, r, u)$ is performed by x , and it causes $C(x, y, r, u)$ to not hold.
- $RELEASE(x, y, r, u)$ is performed by y , and it causes $C(x, y, r, u)$ to not hold.
- $DELEGATE(x, y, z, r, u)$ is performed by x , and it causes $C(z, y, r, u)$ to hold.
- $ASSIGN(x, y, z, r, u)$ is performed by y , and it causes $C(x, z, r, u)$ to hold.
- $DECLARE(x, y, r)$ is performed by x to inform y that the r holds.

DECLARE is not a commitment operation, but may indirectly affect commitments by causing detaches and discharges. In relation to Example 3.2, when Alice informs EBook of the payment by performing $DECLARE(Alice, EBook, \$12)$, then the proposition \$12 holds, and causes a detach of $C(EBook, Alice, \$12, BNW)$.

Further, a commitment arises in a social or legal context. The context defines the rules of encounter among the interacting parties, and often serves as an arbiter in disputes and imposes penalties on parties that violate their commitments. For example, eBay is the context of all auctions that take place through the eBay marketplace; if a bidder does not honor a payment commitment for an auction that it has won, eBay may suspend the bidder's account.

A formal treatment of commitments and communication based on commitments is available in the literature [15, 48].

5.2 Commitment Protocol Specification

Table 3.1 shows the specification of a commitment protocol between a merchant and a customer (omitting sort and variable declarations). It simply states the meanings of the messages in terms of the commitments arising between the

<i>Offer</i> (<i>mer, cus, price, item</i>)	means	CREATE(<i>mer, cus, price, item</i>)
<i>Accept</i> (<i>cus, mer, price, item</i>)	means	CREATE(<i>cus, mer, item, price</i>)
<i>Reject</i> (<i>cus, mer, price, item</i>)	means	RELEASE(<i>mer, cus, price, item</i>)
<i>Deliver</i> (<i>mer, cus, item</i>)	means	DECLARE(<i>mer, cus, item</i>)
<i>Pay</i> (<i>cus, mer, price</i>)	means	DECLARE(<i>cus, mer, price</i>)

Table 3.1: A commitment protocol.

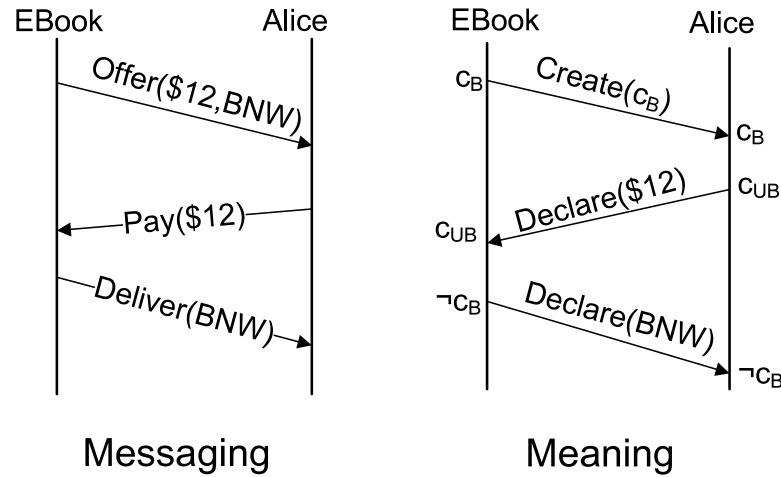


Figure 3.5: Distinguishing message syntax and meaning: two views of the same enactment.

merchant and customer. For instance, the message *Offer*(*mer, cus, price, item*) means the creation of the commitment $C(\text{mer}, \text{cus}, \text{price}, \text{item})$, meaning the merchant commits to delivering the item if the customer pays the price; *Reject*(*cus, mer, price, item*) means a release of the commitment; *Deliver*(*mer, cus, item*) means that the proposition *item* holds.

Figure 3.5 (left) shows an execution of the protocol and Figure 3.5 (right) its meaning in terms of commitments. (The figures depicting executions use a notation similar to UML interaction diagrams. The vertical lines are agent lifelines; time flows downward along the lifelines; the arrows depict messages between the agents; and any point where an agent sends or receives a message is annotated with the commitments that hold at that point. In the figures, instead of writing CREATE, we write *Create*. We say that the *Create* message realizes the CREATE operation. Likewise, for other operations and DECLARE.) In the figure, the merchant and customer roles are played by EBook and Alice, respectively; c_B and c_{UB}

are the commitments $C(EBook, Alice, \$12, BNW)$ and $C(EBook, Alice, \top, BNW)$, respectively.

5.3 Evaluation with Respect to MAS

Compliance Protocol enactments can be judged correct as long as the parties involved do not violate their commitments. A customer would be in violation if it keeps the goods but fails to pay. In this manner, commitments support business-level compliance and do not dictate specific operationalizations [22].

Flexibility The above formulation of correctness enhances flexibility over traditional approaches by expanding the operational choices for each party [13]. For example, if the customer substitutes a new way to make a payment or elects to pay first, no harm is done, because the behavior is correct at the business level. And the merchant may employ a new shipper; the customer may return damaged goods for credit; and so on. By contrast, without business meaning, exercising any such flexibility would result in non-compliant executions.

Software Engineering Commitments offer a high-level abstraction for capturing business interactions. Further, a commitment-based approach accommodates the autonomy of the participants in the natural manner: socially, an agent is expected to achieve no more than its commitments. Commitments thus also support loose coupling among agents. Commitment-based approaches offer a compelling alternative to the traditional software engineering approaches described in Section 3 for building systems comprised of autonomous agents.

Figure 3.6 shows some of the possible enactments based on the protocol in Table 3.1. The labels c_A and c_{UA} are $C(Alice, EBook, BNW, \$12)$ and $C(Alice, EBook, \top, \$12)$, respectively. Figure 3.6(B) shows the enactment where the book and payment are exchanged in Figure 3.3. Figures 3.6(A) and (C) show the additional executions supported in Figure 3.4; Figure 3.6(D) reflects a new execution that we had not considered before, one where Alice sends an *Accept* even before receiving an offer. All these executions are compliant executions in terms of commitments, and are thus supported by the protocol in Table 3.1.

Table 3.2 summarizes the three approaches.

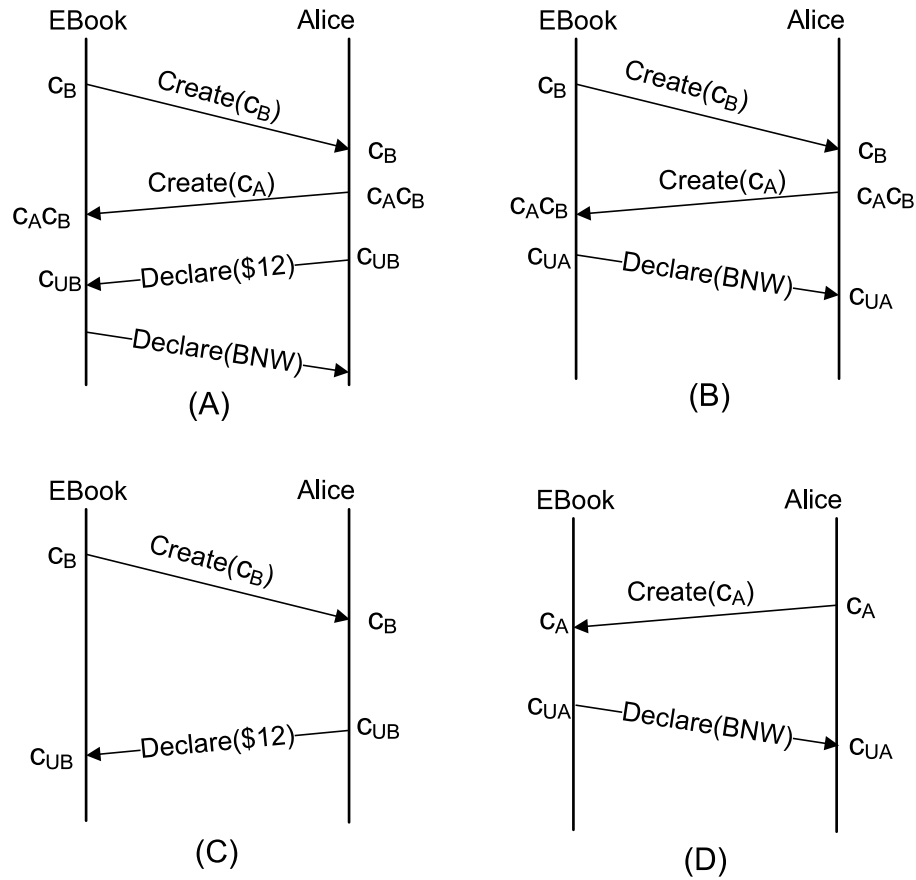


Figure 3.6: Flexible enactment.

	Traditional SE	Traditional AI	Commitment Protocols
<i>Abstraction</i>	control flow	mentalist	business relationship
<i>Compliance</i>	lexical basis	unverifiable	semantic basis
<i>Flexibility</i>	low	low	high
<i>Interoperability</i>	message level	integration	business level

Table 3.2: Comparison of agent communication approaches.

6 Engineering with Agent Communication

Protocols support the development of distributed systems. A natural way to apply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and

implementing the agents who would participate in the given protocol. Role specifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles suggests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those included in the protocol and which feature in the individual role specifications. In other words, we would like the agents implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. Any message exchange involves two agents (neglecting multicast across roles): the sender and the receiver. Any message sent by the sender is received by the receiver. Thus it is easy to ensure their joint computations generate correct outcomes. In systems with three or more roles, however, whenever a message exchange occurs, one or more of the other roles would be left unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this non-existence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role – in this metaphor, by fleshing out its skeleton – provides methods to process each incoming message and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite-state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's specification is very much focused on the perspective of the role whereas the machine of a protocol describes the progress of a protocol enactment from a neutral perspective.

6.1 Programming with Communications

The Java Agent Development Framework (JADE) is a popular platform for developing and running agent-based applications. It implements the FIPA protocols discussed earlier. JADE provides support for the notion of what it terms *behaviors*. A behavior is an abstract specification of an agent that characterizes important events such as the receipt of specified messages and the occurrence of timeouts. To implement an agent according to a behavior involves defining the methods

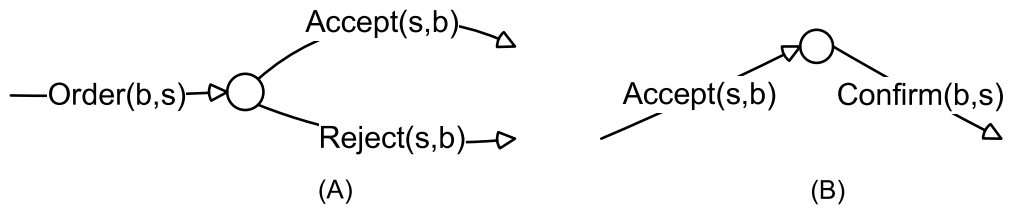


Figure 3.7: Example of operational patterns.

it specifies as callbacks. In particular, a role skeleton can be implemented by defining the handlers for any incoming methods. The JADE tutorial online offers comprehensive instructions for building JADE applications.

6.2 Modeling Communications

It is not trivial to specify the *right* commitments for particular applications. For instance, Desai et al. [19] show how a scenario dealing with foreign exchange transactions may be formalized in multiple ways using commitments, each with different ramifications on the outcomes. The challenge of specifying the right commitments leads us to the question: *How can we guide software engineers in creating appropriate commitment-based specifications?*

Such guidance is often available for operational approaches such as state machines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 3.7 shows two common patterns expressed as (partial) state machines, which can aid software engineers in specifying operational interactions. Here, *b* and *s* are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.

By contrast, commitment protocols abstract away from operational details, focusing on the meanings of messages, not their flow. Clearly, operational patterns such as the above would not apply to the design of commitment protocols. What kinds of patterns would help in the design of commitment protocols? By and large, they would need to be *business patterns* – characterizing requirements, not operations – that emphasize meanings in terms of commitments. In contrast with Figure 3.7, these patterns describe what it *means* to make, accept, reject, or update an offer, not when to send specific messages.

Business patterns support specifying business protocols. These patterns are motivated by the following principles.

Autonomy compatibility Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the

appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed would cause hidden coupling.

Explicit meanings The meaning ought to be made public, not hidden within agent implementations.

6.2.1 Business Patterns

Business patterns encode the common ways in which businesses engage each other. Below is an example of the *compensation* pattern.

- COMPENSATION

Intent To compensate the creditor in case of commitment cancellation or violation by the debtor.

Motivation It is not known in advance whether a party will fulfill its commitments; compensation commitments provide some assurance to the creditor in case of violations.

Implementation $Compensate(x, y, r, u, p)$ means
 $Create(x, y, violated(x, y, r, u), p)$.

Example $Compensate(mer, cus, price, item, discount)$; it means that the merchant will offer the customer a discount on the next purchase if the item is paid for but not delivered.

Consequences A commitment (even a compensation commitment) should ideally be supported by compensation; however, at some level, the only recourse is escalation to the surrounding business *context* – for example, the local jurisdiction [51].

6.2.2 Enactment Patterns

Whereas a business pattern describes the meaning of communication, an enactment pattern describes the conditions under which an agent should enact a business pattern, that is, *when* to undertake the corresponding communication. A locus of such enactments may serve as the basic agent skeleton.

- COUNTER OFFER

Intent One party makes an offer to another, who responds with a modified offer of its own.

Motivation Essential for negotiation.

When Let $C(x, y, r, u)$ be the commitment corresponding to the original offer. Making a counteroffer would amount to creating the commitment $C(y, x, u', r')$ such that $u' \vdash u$ and $r \vdash r'$, in other words, if the consequent is strengthened and the antecedent is weakened. An alternative implementation includes doing *Release*(x, y, r, u) in addition.

Example Let's say $C(\text{EBook}, \text{Alice}, \$12, \text{BNW})$ holds. Alice can make the counteroffer $C(\text{Alice}, \text{EBook}, \text{BNW} \wedge \text{Dune}, \$12)$, meaning that she wants *Dune* in addition to *BNW* for the same price.

Consequences When $u \equiv u'$ and $r \equiv r'$, the counteroffer amounts to a mutual commitment.

6.2.3 Semantic Antipatterns

Semantic antipatterns identify forms of representation and reasoning to be avoided because they conflict with the autonomy of the participants or with a logical basis for commitments.

- COMMIT ANOTHER AS DEBTOR

Intent An agent creates a commitment in which the debtor is another agent.

Motivation To capture delegation, especially in situations where the delegator is in a position of power over the delegatee.

Implementation The sender of *Create*(y, z, p, q) is x (x and y are different agents), thus contravening the autonomy of y .

Example Consider two sellers EBook and BookWorld. EBook sends *Create*(*BookWorld*, *Alice*, *\$12*, *BNW*) to Alice, which violated BookWorld's autonomy.

Consequences A commitment represents a public undertaking by the debtor. A special case is when $x = z$. That is, x unilaterally makes itself the creditor.

Criteria Failed y 's autonomy is not respected.

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments. In the above example, BookWorld could have a standing commitment with EBook to accept delegations. EBook can then send a delegate "instruction" to BookWorld upon which BookWorld commits to Alice.

The above are some examples of patterns. For a more exhaustive list of patterns, see [16].

6.3 Communication-Based Methodologies

Because of the centrality of agent communication to multiagent systems, a number of methodologies for designing and implementing multiagent systems are based on communications. We point out a few such methodologies in the Further Reading section.

The common idea behind these methodologies is to identify the communications involved in the system being specified and to state the meanings of such communications. The main protocol concepts are roles, messages, and message meanings. Below we briefly outline the high-level considerations involved in designing a protocol.

- Identify stakeholder requirements.
- Identify the roles involved in the interaction. Let's say the roles identified are customer, merchant, shipper, and banker.
- If a suitable protocol is available from a repository, then choose it and we're done. After all, one of the key benefits of protocols is reusability. For instance, suppose the stakeholders wanted to design a purchase protocol. If the protocol of Table 3.1 fits their requirements, we're done.
- Often the required protocol may be obtained by *composing* existing protocols. For example, the desired protocol could potentially be obtained by combining *Ordering*, *Payment*, and *Shipping* protocols.
- Sometimes the protocol or parts of it may need to be written up from scratch. Identify the communications among the roles. For example, there would be messages between the customer and the merchant that would pertain to ordering items. The messages between the customer and bank would pertain to payment, and so on.
- Identify how each message would affect the commitments of its sender and receiver. For example, the *Offer* message could be given a meaning similar to the one in Table 3.1. The customer's payment to the bank would effectively discharge his or her commitment to pay the merchant. Similarly, the delivery of the goods by the shipper would effectively discharge the merchant's commitment to pay, and so on.

7 Advanced Topics and Challenges

This section describes some important current directions in agent communication.

7.1 Primacy of Meaning

As we outlined in the foregoing, there is an unfortunate tendency to specify communication protocols in operational terms at the cost of the meanings that they convey. However, agent communication should be understood at the level of the “social state” of the parties involved and how it affects and is affected by communications. Adopting a meaning-based stance protects one’s models from inadvertent dependencies upon implementation and yields the highest flexibility for the participating agents while maintaining correctness.

The earlier meaning-based approaches to agent communication tended to combine assertions regarding the meanings of communications with operational details, such as the conditions under which communication must occur and how the communications must be mutually ordered. Such operational details interfere with an application of meaning-based reasoning because they require that the agents maintain not only the meanings of the communication and the changing social state but also additional, otherwise irrelevant dependencies with the decisions of other agents.

We have not been able to find even a single compelling “natural” situation where such details are necessary. Any requirement that an agent produce a message is a violation of its autonomy. When we think of meaning properly, there is never a natural need for ordering constraints – the only ordering constraints that might arise are those based on artificial grounds such as arbitrary conventions in a particular domain. Such conventions are fine and an approach for agent communication should support them. However, they do not explain the large number of ordering constraints that traditional specifications tend to include.

Although the operational details interfere with reasoning about meaning, they are essential to ensure that each party obtains the information it needs at the right time so as to proceed effectively. The recent approach termed the Blindingly Simple Protocol Language [49] provides a simple resolution to this tension by capturing the necessary operational details in a declarative manner. The declarative representation of messages facilitates producing assertions regarding social state from them, and using such assertions as a basis for reasoning about the meanings of the messages.

A research challenge, then, is to develop languages and methodologies in which (and with which to formulate) proper meanings for communications, so as to capture the needs of domain settings precisely.

7.2 Verifying Compliance

Because agent communication involves the interactions of two or more autonomous parties, it inherently has the weight of a “standard” – albeit a minor – non-universal standard. In other words, when two agents talk to one another, they must agree sufficiently on what they are talking about and they must be able to judge if their counterparty is interacting in a manner that they would expect. To the first point, the traditional approaches missed stating expectations properly.

Just as a standard in any domain of practice is worthless if we cannot judge whether the parties subject to the standard are complying with it or not, so it is with agent communication. Any approach for agent communication must support the statement of the mutual expectations of the parties involved *and* do so in a manner that supports each party verifying if the others are complying with its expectations of them. This is an obvious point in retrospect. However, the mentalist approaches disregarded the problem of compliance. Despite this point having been explained over a decade ago [44], there remains a tendency to disregard it in approaches to communication, especially as such approaches are applied within software engineering methodologies.

A research challenge here is to design specification languages that promote the verification of compliance and, more importantly, to develop algorithms by which an agent or a set of cooperating agents could verify the compliance of others based on the communications it can monitor.

7.3 Protocol Refinement and Aggregation

If we are to treat communication as a first-class abstraction for specifying multi-agent systems, we must be ready to support dealing with conceptual modeling using that abstraction. Classically, two conceptual modeling relations are known: refinement and aggregation. *Refinement* deals with how a concept refines another in the sense of the is-a hierarchy. *Aggregation* deals with how concepts are put together into composites in the sense of the part-whole hierarchy. Refinement and aggregation are well-understood for traditional object-oriented design and are supported by modern programming languages.

However, dealing with refinement in particular has been non-trivial for communication protocols. Recent work on session types is promising in this regard [32], as is work on refinement with respect to commitment-based protocols [30]. An important challenge is to produce a generalized theory and associated languages and tools that would support refinement and aggregation of protocols for more powerful meaning specifications.

7.4 Role Conformance

As we stated above, the meaning of communication captures the expectations that the parties involved can have of each other. Accordingly, an important engineering challenge is to develop agents who would meet such expectations. An agent can potentially apply complex reasoning, and, therefore, verifying that an agent (implementation) would meet the expectations of another agent is non-trivial.

A natural way to approach the problem is to formulate a role description or a role *skeleton* based on the specification of a communication protocol. A skeleton describes the basic structure of a role. An agent who plays (and hence implements) a role would provide additional details so as to flesh out the structure that is the skeleton. Since a protocol involves two or more roles, the challenge is to determine sufficient structural properties of each role, in terms of what messages it can receive and send under what circumstances and any constraints on how the local representation of the social state should progress in light of the messages received and sent. We can then publish the descriptions of each role in a protocol along with the protocol specification.

At the same time, one can imagine that software vendors may produce agent implementations that are compatible with different roles. A vendor would not and should not provide the internal details but would and should provide the public “interface” of the agent in terms of its interactions. In other words, a vendor would describe a role that its agent would be able to play. In general, an agent may need to participate in more than one protocol. Thus it would help to know if the role as published by a vendor conforms with the role as derived from a protocol. This is the problem of *role conformance*. Solving this problem for a particular language would help automate part of the task of creating a multiagent system from disparate agents while ensuring that the agents, even if implemented heterogeneously, would be able to interoperate with respect to a specified protocol.

An important research challenge is to identify formal languages for specifying roles along with algorithms for determining whether a role conforms with another.

8 Conclusions

It should be no surprise to anyone that communication is at the heart of multiagent systems, not only in our implementations but also in our conception of what a multiagent system is and what an agent is.

To our thinking, an agent is inherently autonomous. Yet, autonomous, heterogeneously constructed agents must also be interdependent on each other if they are to exhibit complex behaviors and sustain important real-world applications. A multiagent system, if it is any good, must be loosely coupled and communica-

tion is the highly elastic glue that keeps it together. Specifically, communication, understood in terms of agents and based on high-level abstractions such as those we explained above, provides the quintessential basis for the arms-length relationships desired in all modern software engineering as it addresses the challenges of large decentralized systems.

The foregoing provided a historical view of agent communication, identifying the main historical and current ideas in the field. This chapter has only scratched the surface of this rich and exciting area. We invite the reader to delve deeper and to consider many of the fundamental research problems that arise in this area. An important side benefit is that, faced with the challenges of open systems such as on the web, in social media, in mobile computing, and cyberphysical systems, traditional computer science is now beginning to appreciate the importance and value of the abstractions of agent communication. Thus progress on the problems of agent communication can have significant impact on much of computer science.

Further Reading

Agent communication is one of the most interesting topics in multiagent systems, not only because of its importance to the field but also because of the large number of disciplines that it relates to. In particular, it touches upon ideas in philosophy, linguistics, social science (especially organizations and institutions), software engineering, and distributed computing. The readings below will take the reader deeper into these subjects.

Philosophical foundations. Some of the most important works on the philosophy of language undergird the present understanding of communication. Austin [6] introduced the idea of communication as action. Searle developed two accounts of communication, one emphasizing the mental concepts of the parties involved [41] and the second the notion of social reality that sustains and is sustained by language [42]. Some recent works by Chopra, Singh and their colleagues have exploited the distinction between constitution and regulation that Searle described [14, 38].

Organizations and institutions. Several researchers in multiagent systems have studied the notions of organizations and institutions. These works provide computational bases for agents to participate in structured relationships. The works of Vázquez-Salceda and the Dignum [3, 55] and of Fornara and Colombetti [27] highlight important conceptual and practical considerations in this area.

Norms, conventions, and commitments. The notions of organizations and institutions are defined based on the normative relationships that arise among

their participants. Artikis, Jones, Pitt, and Sergot have developed formalizations of norms that are worth studying as influential papers [5, 37]. Jones and Parent [36] formalize conventions as a basis for communication.

Singh proposed the notion of social commitments [43, 45] as an important normative concept to be used for understanding social relationships. He proposed commitments as a basis for a social semantics for communication [46]. A related idea has been developed by Colombetti [17]. A formal semantics for commitments [48] and the proper reasoning about commitments in situations with asynchronous communication among decoupled agents [15] are significant to practice and promising as points of departure for important research in this area.

Software engineering. A number of approaches apply communications as central to the development of multiagent systems [10, 16, 21, 29, 39]. Further, several design and verification tools for communication protocols and agent communication generally have been proposed [1, 2, 4, 23, 56, 59]. The development of well-principled tools is an important research direction because of their potential impact on computer science – if they could lead to the expanded deployment of multiagent systems.

Challenges. The agent communication manifesto is a collection of short essays by several researchers who seek to articulate the main challenges and directions in this area [12]. The reader should consult it before embarking on research in this area.

Acknowledgments

We have benefited from valuable discussions about agent communication with several colleagues, in particular, our coauthors on papers relating to agent communication: Matteo Baldoni, Cristina Baroglio, Nirmal Desai, Scott Gerard, Elisa Marengo, Viviana Patti, and Pinar Yolum.

Some parts of this chapter have appeared in previous works by the authors [16, 38].

Amit Chopra was supported by a Marie Curie Trentino award. Munindar Singh's effort was partly supported by the National Science Foundation under grant 0910868. His thinking on this subject has benefited from participation in the OOI Cyberinfrastructure program, which is funded by NSF contract OCE-0418967 with the Consortium for Ocean Leadership via the Joint Oceanographic Institutions.

9 Exercises

1. **Level 1** Which of the following statements are true?
 - (a) Communications are an important class of interactions because they support the autonomy of the parties involved.
 - (b) The three elements of a communicative act are locution, illocution, and perlocution.
 - (c) Unlike traditional settings, perlocutions provide the right basis for communicative acts in open, service-oriented settings.
 - (d) Unlike in a traditional, finite-state machine, the states of a commitment machine are specified using logic and each transition corresponds to the meaning of the message that labels the transition.
 - (e) In an open environment, two agents might sometimes need to combine their local observations in order to determine that a third agent is complying with its commitments.
2. **Level 1** Which of the following statements are true?
 - (a) In an open environment, we can typically ensure compliance based upon the implementations of the interacting agents.
 - (b) The benefit of employing a commitment protocol is that it exactly specifies the order of the messages without regard to their meaning.
 - (c) Using the meanings of the messages, we can compute whether a message may be sent in the current state, and the next state that would result from doing so.
 - (d) Ideally, each participant in a protocol should be able to verify if any of the commitments where it is the creditor are violated.
3. **Level 1** Which of the following statements are true about interaction and communication?
 - (a) Perlocutions are considered the core aspect of a communicative act.
 - (b) The same proposition, e.g., `reserve(Alice, UA 872, 14 May 2020)`, may feature in a *request* and a *declare*.
 - (c) We may not be able to decide if a statement such as *Shut the door!* is true or false but we can decide whether such a statement was made.
 - (d) A statement such as *Shut the door!* becomes true if the door in question is shut on *purpose*, not accidentally.

4. **Level 1** Identify all of the following statements that are true about commitments and commitment protocols.

- (a) If the debtor of a commitment delegates it simultaneously with the creditor of the same commitment assigning it, additional messages are in general needed for the new debtor and the new creditor to learn about each other.
- (b) If the debtor of a commitment discharges it simultaneously with the creditor of the same commitment assigning it, no additional messages are needed for the new creditor to learn that the debtor is compliant.
- (c) A protocol for payment through a third party could naturally be specified using the delegate of a commitment to pay.
- (d) Forward-going interactions such as ordering and payment may be modeled as commitment protocols, but not backward-going interactions such as returning goods for a refund.
- (e) Even though a commitment protocol captures the meanings of the messages involved, the participants must accept the protocol in order for it to work.

5. **Level 2** We say that a commitment is *discharged* when the consequent holds, *expired* when the antecedent cannot ever hold, and *violated* when the antecedent holds but the consequent cannot ever hold.

Let $\mathbf{E} = \{e_0, e_1, e_2, \dots, \bar{e}_0, \bar{e}_1, \bar{e}_2, \dots\}$ be a set of events such that \bar{e}_i is the complement of e_i . For instance, if e_0 means *package was delivered by 5PM*, \bar{e}_0 means *package was not delivered by 5PM*. Further, $\bar{\bar{e}}_0 = e_0$.

Let $\langle v_0, v_1, \dots, v_n \rangle$ represent an event trace, that is, the sequence of events that have been recorded, where all the v_i are variables that range over \mathbf{E} . Further, in any event trace, for any event, only the event or its complement may occur, but not both (e.g., the package was either delivered by 5PM or it was not, but not both). Thus, for example, $\langle e_0, e_3, \bar{e}_5 \rangle$ is a valid trace, but $\langle e_0, e_3, \bar{e}_5, \bar{e}_0 \rangle$ is not.

Assume that the commitment $C(x, y, e_0, e_1 \wedge e_2)$ holds right before we start recording events (x commits to y that if e_0 occurs, both e_1 and e_2 will occur).

For each of the following event traces, indicate whether the commitment is (1) satisfactorily resolved (via discharge or expiration), (2) violated, or (3) continues to hold.

- (a) $\langle e_0, e_1, e_5 \rangle$

- (b) $\langle \bar{e}_1, e_0, e_2 \rangle$
 - (c) $\langle e_1, \bar{e}_0, e_3 \rangle$
 - (d) $\langle e_1, e_0, e_2 \rangle$
 - (e) $\langle \bar{e}_0, \bar{e}_1, \bar{e}_2 \rangle$
6. **Level 2** Examine Figure 3.1. Now create an FSM for the commitment compensate pattern discussed in the chapter.
 7. **Level 2** Examine Figure 3.1. Now specify a commitment pattern that captures the idea of updating commitments.
 8. **Level 2** Create an FSM corresponding to the FIPA request protocol shown in Figure 3.2.
 9. **Level 3** Create a WS-CDL specification for the FIPA request protocol.
 10. **Level 3** Consider the following outline of a process for buying books. A merchant offers an online catalog of books with price and availability information. A customer can browse the catalog and purchase particular books from the catalog or the merchant may contact the customer directly with offers for particular books. However, the customer must arrange for shipment on his or her own: in other words, the customer must arrange for a shipper to pick up the books from the merchant's store and deliver them to him or her. All payments – to the merchant for the books and to the shipper for delivery – are carried out via a payment agency (such as PayPal).
 - (a) List the roles and messages involved in the protocol underlying the above business process.
 - (b) Specify the messages in terms of communicative acts.
 - (c) Specify the protocol in three different ways: as an FSM with messages as the transitions, (2) as an MSC, and (3) as a commitment protocol.
 - (d) Show a simplified MSC representing one possible enactment where the books have been delivered and the payments have been made.
 - (e) Based on the commitment protocol you specified above, annotate points in the above-described enactment with commitments that hold at those points.
 11. **Level 4** Suppose the business process described in Question 10 above also supported returns and refunds for customers.

- (a) As we did above, specify the underlying protocol as an FSM, as an MSC, and as a commitment protocol.
 - (b) Show both a synchronous and an asynchronous return-refund enactment.
 - (c) Annotate both with the commitments at various points. (Hint: for the asynchronous enactment, read [15]).
12. **Level 3** Specify role skeletons for the purchase process with returns and refunds
- (a) in the JADE style.
 - (b) in the rule-based style. (Hint: read [22])
13. **Level 3** Map Figure 3.6 to an FSM and an MSC.
14. **Level 3** Compare the FSM and MSC from Question 13 to the commitment protocol specification of Table 3.1 with respect to compliance, ease of creation, and ease of change.
15. **Level 4** Implement the logic for practical commitments described in [48].
16. **Level 5** Implement a commitment-based middleware based on the postulates given in [15].

References

- [1] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Web service contracting: Specification and reasoning with SCIFF. In *Proceedings of the 4th European Semantic Web Conference*, pages 68–83, 2007.
- [2] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Modeling interactions using social integrity constraints: A resource sharing case study. In *Proceedings of the International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 2990 of *LNAI*, pages 243–262. Springer, 2004.
- [3] Huib Aldewereld, Sergio Álvarez-Napagao, Frank Dignum, and Javier Vázquez-Salceda. Making norms concrete. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 807–814, Toronto, 2010. IFAAMAS.

- [4] Alexander Artikis, Marek J. Sergot, and Jeremy Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15):776–804, 2007.
- [5] Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1), 2009.
- [6] John L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.
- [7] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley-Blackwell, 2007.
- [8] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling ER 2004*, volume 3288 of *LNCS*, pages 524–541. Springer, 2004.
- [9] Carlos Canal, Lidia Fuentes, Ernesto Pimentel, José M. Troya, and Antonio Vallecillo. Adding roles to CORBA objects. *IEEE Transactions on Software Engineering*, 29(3):242–260, 2003.
- [10] Christopher Cheong and Michael P. Winikoff. Hermes: Designing flexible and robust agent interactions. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter 5, pages 105–139. IGI Global, Hershey, PA, 2009.
- [11] Amit Chopra and Munindar P. Singh. Nonmonotonic commitment machines. In Frank Dignum, editor, *Advances in Agent Communication: Proceedings of the 2003 AAMAS Workshop on Agent Communication Languages*, volume 2922 of *LNAI*, pages 183–200. Springer, 2004.
- [12] Amit K. Chopra, Alexander Artikis, Jamal Bentahar, Marco Colombetti, Frank Dignum, Nicoletta Fornara, Andrew J. I. Jones, Munindar P. Singh, and Pinar Yolum. Research directions in agent communication. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011.
- [13] Amit K. Chopra and Munindar P. Singh. Contextualizing commitment protocols. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1345–1352, 2006.
- [14] Amit K. Chopra and Munindar P. Singh. Constitutive interoperability. In *Proceedings of the 7th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 797–804, Estoril, Portugal, May 2008. IFAAMAS.
- [15] Amit K. Chopra and Munindar P. Singh. Multiagent commitment alignment. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 937–944, Budapest, May 2009. IFAAMAS.

- [16] Amit K. Chopra and Munindar P. Singh. Specifying and applying commitment-based business patterns. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Taipei, May 2011. IFAA-MAS.
- [17] Marco Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of the Autonomous Agents Workshop on Agent Languages and Communication Policies*, pages 21–29, May 2000.
- [18] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.
- [19] Nirmal Desai, Amit K. Chopra, Matthew Arrott, Bill Specht, and Munindar P. Singh. Engineering foreign exchange processes via commitment protocols. In *Proceedings of the 4th IEEE International Conference on Services Computing*, pages 514–521, Los Alamitos, 2007. IEEE Computer Society Press.
- [20] Nirmal Desai, Amit K. Chopra, and Munindar P. Singh. Representing and reasoning about commitments in business processes. In *Proceedings of the 22nd Conference on Artificial Intelligence*, pages 1328–1333, 2007.
- [21] Nirmal Desai, Amit K. Chopra, and Munindar P. Singh. Amoeba: A methodology for modeling and evolution of cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(2):6:1–6:45, October 2009.
- [22] Nirmal Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31(12):1015–1027, December 2005.
- [23] Nirmal Desai and Munindar P. Singh. On the enactability of business protocols. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 1126–1131, Chicago, July 2008. AAAI Press.
- [24] Hywel R. Dunn-Davies, Jim Cunningham, and Shamimabi Paurobally. Propositional statecharts for agent interaction protocols. *Electronic Notes in Theoretical Computer Science*, 134:55–75, 2005.
- [25] ebBP. Electronic business extensible markup language business process specification schema v2.0.4, December 2006. docs.oasis-open.org/ebxml-bp/2.0.4/OS/.
- [26] FIPA. FIPA interaction protocol specifications, 2003. FIPA: The Foundation for Intelligent Physical Agents, <http://www.fipa.org/repository/ips.html>.

- [27] Nicoletta Fornara, Francesco Viganò, Mario Verdicchio, and Marco Colombetti. Artificial institutions: A model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16(1):89–105, March 2008.
- [28] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, Reading, MA, 3rd edition, 2003.
- [29] Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenani, and Jorge Valenzuela. O-MaSE: A customizable approach to developing multiagent processes. In *Proceedings of the 8th International Workshop on Agent Oriented Software Engineering (AOSE)*, 2007.
- [30] Scott N. Gerard and Munindar P. Singh. Formalizing and verifying protocol refinements. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011.
- [31] HL7 reference information model, version 1.19. www.hl7.org/Library/data-model/RIM/C30119/Graphics/RIM_billboard.pdf, 2002.
- [32] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 273–284. ACM, 2008.
- [33] <http://www.gs1.org/productssolutions/gdsn/>. GDSN: Global Data Synchronization Network.
- [34] <http://www.hitsp.org/>. Hitsp: Healthcare information technology standards panel.
- [35] Marc-Philippe Huget and James Odell. Representing agent interaction protocols with agent UML. In *Agent-Oriented Software Engineering V*, volume 3382 of *LNCS*, pages 16–30. Springer, 2005.
- [36] Andrew J. I. Jones and Xavier Parent. A convention-based approach to agent communication languages. *Group Decision and Negotiation*, 16(2):101–141, March 2007.
- [37] Andrew J. I. Jones and Marek Sergot. A formal characterisation of institutionalized power. *Journal of the IGPL*, 4(3):429–445, 1996.
- [38] Elisa Marengo, Matteo Baldoni, Amit K. Chopra, Cristina Baroglio, Viviana Patti, and Munindar P. Singh. Commitments with regulations: Reasoning about safety and control in REGULA. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Taipei, May 2011. IFAA-MAS.
- [39] Lin Padgham and Michael Winikoff. Prometheus: A practical agent-oriented methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, chapter 5, pages 107–135. Idea Group, Hershey, PA, 2005.

- [40] RosettaNet. Home page, 1998. www.rosettanel.org.
- [41] John R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, UK, 1969.
- [42] John R. Searle. *The Construction of Social Reality*. Free Press, New York, 1995.
- [43] Munindar P. Singh. Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, pages 104–106, 1991.
- [44] Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, December 1998.
- [45] Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7(1):97–113, March 1999.
- [46] Munindar P. Singh. A social semantics for agent communication languages. In *Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 31–45, Berlin, 2000. Springer.
- [47] Munindar P. Singh. Distributed enactment of multiagent workflows: Temporal logic for service composition. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 907–914, Melbourne, July 2003. ACM Press.
- [48] Munindar P. Singh. Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 176–181, Chicago, July 2008. AAAI Press.
- [49] Munindar P. Singh. Information-driven interaction-oriented programming. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 491–498, Taipei, May 2011. IFAAMAS.
- [50] Munindar P. Singh. LoST: Local state transfer—An architectural style for the distributed enactment of business protocols. In *Proceedings of the 7th IEEE International Conference on Web Services (ICWS)*, pages 57–64, Washington, DC, 2011. IEEE Computer Society.
- [51] Munindar P. Singh, Amit K. Chopra, and Nirmal Desai. Commitment-based service-oriented architecture. *IEEE Computer*, 42(11):72–79, November 2009.
- [52] Munindar P. Singh, Amit K. Chopra, Nirmal Desai, and Ashok U. Mallya. Protocols for processes: Programming in the large for open systems. *ACM SIGPLAN Notices*, 39(12):73–83, December 2004.

- [53] Transaction workflow innovation standards team, February 2006. <http://www.twiststandards.org>.
- [54] Wil M. P. van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods*, volume 4184 of *LNCS*, pages 1–23. Springer, 2006.
- [55] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.
- [56] Michael Winikoff, Wei Liu, and James Harland. Enhancing commitment machines. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 3476 of *LNAI*, pages 198–220, Berlin, 2005. Springer-Verlag.
- [57] WS-CDL. Web services choreography description language version 1.0, November 2005. www.w3.org/TR/ws-cdl-10/.
- [58] Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.
- [59] Pinar Yolum. Design time analysis of multiagent protocols. *Data and Knowledge Engineering Journal*, 63:137–154, 2007.
- [60] Pinar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 527–534. ACM Press, July 2002.

Chapter 4

Negotiation and Bargaining

Shaheen Fatima and Iyad Rahwan

1 Introduction

Negotiation is a form of interaction in which a group of agents with conflicting interests try to come to a mutually acceptable agreement over some outcome. The outcome is typically represented in terms of the allocation of resources (commodities, services, time, money, CPU cycles, etc.). Agents' interests are conflicting in the sense that they cannot be simultaneously satisfied, either partially or fully. Since there are usually many different possible outcomes, negotiation can be seen as a “distributed search through a space of potential agreements” [36].

Negotiation is fundamental to distributed computing and multiagent systems. This is because agents often cannot fulfill their design objectives on their own, but instead need to exchange resources with others. After an informal discussion of the different aspects of negotiation problems (Section 2), we turn to the use of game theory to analyze strategic interaction in simple single-issue negotiation (Section 3). Next, we talk about game-theoretic analysis of multi-issue negotiation (Section 4).

After covering game-theoretic approaches, we describe various heuristic approaches for bilateral negotiation (Section 5). These approaches are necessary when negotiation involves solving computationally hard problems, or when assumptions underlying the game-theoretic approaches are violated. We also explore recent developments in agent-human negotiation (Section 6) and work on logic-based argumentation in negotiation.

We note that this chapter is concerned with the study of *bilateral negotiation* in multiagent systems; that is, negotiation involving two agents. Multiparty negotiation is often conducted under the banner of *auctions* and are outside the scope of this chapter.

2 Aspects of Negotiation

Any negotiation problem requires defining the following main ingredients: (i) the set of possible outcomes; (ii) the agents conducting the negotiation; (iii) the protocol according to which agents search for a specific agreement in this space; and (iv) the individual strategies that determine the agents' behavior, in light of their preferences over the outcomes.

The first ingredient in a negotiation scenario is the negotiation object, which defines the set of possible outcomes. Abstractly, we can simply think of a space \mathcal{O} of possible *outcomes* (or *deals* or *agreements*), which can be defined in arbitrary ways. There are many ways to define the set of possible outcomes concretely. The simplest possible way is a *single-issue* negotiation scenario, in which outcomes are described as members in discrete or continuous sets of outcomes. For example, two agents may be negotiating over how to divide a tank of petrol (or some other *resource*), and the question is who gets how much petrol. The set of possible outcomes may be represented as a number in the interval $[0, 1]$, each of which represents a percentage that goes to the first agent, with the rest going to the second agent. Petrol is an example of a *continuous* issue. Alternatively, the issue may be defined in terms of a set of *discrete* outcomes, such as a set of 8 time slots to allocate to multiple teachers.

Single-issue negotiation contrasts with *multi-issue* negotiation, in which the set of outcomes is defined in terms of multiple (possibly independent) issues. For example, two people may negotiate over both the time and place of dinner. Each of these represents one of the issues under negotiation, and an outcome is defined in terms of combinations of choices over these issues (i.e., a specific restaurant at a specific time). In general, given a set of issues (or attributes) A_1, \dots, A_n , each ranging over a (discrete or continuous) domain $Dom(A_i)$, then the space of possible outcomes is the Cartesian product $\prod_{i=1}^n Dom(A_i)$.

There are other approaches to defining the space of possible outcomes of negotiation. In their classic book, Rosenschein and Zlotkin distinguished between three different types of domains in terms of the nature of the negotiation object [66]:

1. *Task-oriented domains*: domains involving the division of tasks to execute; agent preferences are measured in terms of the costs associated with differ-

ent task allocations; each agent tries to minimize the cost of the tasks it has to execute.

2. *State-oriented domains*: domains involving a joint decision about what state agents will achieve; agent preferences are over states that result from different deals; each agent tries to get to a more preferable state for itself.
3. *Worth-oriented domains*: domains involving a joint decision about what goals to achieve; agent preferences are measured in terms of the number of individual goals each outcome achieves; each agent tries to achieve as many of its goals as possible.

In general, it may be possible to use any suitable approach to define the space of possible outcomes, including the use of expressive logical languages for describing combinatorial structures of negotiation objects.

The very nature of competition over resources means that different agents prefer different allocations of the resources in question. Hence, we need to capture the *individual agent preference* over the set Ψ of possible deals. Preferences of agent i can be captured using a binary *preference relation* \succeq_i over Ψ , and we denote by $o_1 \succeq_i o_2$ that for agent i , outcome o_1 is at least as good as outcome o_2 . It is also common to use $o_1 \succ_i o_2$ to denote that $o_1 \succeq_i o_2$ and it is not the case that $o_2 \succeq_i o_1$ (this is called *strict preference*). Economists and decision theorists consider a preference relation to be *rational* if it is both transitive and complete [53].

It is worth noting that agents may already have a particular allocation of resources before they begin negotiation. Negotiation becomes an attempt to *reallocate* the resources in order to reach a new allocation that is more preferable to both. In this case, the *conflict deal* (also known as the *no negotiation alternative*) refers to the situation in which agents do not reach an agreement in negotiation.

One way to define the preference relation of agent i is in terms of a *utility function* $U^i : \mathcal{O} \rightarrow \mathbb{R}^+$, which assigns a real number to each possible outcome. The utility function $U^i(\cdot)$ represents the relation \succeq_i if we have $U^i(o_1) \geq U^i(o_2)$ if and only if $o_1 \succeq_i o_2$.

In a sense, the utility function (and corresponding preference relation) captures the *level of satisfaction* of an agent with a particular deal. A *rational* agent attempts to reach a deal that *maximizes the utility* it receives.

In the case of multi-issue negotiation, it may be possible to define a *multi-attribute utility function*, $U^i : A_1 \times \cdots \times A_n \rightarrow \mathbb{R}^+$ which maps a vector of attribute values to a real number. And if the attributes are *preferentially independent*, then the utility function can be defined using a linear combination of sub-utility functions over the individual attributes. In other words, the utility of outcome

$o = \langle a_1, \dots, a_n \rangle$ becomes $U^i = \sum_{k=1}^n w_k^i u_k^i(a_k)$, where $u_k^i : A_k \rightarrow \mathbb{R}^+$ is the sub-utility function over attribute a_k and w_k^i is the attribute's weight.

Given a set of agents (and their preferences), we need a protocol (rules of interaction) for enabling these agents to search for a deal. These protocols range from very simple *alternating-offer* protocols [43, 69, 70, 74], in which agents take turns as they exchange possible deals, to complex *argumentation-based* protocols [46, 63], in which agents can exchange logical sentences intended to persuade one another to change their states of mind.

Given a set of agents, and a protocol for regulating their negotiation, the final ingredient is the agent's *strategy*. The strategy may specify, for example, what offer to make next, or what information to reveal (truthfully or otherwise) to the counterpart. A rational agent's strategy must aim to achieve the best possible outcome for itself, while still following the agreed protocol. The main role played by game theory in negotiation is the analysis of the final outcome in light of such strategic behavior.

3 Game-Theoretic Approaches for Single-Issue Negotiation

In this section, we will describe how game theory can be used to analyze negotiation. We will begin by considering the following scenario. There is a *single resource* and there are two agents competing for the resource. Each agent wants to get as large a share of the resource as possible, so there is a conflict between the agents with regard to how the resource must be divided between them. To resolve this conflict, the agents must *negotiate* or *bargain* and decide upon a division that will be acceptable to both parties. This decision is jointly made by the agents themselves, so each party can only obtain what the other is prepared to allow it. Given this, the negotiation will either end successfully, whereby the parties reach an agreement on a mutually acceptable split, or else it will end in a failure to reach an agreement. In the event of the latter happening, both agents get nothing. Hence, each agent will prefer to get a non-zero share than allow the negotiation to break down.

There are two ways to model such bilateral negotiations: using *cooperative* game theory and using *non-cooperative* game theory. For the former, negotiation is modeled as a cooperative game. In cooperative games, agreements are binding or enforceable, possibly by law. When agreements are binding, it is possible for the players to negotiate outcomes that are mutually beneficial. In non-cooperative games, agreements are not binding. Here, the players are self-interested and their focus is on *individually beneficial* outcomes. So a player may have an incentive

		a	
		deny	confess
b	deny	$-1, -1$	$-3, 0$
	confess	$0, -3$	$-2, -2$

Table 4.1: Prisoner's Dilemma game.

to deviate from an agreement in order to improve its utility. Thus the outcome of a game when agreements are binding may be different from the outcome of the same game when they are not [7]. This difference can be illustrated with the *Prisoner's Dilemma* game given in Table 4.1. Assume that this game is non-cooperative. Then the dominant strategy for both players will be to confess. The equilibrium outcome would be $(-2, -2)$ which is not Pareto optimal. In contrast, if the same game was played as a cooperative game, and the players agreed not to confess, then both players would benefit. The agreement (deny, deny) would be binding and the resulting outcome $(-1, -1)$ would be Pareto optimal. This outcome would also be better from each individual player's perspective as it would increase each player's utility from -2 to -1 .

Note that the outcome of a game with binding agreements need not necessarily be better than the outcome for the same game with agreements that are not binding. The former are aimed at reaching agreements that are a *reasonable compromise*. However, from an individual player's perspective, the latter may be better than the former. In more detail, cooperative and non-cooperative bargaining is modeled as follows.

3.1 Cooperative Models of Single-Issue Negotiation

In this section, we will explain how bargaining can be modeled as a two-person cooperative game. A two-person bargaining situation typically involves two individuals who have the opportunity to collaborate for mutual benefit. Furthermore, there will be more than one way of collaborating, and how much an individual benefits depends on the actions taken by both agents. In other words, neither individual can unilaterally affect these benefits. Thus, situations such as those involving trading between a buyer and a seller, and those involving negotiation between an employer and a labor union may be regarded as bargaining problems.

Most of the work on cooperative models of bargaining followed from the seminal work of Nash [55, 56]. Nash analyzed the *bargaining problem* and defined a *solution/outcome* for it using an *axiomatic approach*. A solution means a determination of how much each individual should expect to benefit from the situation.

Nash defined a solution without modeling the details of the negotiation process. In this approach, there is a set of *possible* or *feasible* outcomes, some of which are *acceptable* or *reasonable* outcomes. The problem then is to find a *bargaining function* that maps the set of possible outcomes to the set of acceptable ones.

Nash idealized the bargaining problem by assuming that the two individuals are *perfectly rational*, that each can accurately compare its preferences for the possible outcomes, that they are equal in bargaining skill, and that each has complete knowledge of the preferences of the other. Under these assumptions, Nash formed a mathematical model of the situation. In this model, he employed *numerical utilities* to express the preferences of each individual, and each individual's desire to maximize its own gain.

More formally, Nash defined a two-person bargaining problem as follows. There are two players (say a and b) who want to come to an agreement over the alternatives in an arbitrary set \mathcal{A} . Failure to reach an agreement, i.e., disagreement, is represented by a designated outcome denoted $\{D\}$. Agent $i \in \{a, b\}$ has a von Neumann-Morgenstern utility function U^i defined as follows:

$$U^i : \{\mathcal{A} \cup \{D\}\} \rightarrow \mathbb{R} \quad \text{for } i = a, b$$

The set of all utility pairs that result from an agreement is called the *bargaining set*. This set is denoted \mathcal{S} where

$$\mathcal{S} = \{(U^a(z), U^b(z)) \in \mathbb{R}^2 : z \in \mathcal{A}\}$$

and the utility pair that results from disagreement is denoted $d = (d^a, d^b)$, where $d^i = U^i(D)$. The point $d \in \mathbb{R}^2$ is called the *disagreement point* or *threat point*. Thus, if the players reach an agreement $z \in \mathcal{A}$, then a gets a utility of $U^a(z)$ and b gets $U^b(z)$. But if they do not reach an agreement, then the game ends in the disagreement point d , where a gets utility d^a and b gets d^b . Given this, the bargaining problem is defined as follows:

Definition 4.1 A bargaining problem is defined as a pair (\mathcal{S}, d) . A bargaining solution is a function f that maps every bargaining problem (\mathcal{S}, d) to an outcome in \mathcal{S} , i.e.,

$$f : (\mathcal{S}, d) \rightarrow \mathcal{S}$$

Thus the *solution* to a bargaining problem is a pair in \mathbb{R}^2 . It gives the values of the game to the two players and is generated through the function called *bargaining function*. The values of the game to the two players are denoted f^a and f^b .

It is assumed that the set \mathcal{S} is a closed and convex subset of \mathbb{R}^2 , that $d \in \mathcal{S}$, and that the set

$$\mathcal{S} \cap \{(x^a, x^b) \mid x^a \geq d^a \text{ and } x^b \geq d^b\}$$

is non-empty and bounded. The assumption that \mathcal{S} is convex is the same as assuming that players can agree on jointly randomized strategies, such that, if the utility allocations $x = (x^a, x^b)$ and $y = (y^a, y^b)$ are feasible and $0 \leq \theta \leq 1$, then the expected utility allocation $\theta x + (1 - \theta)y$ can be achieved by planning to implement x with probability θ and y otherwise. Closure of \mathcal{S} is a natural topological requirement. The non-emptiness and boundedness condition means that not all feasible allocations are worse than disagreement for both players, and unbounded gains over the disagreement point are impossible.

The bargaining problem is solved by stating general properties (or axioms) that a *reasonable* solution should possess. By specifying enough such properties one can exclude all but one *solution*. For example, a reasonable solution must be *individual rational*, i.e., it must give each player at least as much utility as it would get in the event of no agreement. So individual rationality is an axiom. The term “reasonable solution” has no standard definition. Different axiomatic models define this term differently [68]. Nash’s [55] idea of a reasonable solution is based on the assumption that when two players negotiate or an impartial arbitrator arbitrates, the payoff allocations that the two players ultimately get should depend only on the following two factors:

1. the set of payoff allocations that are jointly feasible for the two players in the process of negotiation or arbitration, and
2. the payoffs they would expect if negotiation or arbitration were to fail to reach a settlement.

Based on these assumptions, Nash generated a list of axioms that a reasonable solution ought to satisfy. These axioms are as follows:

Axiom 1 (Individual Rationality) This axiom asserts that the bargaining solution should give neither player less than what it would get from disagreement, i.e., $f(\mathcal{S}, d) \geq d$.

Axiom 2 (Symmetry) As per this axiom, the solution should be independent of the names of the players, i.e., who is named a and who is named b . This means that when the players’ utility functions and their disagreement utilities are the same, they receive equal shares. So any asymmetries in the final payoff should only be due to the differences in their utility functions or their disagreement outcomes.

Axiom 3 (Strong Efficiency) This axiom asserts that the bargaining solution should be feasible and Pareto optimal.

Axiom 4 (Invariance) According to this axiom, the solution should not change as a result of linear changes to the utility of either player. So, for example, if a player's utility function is multiplied by 2, this should not change the solution. Only the player will value what it gets twice as much.

Axiom 5 (Independence of Irrelevant Alternatives) This axiom asserts that eliminating feasible alternatives (other than the disagreement point) that would not have been chosen should not affect the solution, i.e., for any closed convex set S' , if $S' \subseteq S$ and $f(S, d) \in S'$, then $f(S', d) = f(S, d)$.

Nash proved that the bargaining solution that satisfies the above five axioms is given by:

$$f(S, d) \in \operatorname{argmax}_{x \in S, x \geq d} (x^a - d^a)(x^b - d^b)$$

and that such a solution is *unique*.

Recall that a two-person bargaining problem was defined in terms of a threat point d . This is called the *fixed threat* bargaining model. A salient feature of the threat point d is that the players will receive the payoffs $d = (d^a, d^b)$ if they fail to reach an agreement. This means that there is no way that one player can take unilateral action that hurts the other. However, one can easily imagine bargaining scenarios in which each player has a range of possible actions to take in the event of absence of an agreement. Each player can choose an action from a set of possible actions and each player's choice of action affects both of them. Such a scenario is called a *variable threat* scenario.

In more detail, a bargaining situation is said to allow variable threats if each player can choose any one of several retaliatory strategies available to it, and can commit itself to use this strategy, called its *threat strategy*, against the other player if they cannot reach an agreement. Therefore, the payoffs the players would receive in such a conflict situation would depend on the threat strategies to which they had chosen to commit themselves. In contrast, a given bargaining situation is said to allow only fixed threats if the payoffs that the players would receive in the absence of an agreement are determined by the nature of the bargaining situation itself, instead of being determined by the players' choice of threat strategies or by any other actions the players may take. The fixed threats case is relevant mainly to economic situations, such as buying and selling; in the event of no agreement, there is simply "no deal." The variable threats case is more relevant to military situations, where a "failure to reach agreement" can have a wide range of possible outcomes.

In order to deal with variable threat scenarios, Nash [56] extended his solution for games with a fixed threat point. The Nash variable threat game has a unique solution, as is the case with the Nash fixed threat game. However, in contrast to

the Nash fixed threat game, the Nash variable threat game does not, in general, guarantee existence of a solution.

Following Nash's work, several other bargaining solution concepts were proposed using other systems of axioms [68]. Nash's work was also extended to bargaining with incomplete information [31, 76]. In general, for these axiomatic models of bargaining, the solution depends only on two factors: the set of possible agreements and the disagreement point. However, in many practical scenarios, the outcome of bargaining depends on other factors, such as the tactics employed by the bargainers, the procedure through which negotiation is conducted, and the players' information. Non-cooperative models of bargaining [69, 70, 77] incorporate these factors.

3.2 Non-Cooperative Models of Single-Issue Negotiation

In the axiomatic model, a bargaining problem was defined as a pair (S, d) but no *bargaining procedure* or *protocol* was specified. A key difference between the cooperative and non-cooperative models is that the former does not specify a procedure, whereas the latter does. Perhaps the most influential non-cooperative model is that of Rubinstein [69, 70]. He provided a basic framework that could easily be adapted and extended for different applications. He started with a simple complete information model [69] and then extended it to an incomplete information case [70]. The following paragraphs provide a brief overview and some key insights of Rubinstein's work.

In [69], Rubinstein considered the following scenario. There are two players and a unit of good, a pie, to be split between them. If player a gets a share of $x^a \in [0, 1]$, then player b gets $x^b = 1 - x^a$. Neither player receives anything unless the two players come to an agreement. Here, the issue, or the pie, can be split between the players. So the issue is said to be *divisible*.

This is a *strategic form* game and is played over a series of discrete time periods $t = 1, 2, \dots$. In the first time period, one of the players, say player a , begins by proposing an offer to player b . Player b can accept or reject the offer. If player b accepts, the game ends successfully with the pie being split as per player a 's proposal. Otherwise, the game continues to the next time period in which player b proposes a counteroffer to player a . This process of making offers and counteroffers continues until one of the players accepts the other's offer. Prior to termination, player a makes offers in all odd-numbered time periods $t = 1, 3, \dots$, and accepts or rejects player b 's offers in all even-numbered time periods $t = 2, 4, \dots$. Player b makes offers in all even-numbered time periods and accepts or rejects player a 's offer in all odd-numbered time periods. Since the players take turns in making offers, this is known as *alternating offers protocol*. In this protocol, there are no rules that prescribe how a player's offer should relate to any previous or

succeeding offers (this is in contrast to the monotonic concession protocol [67] that prescribes a relation on a current offer and the previous one).

Here, the utility of an outcome to a player depends on its share and on the time period in which an agreement is reached. The utility is increasing in the player's share and decreasing in time. The decrease in utility with time may be attributed to the object under negotiation being perishable, or to inflation. This decrease in utility with time is modeled with a discount factor. Let δ_a and δ_b denote the discount factor for a and b , respectively. If a and b receive a share of x^a and x^b respectively where $x^a + x^b = 1$, then their utilities at time t are as follows:

$$U^a = x^a \delta_a^{t-1} \quad \text{and} \quad U^b = x^b \delta_b^{t-1}.$$

If this discounted game is played infinitely over time, then Rubinstein [69] showed that there is a *unique* perfect equilibrium outcome in which the players' immediately reach an agreement on the following shares:

$$x^a = \frac{1 - \delta_b}{1 - \delta_a \delta_b} \quad \text{and} \quad x^b = \frac{\delta_b - \delta_a \delta_b}{1 - \delta_a \delta_b}$$

where player a is the first mover. The properties of *uniqueness* and *immediate agreement* are especially desirable in the context of agents [43, 45]. However, this infinite horizon model may not be immediately applicable to multiagent systems since agents are typically constrained by a deadline (we will deal with deadlines shortly).

Although Rubinstein's model may not be directly applicable to the design of automated negotiating agents, it provides two key intuitive insights. First, in frictionless¹ bargaining, there is nothing to prevent the players from haggling for as long as they wish. It seems intuitive that the cost of haggling serves as an incentive for the players to reach an agreement. Second, a player's bargaining power depends on the relative magnitude of the players' respective costs of haggling. The absolute magnitudes of these costs are irrelevant to the bargaining outcome.

It is now clear how the discount factor can influence negotiation. Apart from the discount factor, a *deadline* can also impact on negotiation. Deadlines are important because, in many applications [37], the agents must reach an agreement within a time limit. So let us now study negotiations that are constrained by both a discount factor and a deadline.

Work on negotiation with deadlines and discount factors includes [18, 25, 73]. While Fatima et al. [18] and Gatti et al. [25] use the alternating offers protocol, Sandholm et al. [73] use a simultaneous offers protocol. Also, while [18] considers negotiation over a pie, [25] considers the players' reserve prices. Since the

¹If it does not cost the players anything to make offers and counteroffers, the bargaining process may be considered frictionless.

model in [18] resembles Rubinstein's model (described above) more closely, we will study this one in detail.

In [18], δ is the discount factor for both players. And n , where n denotes a positive integer, is the deadline. Thus, negotiation must end by the time period $t = n$. Otherwise, both players get zero utilities. For time $t \leq n$, the utility functions are defined as follows. If a gets a share of x^a and b gets x^b where $x^a + x^b = 1$, then $U^a = x^a \delta^{t-1}$ and $U^b = x^b \delta^{t-1}$. As before, the negotiation is conducted as per the alternating offers protocol.

Here, under the complete information assumption, the equilibrium offer for a time $t \leq n$ can be obtained using *backward induction* as follows. Let $x^a(t)$ and $x^b(t)$ denote a 's and b 's equilibrium shares, respectively, for a time period t . Consider the last time period $t = n$ and assume that a is the offering agent. Since the deadline is n and the pie shrinks with time, it will be optimal for a to propose to keep the entire shrunken pie and give nothing to b . Because of the deadline, b will accept such an offer.² If we let $\text{STRATA}(t)$ and $\text{STRATB}(t)$ denote a 's and b 's equilibrium strategies for time t , then the following strategies will form a subgame perfect equilibrium. For $t = n$, these strategies are:

$$\begin{aligned} \text{STRATA}(n) &= \begin{cases} \text{OFFER } (\delta^{n-1}, 0) & \text{If } a\text{'s turn to offer} \\ \text{ACCEPT} & \text{If } b\text{'s turn to offer} \end{cases} \\ \text{STRATB}(n) &= \begin{cases} \text{OFFER } (0, \delta^{n-1}) & \text{If } b\text{'s turn to offer} \\ \text{ACCEPT} & \text{If } a\text{'s turn to offer} \end{cases} \end{aligned}$$

For all preceding time periods, the equilibrium strategies are obtained as follows. Consider a time $t < n$ and let a be the offering agent. Agent a will propose an offer such that b 's utility from it will be equal to what b would get from its own offer for the next time period. Agent b will accept such an offer. So for $t < n$, the following strategies will form a subgame perfect equilibrium:

$$\begin{aligned} \text{STRATA}(t) &= \begin{cases} \text{OFFER } (\delta^{t-1} - x^b(t+1), x^b(t+1)) & \text{If } a\text{'s turn to offer} \\ \text{If } (U^a(x^a, t) \geq U^a(t+1)) & \text{If } a \text{ receives } (x^a, x^b) \\ \text{ACCEPT} \\ \text{else REJECT} \end{cases} \\ \text{STRATB}(t) &= \begin{cases} \text{OFFER } (x^a(t+1), \delta^{t-1} - x^a(t+1)) & \text{If } b\text{'s turn to offer} \\ \text{If } (U^b(x^b, t) \geq U^b(t+1)) & \text{If } b \text{ receives } (x^a, x^b) \\ \text{ACCEPT} \\ \text{else REJECT} \end{cases} \end{aligned}$$

²It is possible that b may reject the offer. In practice, a will have to propose an offer that will be just enough to induce b to accept.

where $UA(t)$ ($UB(t)$) denotes a 's (b 's) equilibrium utility for time t . An agreement takes place at $t = 1$.

The above model was also analyzed in [18] in an incomplete information setting with uncertainty about utility functions. In contrast, [25, 73] consider uncertainty over the negotiation deadline. Also, as we mentioned earlier, a key difference between [18] and [73] is that the former uses the alternating offers protocol, while the latter uses a simultaneous offers protocol and treats time as a continuous variable. These differences in the setting result in the following differences in the outcomes. First, for the former, an agreement can occur in the first time period and the entire surplus does not necessarily go to just one of the players. For the latter, an agreement only occurs at the earlier deadline and the entire surplus goes to the player with the later deadline (different players have different deadlines). Second, unlike the former, the deadline effect in the latter completely overrides the effect of the discount factor. For the former, an agent's share of the surplus depends on both the deadline and the discount factor. These differences show that the protocol is a key determinant of the outcome of negotiation.

However, apart from the protocol, the parameters of negotiation (such as the deadline and the discount factor) also influence an agent's share. For the alternating offers protocol, [19] shows how the deadline and discount factor affects an agent's share.

Before closing this section, we will provide a brief overview of some of the key approaches for analyzing games with *incomplete information*. Incomplete information games are those where either or both players are uncertain about some parameters of the game, such as the utility functions, the strategies available to the players, the discount factors, etc. Furthermore, for sequential games, such as the alternating offers game described in Section 3.2, the players may acquire new information, and so, their information may change during the course of play.

There is a widely used approach for dealing with such incomplete information cases. This approach was originated by Harsanyi [28, 29, 30] in the context of simultaneous move games. In this approach, a player is assumed to have beliefs, in the form of a random variable, about an uncertain parameter. Thus, there is a *set of possible values* for the parameter, and a *probability distribution* over these possible values. And the uncertain parameter is determined by the realization of this random variable. Although the random variable's actual realization is observed only by the player, its ex-ante probability distribution is assumed to be common knowledge to the players. Such a game is called a *Bayesian game* and the related equilibrium notion is *Bayesian Nash equilibrium*.

Although a Bayesian game deals with incomplete information, it is a simultaneous moves game. However, most multiagent negotiations require agents to choose actions over time. *Dynamic games* [53] are a way of modeling such nego-

tiations. An example of a dynamic game is the alternating offers game described in Section 3.2 where the players take turns in making offers. In dynamic games, the uncertainty in information may be *one-sided* [70] (meaning that one of the two players has complete information, while the other is uncertain about some negotiation parameters), or it may be *two-sided* [8, 9] (meaning that both players are uncertain about some parameters). Depending on the type of uncertainty, a range of refinements of Nash equilibrium have been defined for dynamic games. These include *subgame perfect equilibrium* and *sequential equilibrium* [53].

Having looked at both the axiomatic and the non-cooperative models, let us now examine the similarities and differences between them. A key similarity is that, in both cases, there is a degree of conflict between the agents' interest, but there is also room for them to cooperate by resolving the conflict. Another similarity is between the solutions for the Nash bargaining model and Rubinstein's model: as the discount factor approaches 1 (i.e., as the players become more patient), the solution to the latter converges to the solution to the former. In this solution, the pie is split almost equally between the players.

A key difference between these two models is the way in which players are modeled and the way in which cooperation is enforced. In non-cooperative game theory, the basic modeling unit is the individual, and cooperation between individuals is self-enforcing. In contrast, in cooperative game theory the basic modeling unit is the group, and players can enforce cooperation in the group through a third party.

Another difference is that in a non-cooperative game, each player independently chooses its strategy. But in a cooperative game, the players are allowed to communicate before choosing their strategies and playing the game. They can agree but also disagree about a joint strategy. In the context of the Prisoner's Dilemma game, the players first communicate to decide on a joint strategy. Assume that the joint strategy they choose is (deny, deny). The players then play this strategy and each player is guaranteed (by means of a third party that enforces agreement) a utility of -1 .

Yet another difference between cooperative and non-cooperative models is that for the former, there is no element of time but for the latter there is. A non-cooperative bargaining game is played over a series of time periods. In general, the outcome for non-cooperative bargaining is determined through a process in which players make offers and counteroffers. This process is modeled as an alternating offers game in which the players take turns in making offers.

4 Game-Theoretic Approaches for Multi-Issue Negotiation

One of the key differences between single and multi-issue bargaining is that multiple issues can be bargained using one of several different procedures. A procedure or protocol specifies the rules for negotiating the issues. For example, a set of issues may be negotiated together or one by one. Furthermore, the outcome of a negotiation need not be independent of the procedure. It is necessary to understand these procedures in order to interpret the results of both cooperative (i.e., axiomatic) and non-cooperative models of multi-issue bargaining. So we will first outline these procedures, and then describe some of the key results from the axiomatic and the non-cooperative models. The following are the four key procedures for bargaining over multiple issues [58].

1. Global bargaining: Here, the bargaining agents directly tackle the global problem in which all the issues are addressed at once. In the context of non-cooperative theory, the global bargaining procedure is also called the *package deal procedure*. In this procedure, an offer from one agent to the other would specify how each one of the issues is to be resolved.
2. Independent/separate bargaining: Here negotiations over the individual issues are totally separate and independent, with each having no effect on the other. This would be the case if each of the two parties employed m agents (for negotiating over m issues), with each agent in charge of negotiating one issue. For example, in negotiations between two countries, each issue may be resolved by representatives from the countries who care only about their individual issue.
3. Sequential bargaining with independent implementation: Here the two parties consider one issue at a time. For instance, they may negotiate over the first issue, and after reaching an agreement on it, move on to negotiate the second, and so on. Here, the parties may not negotiate an issue until the previous one is resolved. There are several forms of the sequential procedure. These are defined in terms of the *agenda* and the *implementation rule*. For sequential bargaining, the agenda³ specifies the order in which the issues will be bargained. The implementation rule specifies when an agreement on an individual issue goes into effect. There are two implementation rules that have been studied in the literature [58]: the rule of *independent implementation* and the rule of *simultaneous implementation*. For the former, an

³As we will see in Section 5.3, the term “agenda” is defined differently in the context of the package deal procedure.

agreement on an issue goes into effect immediately (i.e., before negotiation begins on the next issue). The latter is as described below.

4. Sequential bargaining with simultaneous implementation: This is similar to the previous case except that now an agreement on an issue does not take effect until an agreement is reached on all the subsequent issues.

The dependence of the outcome of bargaining on the procedure has been recognized in the context of both axiomatic and non-cooperative models. We will describe some of the key results of the former in Section 4.1 and of the latter in Section 4.2.

4.1 Cooperative Models of Multi-Issue Negotiation

A lot of the work on cooperative models of multi-issue bargaining has dealt with proposing axioms that relate the outcomes of the above procedures. This includes work by Myerson [54] and by Ponsati and Watson [58]. While [54] deals with the first three of the above-listed four procedures, [58] deals with all four of them. In more detail, for a set of axioms, [58] defined a range of solutions to a multi-issue bargaining problem. In addition to the efficiency, invariance, and symmetry axioms, these include the following:

1. Simultaneous implementation agenda independence: This axiom states that global bargaining and sequential bargaining with simultaneous implementation yield the same agreement.
2. Independent implementation agenda independence: This axiom states that global bargaining and sequential bargaining with independent implementation yield the same agreement.
3. Separate/global equivalence: This axiom states that global bargaining and separate bargaining yield the same agreement.

Different axioms characterize different solutions. Details on these solutions can be found in [58].

4.2 Non-Cooperative Models of Multi-Issue Negotiation

We will begin by looking at the package deal procedure in a complete information setting. The package deal procedure is similar to the alternating offers protocol in that the parties take turns in making offers. However, here, an offer must include a share for each issue under negotiation.

The strategic behavior for the package deal procedure was analyzed in [18]. Below, we will describe this model in the context of the complete information setting (see [18] for details regarding strategic behavior in an incomplete information setting). Here, a and b negotiate over $m > 1$ divisible issues. These issues are m distinct pies and the agents want to determine how to split each of them. The set $S = \{1, 2, \dots, m\}$ denotes the set of m pies. As before, each pie is of size 1. For both agents, the discount factor is δ for all the issues (in [18], the discount factor is different for different issues, but for ease of discussion we will let it be the same for all the issues). For each issue, n denotes each agent's deadline.

In the offer for time period t (where $1 \leq t \leq n$), agent a 's (b 's) share for each of the m issues is represented as an m element vector x^a (x^b) such that, for $i \leq i \leq m$, $0 \leq x_i^a \leq 1$ and $x_i^a + x_i^b = 1$. Thus, if agent a 's share for issue c at time t is x_c^a , then agent b 's share is $x_c^b = (1 - x_c^a)$. The shares for a and b are together represented as the package (x^a, x^b) .

An agent's cumulative utility is linear and additive [41]. The functions U^a and U^b give the cumulative utilities for a and b respectively at time t and are defined as follows:

$$U^a((x^a, x^b), t) = \begin{cases} \sum_{c=1}^m w_c^a \delta^{t-1} x_c^a & \text{if } t \leq n \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

$$U^b((x^a, x^b), t) = \begin{cases} \sum_{c=1}^m w_c^b \delta^{t-1} x_c^b & \text{if } t \leq n \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

where $w^a \in \mathbb{R}_m^+$ denotes an m element vector of constants for agent a and $w^b \in \mathbb{R}_m^+$ such a vector for b . These vectors indicate how the agents prefer different issues. For example, if $w_c^a > w_{c+1}^a$, then agent a values issue c more than issue $c + 1$. Likewise for agent b .

It is clear from the above definition of utility functions that the parties may have different preferences over the issue. So, during the process of negotiation, it might be possible for an agent to perform trade-offs across the issues to improve its utility. Since the utilities are linear, the problem of making *trade-offs* becomes computationally tractable.⁴

For this setting, let us see how we can determine an equilibrium for the package deal procedure. Since there is a deadline, we can find an equilibrium using *backward induction* (as was done for single-issue negotiation). However, since an

⁴For a non-linear utility function, an agent's trade-off problem becomes a *non-linear optimization problem*. Due to the computational complexity of such an optimization problem, a solution can only be found using *approximation methods* [4, 32, 42]. Moreover, these methods are not general in that they depend on how the cumulative utilities are actually defined. Since we use linear utilities, the trade-off problem will be a linear optimization problem, the *exact* solution to which can be found in polynomial time.

offer for the package deal must include a share for all the m issues, an agent must now make trade-offs across the issues in order to maximize its cumulative utility. For a time t , agent a 's trade-off problem (TA(t)) is to find an allocation (x^a, x^b) that solves the following optimization problem:

$$\begin{aligned} &\text{Maximize} && \sum_{c=1}^m w_c^a x_c^a \\ &\text{subject to} && \delta^{t-1} \sum_{c=1}^m w_c^b (1 - x_c^a) \geq \text{UB}(t+1) \\ &&& 0 \leq x_c^a \leq 1 \quad \text{for } 1 \leq c \leq m \end{aligned} \quad (4.3)$$

The problem TA(t) is nothing but the well-known *fractional knapsack* problem.⁵ Let SA(TA(t)) denote a solution to TA(t). For agent b , TB(t) and SB(TB(t)) are analogous. Given this, Theorem 4.1 (taken from [18]) characterizes an equilibrium for the package deal procedure. Here, STRATA(t) (STRATB(t)) denotes a 's (b 's) equilibrium strategy for time t .

Theorem 4.1 *For the package deal procedure, the following strategies form a subgame perfect equilibrium. The equilibrium strategy for $t = n$ is:*

$$\begin{aligned} \text{STRATA}(n) &= \begin{cases} \text{OFFER } (\delta^{n-1}, \mathbf{0}) & \text{If } a\text{'s turn to offer} \\ \text{ACCEPT} & \text{If } b\text{'s turn to offer} \end{cases} \\ \text{STRATB}(n) &= \begin{cases} \text{OFFER } (\mathbf{0}, \delta^{n-1}) & \text{If } b\text{'s turn to offer} \\ \text{ACCEPT} & \text{If } a\text{'s turn to offer} \end{cases} \end{aligned}$$

where $\mathbf{0}$ is a vector of m zeroes. For all preceding time periods $t < n$, the strategies are defined as follows:

$$\begin{aligned} \text{STRATA}(t) &= \begin{cases} \text{OFFER SA(TA}(t)) & \text{If } a\text{'s turn to offer} \\ \text{If } (U^a(x^a, t) \geq \text{UA}(t+1)) & \text{If } a \text{ receives an offer } (x^a, x^b) \\ \text{ACCEPT} & \\ \text{else REJECT} & \end{cases} \\ \text{STRATB}(t) &= \begin{cases} \text{OFFER SB(TB}(t)) & \text{If } b\text{'s turn to offer} \\ \text{If } (U^b(x^b, t) \geq \text{UB}(t+1)) & \text{If } b \text{ receives an offer } (x^a, x^b) \\ \text{ACCEPT} & \\ \text{else REJECT} & \end{cases} \end{aligned}$$

where UA(t) (UB(t)) denotes a 's (b 's) equilibrium utility for time t . An agreement takes place at $t = 1$.

⁵Note that if x_c^a is allowed to take one of two possible values, zero or one, then the trade-off problem becomes an *integer knapsack problem*, which is NP-complete. Such a trade-off problem corresponds to negotiation over *indivisible* issues. Here, an issue must be allocated in its entirety to one of the two agents. So the problem is that of determining who to allocate which issue.

One can easily verify that the outcome of the package deal procedure will be Pareto optimal.

For the separate procedure, the m issues are negotiated independently of each other. So the equilibrium for the individual issues will be the same as that for single-issue negotiation.

For the sequential procedure with independent implementation, the m issues are negotiated sequentially one after another. But since the agreement on an issue goes into effect immediately, the equilibrium for the individual issues can be obtained in the same way as that for single-issue negotiation. An issue will be negotiated only after the previous one is settled.

For the sequential procedure with simultaneous implementation, the m issues are negotiated sequentially one after another. But since the agreement on an issue goes into effect only after all the m issues are settled, the basic idea for obtaining an equilibrium for this procedure will be the same as that for the package deal procedure.

Busch and Horstman [6] consider two issues and show how the outcome for sequential negotiation with independent implementation can differ from the outcome for simultaneous negotiations.

For the sequential procedure with independent implementation, a key determinant of the outcome of negotiation is the *agenda*. In the context of this procedure, the term agenda means the order in which the issues are settled. The importance of the agenda was first recognized by Schelling [75]. This initiated research on agendas for multi-issue negotiation. The existing literature on agendas can broadly be divided into two types: those that treat the agenda as an *endogenous* variable [3, 33, 34], and those that treat it as an *exogenous* [20] variable.

The difference between these two types of agendas is that, for the former, an agenda is selected during the process of negotiating over the issues. For the latter, an agenda is decided first, and then the parties negotiate over the issues on the agenda.

Research on endogenous agendas includes [3, 33, 34]. Bac and Raff [3] deal with two divisible issues in the context of an incomplete information setting. Here, the uncertainty is about the discount factor. For this setting, they study the properties of resulting equilibrium. While [3] deals with two divisible issues, Inderst [34] deals with multiple divisible issues and potentially infinite discrete time periods. Here, the parties are allowed to make an offer on any subset of a given set of issues but can only accept/reject a complete offer. In this sense, an agenda is selected “endogenously.” For this setting, he showed existence and uniqueness conditions for equilibrium under the complete information assumption. In and Serrano [33] generalized these results to a larger class of utility functions by considering a complete information setting. Similar work in the context of an

incomplete information setting was later dealt with by Fatima et al. in [17] .

Fershtman [20] dealt with exogenous agendas. For the complete information setting, he considered two divisible issues, and showed how the order in which they are negotiated affects the outcome.

5 Heuristic Approaches for Multi-Issue Negotiation

The heuristic approach is another approach for the design of negotiating agents. This approach is particularly useful when there are multiple issues to negotiate, and finding an equilibrium offer is computationally hard. As we saw in Section 4.2, finding an equilibrium offer for multi-issue negotiation requires solving a *trade-off* problem, which can be computationally hard. In order to overcome the computational hardness of the trade-off problem, heuristic strategies can be used instead of equilibrium ones.

Heuristics can be used in a number of ways. For example, they can be used for generating counteroffers. Apart from counteroffer generation, heuristics can also be used to predict information about the opponent. This approach is particularly relevant to situations where the negotiators have limited information about each other. Yet another use of heuristics is in the generation of *negotiation agendas*. The agenda is a key determinant of the outcome of a negotiation. So finding the right agenda is crucial. Since the problem of finding optimal agendas may be computationally hard, a heuristic approach will be useful for solving this problem. Below, we will explain how heuristics can be employed for generating counteroffers, for predicting the opponent's preferences, and for generating optimal negotiation agendas.

5.1 Heuristics for Generating Counteroffers

Faratin et al. defined a heuristic model based on *negotiation decision functions* [14]. They defined a wide range of negotiation strategies that agents can use to make offers, and also showed how an agent can dynamically change its strategy over time. In order to generate a counteroffer for an issue under negotiation, they defined three types of strategies: *time dependent*, *resource dependent*, and *behavior dependent*, or *imitative*, strategies.

A time dependent strategy is defined as a function that takes “time” as an input and returns an offer in such a way that concessions are small at the beginning of negotiation but increase as the deadline approaches. A family of such functions is defined by varying the concessions made during the course of negotiation. Thus an agent can use one of these functions as a negotiation strategy. For example, let there be two negotiators a and b and let $j \in \{1, 2, \dots, n\}$ be an issue under

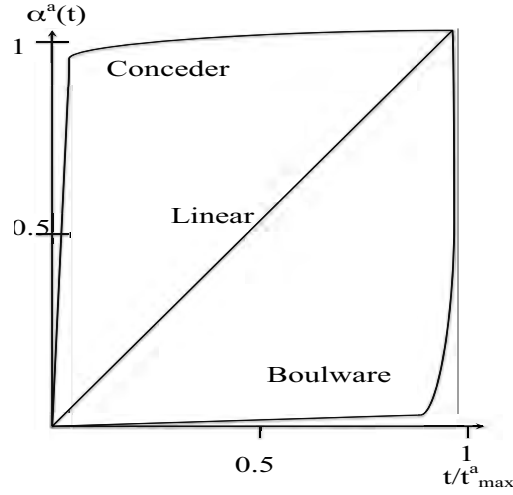


Figure 4.1: A Boulware, a linear, and a Conceder strategy.

negotiation. Assume that issue j represents the “price” of a product. If $x_{a \rightarrow b}^j(t)$ denotes the price offered by a at time t , then we have the following equation for computing an offer:

$$x_{a \rightarrow b}^j(t) = \begin{cases} \min^a + \alpha^a(t)(\max^a - \min^a) & \text{if } a\text{'s utility decreases with price} \\ \min^a + (1 - \alpha^a(t))(\max^a - \min^a) & \text{if } a\text{'s utility increases with price} \end{cases}$$

where \max^a and \min^a denote a 's reserve prices. Here, $\alpha^a(t)$ is a function such that $0 \leq \alpha^a(t) \leq 1$, $\alpha^a(0) = \kappa^a$, and $\alpha^a(t_{\max}^a) = 1$. So at the beginning of negotiation, the offer will be a constant (κ^a), and when the deadline (t_{\max}^a) is reached, the offer will be a 's reserve price.

A family of time dependent strategies can be defined by varying the definition of the function $\alpha^a(t)$. For instance, $\alpha^a(t)$ could be defined as a polynomial function parameterized by $\beta \in \mathbb{R}^+$ as follows:

$$\alpha^a(t) = \kappa^a + (1 - \kappa^a)(\min(t, t_{\max}^a)/t_{\max}^a)^{1/\beta}$$

An infinite number of possible strategies can be obtained by varying β . One can easily verify that for small β (such a strategy is called *Boulware* in [64]), the initial offer is maintained until time almost approaches the deadline and then a quick concession is made to the reserve value. On the other hand, for large β (such a strategy is called *Conceder* in [59]), the offer quickly goes to the reserve value. The parameter β can also be suitably adjusted to obtain a *linear* strategy. These strategies are illustrated in Figure 4.1.

Resource dependent strategies are also defined in terms of a set of functions similar to the time dependent functions except that the domain of the function is the quantity of resources available instead of the remaining time. Here, the functions are used to model the pressure in reaching an agreement that the limited resources may impose on an agent's behavior.

Behavior dependent strategies are for those situations in which there is no pressure in terms of time or resources. Here, an agent simply imitates its opponent's strategy in order to protect itself from being exploited by the opponent. Again, there is a family of functions that differ in terms of the specific aspect of the opponent's behavior that is imitated and the degree to which it is imitated. See [14] for more details on these strategies and how they can be combined together.

5.2 Heuristics for Predicting Opponent's Preferences and Generating Counteroffers

Heuristics can not only be used for generating offers [15], but can also be used to predict the opponent's preferences for the issues. This prediction is relevant to situations where the negotiators have limited information about their opponents. Here, any information gained from the opponent's offers in the past rounds is used to heuristically form a prediction of the future. Heuristics based on *similarity criteria* [15] are useful in this context. These heuristics predict the opponent's preferences by applying a *fuzzy similarity* criteria to the opponent's past offers. For example, consider a buyer-seller negotiation over two issues: the "price" and the "quality" of a product. If, during negotiation, the opponent is making more concessions on price than on quality, then it implies that it prefers quality more than price. This information can then be used to make trade-offs and generate better counteroffers. See [15] for details on how fuzzy similarity learning technique is combined with *hill climbing* to explore the space of possible trade-offs and find an acceptable counteroffer.

The similarity criteria with hill climbing is defined for direct negotiations between the parties. In some negotiations, such as those involving limited information, non-linear utilities, or interdependent issues, an intermediate agent or a *mediator* can be used to facilitate the process of negotiation by allowing the negotiating parties to reach agreements that are optimal at a system-wide level. Klein, Ito, and Hattori [35] showed how heuristics can be used to reach Pareto optimal agreements through a mediator. Here, the agents themselves might not be able to reach agreements that are Pareto optimal. But by using a mediator, they can reach agreements that are better for both negotiators. The parties send offers to a mediator who then determines a Pareto optimal agreement. This approach works if the parties trust the mediator and truthfully reveal information about their preferred

agreements. An agent's preferences will be revealed when it makes offers to the mediator, because an agent will only propose those offers that are optimal from its individual perspective. However, due to non-linear utilities and interdependencies between the issues, it is not easy for a party to decide what to offer to the mediator. Here comes the role of heuristics such as *hill climbing* and *simulated annealing*. See [35] for details regarding these heuristics. Similar work in the context of auctions was done by Marsa-Maestre et al. [52].

5.3 Heuristics for Generating Optimal Agendas

Heuristics can not only be used to predict the opponent's preferences or for generating counteroffers, but can also be used to find *optimal negotiation agendas*. The approaches described in the previous sections assume that the set of issues to be negotiated are given, and show how to generate offers for those issues. But in many cases, it is not just the negotiation offers that determine the outcome of a negotiation, but the actual issues included for negotiation also play a crucial part. For example, consider a car dealer who has m different cars for sale. A potential buyer may be interested in buying $g < m$ cars. So the buyer must first choose which cars to negotiate the price for. From the $\binom{m}{g}$ possible subsets (i.e., possible agendas) of size g , the buyer must choose one. Since different agendas may yield different utilities to the negotiators, a utility maximizing buyer will want to choose the one that maximizes its utility and is therefore its *optimal agenda*. Finding an optimal agenda is therefore crucial to the outcome of a negotiation.

Due to the complexity of finding optimal agendas, a *genetic algorithm* (GA) approach was used by Fatima et al. in [16] for evolving optimal agendas for the package deal procedure. For the case of non-linear utilities, this approach employs two GA search engines: one for evolving an optimal agenda and another for evolving an optimal negotiation strategy (i.e., an optimal allocation of the issues on an agenda) for a given agenda. The former uses a *surrogate-assisted* GA and the latter uses a standard GA. The surrogate-assisted GA uses a surrogate (or approximation) to model an agent's utility from a given agenda. See [16] for details about this heuristic method.

5.4 Heuristics for Reasoning about Deliberation Cost

In many scenarios, negotiating agents need to solve computationally intractable problems. For example, consider a distributed vehicle routing problem [72], in which two self-interested dispatch centers (e.g., belonging to different companies) need to fulfill a number of tasks (deliveries), and have resources (vehicles) at their disposal. In this case, each agent has an individual optimization problem, aiming

to make all deliveries, while minimizing transportation cost (i.e., driven mileage), subject to the following constraints:

- Each vehicle has to begin and end its tour at the depot of its center (but neither the pickup nor the drop-off locations of the orders need to be at the depot).
- Each vehicle has a maximum load weight constraint. These may differ among vehicles.
- Each vehicle has a maximum load volume constraint. These may differ among vehicles.
- Each vehicle has a maximum route length (prescribed by law).
- Each delivery has to be included in the route of some vehicle.

This individual agent optimization problem is NP-complete [72]. In addition, agents can potentially save mileage by pooling resources and tasks. This joint problem is also NP-complete [72]. Hence, although the agents can negotiate to find ways to share the delivery tasks and resources (and to divide the cost), they must solve intractable problems in the process.

This sort of scenario is precisely why heuristics are needed in some negotiation domains. Indeed, agents may even reason about this process explicitly, for example by controlling an *anytime algorithm* [10] based on the degree to which it is expected to improve the current solution while negotiating with another agent. Based on this idea, Larson and Sandholm introduced a new solution concept, dubbed *deliberation equilibrium*, in which deliberation (i.e., *computation*) costs are factored into the agents' strategies explicitly [48].

6 Negotiating with Humans

So far, we have discussed various approaches to programming software agents that are capable of negotiating rationally with other software agents. In such settings, one builds on the assumption that the opponent is built in such a way as to act rationally, i.e., to further its own goals. In cases where software agents cannot be assumed to be rational (e.g., because they have to solve intractable problems), we saw how heuristics can be useful.

However, when agents negotiate with humans, a completely new challenge arises. This is because, as research on *bounded rationality* has shown [26], humans make systematic deviations from the optimal behavior prescribed by normative theory. For example, people often change their valuations based on how the

choices are framed [78], are averse to inequity [5], and are willing to engage in irrational behavior such as costly punishment [13].

Given this, one often cannot assume that humans will follow the equilibrium strategy, or even be utility maximizers. Instead, one must endow software agents with both (i) *predictive behavioral models* of how humans negotiate, and (ii) *decision-making algorithms* that take these models into account as they guide the agent's behavior. Modeling human negotiation is a non-trivial problem, however. As one can imagine, modeling human negotiation behavior is a non-trivial endeavor, and a field of study in its own right in the fields of psychology and business study [49].

The above challenge is further complicated by the fact that the purpose of agents that negotiate with humans can vary. Following are some possible objectives of such agents:

- Outperform human negotiators in a web-based market.
- Train people in negotiation skills to help them negotiate with other people.

Depending on the purpose of the agent, the types of behavioral and decision-making models may differ substantially. For example, an agent that is designed to train people in negotiation skills would need to mimic other humans, while an agent trying to make money in an online market simply needs to maximize profit.

One of the earliest agents capable of negotiating with humans was designed by Kraus and Lehmann to play the game *Diplomacy* using a variety of heuristics [44]. Surprisingly, humans were unable to discern whether they were playing with a human or an agent. More recently, Katz and Kraus introduced agents that use reinforcement learning to participate in single-shot auctions or games, and have been shown to achieve higher payoffs than humans [39]. Building on this work, they later introduced gender-sensitive learning, which achieves even better results [40].

Another significant line of work builds on the *Colored Trails* (CT) platform, which is a software infrastructure for investigating decision making in groups comprising people, computer agents, and a mix of these two [27]. Although the CT framework is highly customizable, most work focused on an $n \times m$ board of colored squares, around which individual players can move. Each player has a designated goal square, and can move to it provided it possesses chips that match the colors of the squares along the path. Each player is initially endowed with different colored chips, which may or may not be sufficient for reaching its individual goal. Thus, players may need to negotiate with one another in order to redistribute those chips. Figure 4.2 shows a screen shot of a CT game [50], displaying the board, the player's location (marked "me"), the player's goal (marked "G"), the negotiation counterpart (marked as a square), the chips each player has,

and a proposal panel that the player is using to prepare an offer to send to the counterpart. Note that, in this case, the player cannot see the counterpart's goal. Other variants are possible, for example in which the players do not see each other's current chip endowment.

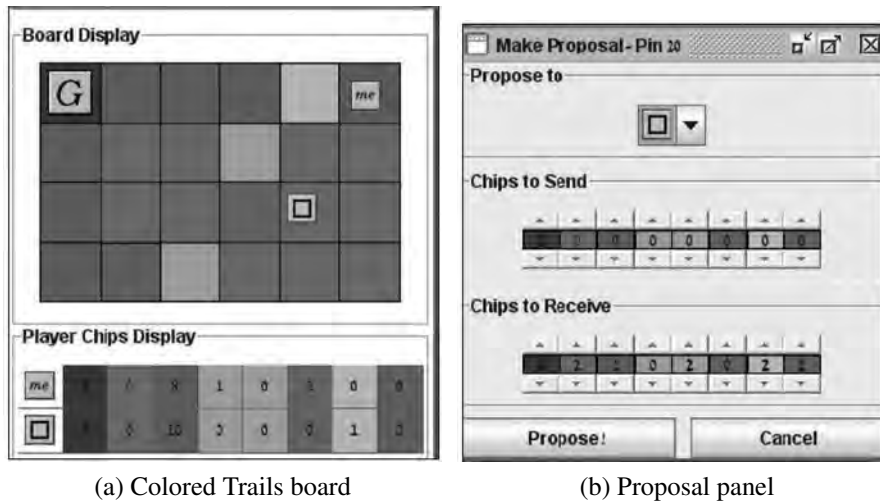


Figure 4.2: Colored Trails user interface example.

The CT platform has been used to conduct controlled experiments to study human negotiation behavior. For example, it was used to build predictive models of humans' reciprocity toward other negotiators [24], or willingness to reveal individual goals [11]. By pitching automated agents against human players, CT was also used to evaluate agents capable of adapting to social [22] and cultural [23] attributes that influence human decision making in negotiation. Finally, CT was used to conduct novel experiments to understand how humans design automated negotiation strategies [51]. A more comprehensive survey of human-agent negotiation can be found in a recent article by Lin and Kraus [50].

7 Argumentation-Based Negotiation

Game-theoretic and heuristics-based approaches to automated negotiation are characterized by the exchange of offers between parties with conflicting positions and are commonly referred to as *proposal-based* approaches. That is, agents exchange proposed agreements – in the form of bids or offers – and when proposed

deals are not accepted, the possible response is either a counterproposal or withdrawal. *Argumentation-based negotiation* (ABN) approaches, on the other hand, enable agents to exchange additional *meta*-information (i.e., arguments) during negotiation [63].

Consider the case in which an agent may not be aware of some alternative plans of achieving some goal. Exchanging this information may enable agents to reach agreements not previously possible. This was shown through the well-known painting/mirror hanging example presented by Parsons et al. [57]. The example concerns two home-improvement agents – agent i trying to hang a painting, and agent j trying to hang a mirror. There is only one way to hang a painting, using a nail and a hammer. But there are two ways of hanging a mirror, using a nail and a hammer or using a screw and a driver, but j is only aware of the former. Agent i possesses a screw, a screw driver, and a hammer, but needs a nail in addition to the hammer to hang the painting. On the other hand, j possesses a nail, and believes that to hang the mirror, it needs a hammer in addition to the nail. Now, consider the dialogue depicted in Figure 4.3 (described here in natural language) between the two agents.

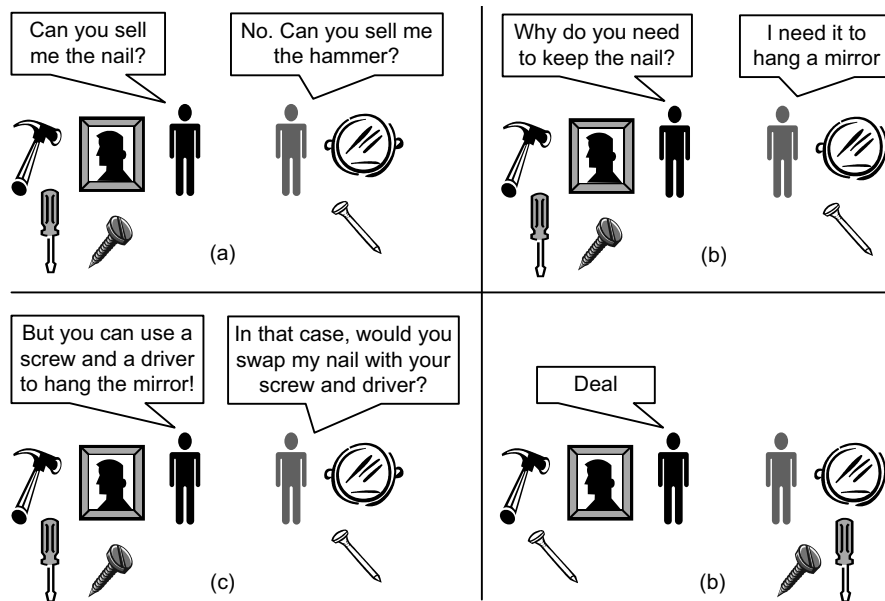


Figure 4.3: Dialogue between agents i (black) and j (gray).

As the figure shows, at first, j was not willing to give away the nail because it needed it to achieve its goal. But after finding out the reason for rejection, i

managed to persuade j to give away the nail by providing an alternative plan for achieving the latter's goal.

This type of negotiation dialogue requires a communication protocol that enables agents to conduct a discussion about a domain of interest using a shared vocabulary (i.e., ontology). Furthermore, it requires the ability to present justifications of one's position, as well as counterarguments that influence the counterpart's mental state (e.g., its goals, beliefs, plans) [46]. Consequently, much work on ABN builds on logic-based argumentation protocols. For example, Parsons et al. [57] present a framework based on the logic-based argumentation framework of Elvang-Gøransson et al. [12]. The framework of Sadri et al. [71] uses abductive logic programming [21]. Other frameworks allow the exchange of a variety of other information relevant to negotiation, such as threats and rewards [2, 65], or claims about social rights and obligations [38].

There has been some work on characterizing the outcomes of argument-based negotiation (e.g., see [62] or [1]). However, the connection between ABN formalisms and classical models of negotiation is still not well-developed. For example, it is not clear whether and how a particular argumentation protocol achieves properties like Pareto optimality or independence of irrelevant alternatives. In addition, most existing models of argumentation lack an explicit model of strategic behavior. This is a major drawback, since agents may withhold or misreport arguments in order to influence the negotiation outcome to their own advantage. For a recent discussion on strategic behavior in argumentation, and initial attempts at grounding it in game theory, see [60, 61].

8 Conclusions

In this chapter, we studied some of the key concepts and techniques used for modeling bargaining. Specifically, we looked at the axiomatic and non-cooperative models of bargaining. Each of these two approaches has strengths and limitations. For example, a main strength of the Nash bargaining model is its simplicity and the uniqueness of its solution. However, this approach can be criticized because it ignores the whole process of making offers and counteroffers, and the possibility of a breakdown. The model may therefore be more relevant to bargaining with arbitration. In non-cooperative bargaining, the process of making offers and counteroffers is modeled as an alternating offers game. But the solution to this model assumes that both players are able to apply backward induction logic. This assumption has been criticized because performing backward induction can sometimes require complex computations regarding events that never actually take place [47]. Despite the differences, the axiomatic and strategic approaches can sometimes be complementary in that each can help to justify and

clarify the other [56].

Although bargaining has long been studied by economists and game theorists, its study in the context of multiagent systems is relatively recent. While the game-theoretic models of bargaining provide some key insights, they also open up new challenges when applied in the context of computational agents. These challenges arise mainly due to the bounded rationality of computational agents. Thus bargaining solutions must be found that are not only individual rational and Pareto optimal, but also computationally feasible. This will require the use of heuristic techniques.

Another challenge that arises from the bounded rationality of negotiators is how to design agents that can not only effectively negotiate with other agents but also with human negotiators. Results from research in the social sciences have shown that humans do not necessarily maximize utilities and they do not always follow equilibrium strategies. The problem of designing heuristic strategies that take this kind of behavior into account is an open research problem.

Acknowledgments

The first author acknowledges the EPSRC grant EP/G000980/1 as some parts of the chapter are based on results of this project. Some parts are also based on joint work with Michael Wooldridge and Nicholas Jennings. We are very thankful to both of them. The topic of bargaining is too broad to cover in a single chapter. Our apologies for not being able to include a lot of important work due to space limitations.

9 Exercises

1. **Level 1** Suppose that $x \in \mathcal{S}$, $y \in \mathcal{S}$, $x^a = d^a$, $y^b = d^b$, and $0.5x + 0.5y$ is a strongly efficient allocation in \mathcal{S} . Find the Nash bargaining solution of (\mathcal{S}, d) .
2. **Level 1** Suppose that there are two divisible pies (x and y) of unit size. Two players (a and b) bargain over the division of these pies using the package deal procedure. Let the negotiation deadline be 3 and the discount factor be $\delta = 0.25$. Assume that a has utility function $U^a = 2x^a + y^a$ and b has $U^b = x^b + 3y^b$, where x^a and y^a denote a 's allocation of the two pies and x^b and y^b denote b 's allocation. Under the complete information assumption, when will an agreement be reached and what will each player's equilibrium allocation be?

3. **Level 2** Develop a program that will take the number of issues, the negotiation deadline, the discount factor, and the two players' utility functions as input, and generate the equilibrium allocation for the package deal procedure under the complete information assumption.
4. **Level 3** Design a software negotiating agent that can generate offers using the negotiation decision functions described in Section 5.1. Do this first for single-issue negotiation and then for multi-issue negotiation.
5. **Level 4** Define a multi-issue negotiation setting in terms of a deadline, a discount factor, and the player's utility functions. For this setting, conduct a negotiation experiment with human participants using the package deal procedure and study their behavior. Then analyze how this compares with game-theoretic equilibrium behavior.

References

- [1] L. Amgoud, Y. Dimopoulos, and P. Moraitis. A unified and general framework for argumentation-based negotiation. In *AAMAS '07: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, USA, 2007. ACM.
- [2] L. Amgoud and H. Prade. Handling threats, rewards and explanatory arguments in a unified setting. *International Journal of Intelligent Systems*, 20(12):1195–1218, 2005.
- [3] M. Bac and H. Raff. Issue-by-issue negotiations: the role of information and time preference. *Games and Economic Behavior*, 13:125–134, 1996.
- [4] Y. Bar-Yam. *Dynamics of Complex Systems*. Addison Wesley, 1997.
- [5] G.E. Bolton. A comparative model of bargaining: Theory and evidence. *The American Economic Review*, pages 1096–1136, 1991.
- [6] L. A. Busch and I. J. Horstman. Bargaining frictions, bargaining procedures and implied costs in multiple-issue bargaining. *Economica*, 64:669–680, 1997.
- [7] F. Carmichael. *A Guide to Game Theory*. Prentice Hall, 2005.
- [8] P. C. Cramton. Strategic delay in bargaining with two-sided uncertainty. *Review of Economic Studies*, 59:205–225, 1992.
- [9] P. C. Cramton, L. M. Ausbel, and R. J. Denekere. Bargaining with incomplete information. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory*, pages 1897–1945. Amsterdam: Elsevier Science B.V., 2002.

- [10] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.
- [11] S. D’souza, Y. Gal, P. Pasquier, S. Abdallah, and I. Rahwan. Reasoning about goal revelation in human negotiation. *Intelligent Systems, IEEE*, (99).
- [12] M. Elvang-Gøransson, P. Krause, and J. Fox. Dialectic reasoning with inconsistent information. In David Heckerman and Abe Mamdani, editors, *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 114–121, Washington D.C., USA, 1993. Morgan Kaufmann.
- [13] F. Ernst and G. Simon. Altruistic punishment in humans. *Nature*, 415(6868):137–140, 2002.
- [14] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.
- [15] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make trade-offs in automated negotiations. *Artificial Intelligence Journal*, 142(2):205–237, 2002.
- [16] S. S. Fatima and A. Kattan. Evolving optimal agendas for package deal negotiation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO–2011)*, pages 505–512, Dublin, Ireland, July 2011.
- [17] S. S. Fatima, M. Wooldridge, and N. R. Jennings. An agenda-based framework for multi-issue negotiation. *Artificial Intelligence Journal*, 152(1):1–45, 2004.
- [18] S. S. Fatima, M. Wooldridge, and N. R. Jennings. Multi-issue negotiation with deadlines. *Journal of Artificial Intelligence Research*, 27:381–417, 2006.
- [19] S. S. Fatima, M. Wooldridge, and N. R. Jennings. An analysis of feasible solutions for multi-issue negotiation involving non-linear utility functions. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multi-agent Systems*, pages 1041–1048, Budapest, Hungary, May 2009.
- [20] C. Fershtman. The importance of the agenda in bargaining. *Games and Economic Behavior*, 2:224–238, 1990.
- [21] T. Fung and Robert Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(1):151–165, 1997.
- [22] Y. Gal, B. Grosz, S. Kraus, A. Pfeffer, and S. Shieber. Agent decision-making in open mixed networks. *Artificial Intelligence*, 174(18):1460–1480, 2010.
- [23] Y. Gal, S. Kraus, M. Gelfand, H. Khashan, and E. Salmon. An adaptive agent for negotiating with people in different cultures. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(1):8, 2011.

- [24] Y. Gal and A. Pfeffer. Modeling reciprocity in human bilateral negotiation. In *National Conference on Artificial Intelligence (AAAI)*, Vancouver, British Columbia, 2007.
- [25] N. Gatti, F. Giunta, and S. Marino. Alternating offers bargaining with one-sided uncertain deadlines: An efficient algorithm. *Artificial Intelligence Journal*, 172:1119–1157, 2008.
- [26] G. Gigerenzer and R. Selten, editors. *Bounded Rationality: The Adaptive Toolbox*. Dahlem Workshop Reports. MIT Press, Cambridge MA, USA, 2002.
- [27] B.J. Grosz, S. Kraus, S. Talman, B. Stossel, and M. Havlin. The influence of social dependencies on decision-making: Initial investigations with a new game. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 782–789. IEEE Computer Society, 2004.
- [28] J. C. Harsanyi. Games of incomplete information played by Bayesian players - Part I. *Management Science*, 14:159–182, November 1967.
- [29] J. C. Harsanyi. Games of incomplete information played by Bayesian players - Part II. *Management Science*, 14:320–334, January 1968.
- [30] J. C. Harsanyi. Games of incomplete information played by Bayesian players - Part III. *Management Science*, 14:486–502, March 1968.
- [31] J. C. Harsanyi and R. Selten. A generalized Nash solution for two-person bargaining games with incomplete information. *Management Science*, 18(5):80–106, January 1972.
- [32] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, 1996.
- [33] Y. In and R. Serrano. Agenda restrictions in multi-issue bargaining (II): Unrestricted agendas. *Economics Letters*, 79:325–331, 2003.
- [34] R. Inderst. Multi-issue bargaining with endogenous agenda. *Games and Economic Behavior*, 30:64–82, 2000.
- [35] T. Ito, H. Hattori, and M. Klein. Multi-issue negotiation protocol for agents: Exploring nonlinear utility spaces. In *Proc. of the Twentieth Int. Joint Conf. on Artificial Intelligence*, pages 1347–1352, 2007.
- [36] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

- [37] N.R. Jennings, P. Faratin, T.J. Norman, P. O'Brien, B. Odgers, and J. L. Alty. Implementing a business process management system using ADEPT: A real-world case study. *Int. Journal of Applied Artificial Intelligence*, 14(5):421–465, 2000.
- [38] N. Karunatilake, N. R. Jennings, I. Rahwan, and P. McBurney. Dialogue games that agents play within a society. *Artificial Intelligence*, 173(9-10):935–981, 2009.
- [39] R. Katz and S. Kraus. Efficient agents for cliff-edge environments with a large set of decision options. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 697–704. ACM, 2006.
- [40] R. Katz and S. Kraus. Gender-sensitive automated negotiators. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 821. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
- [41] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. New York: John Wiley, 1976.
- [42] M. Klein, P. Faratin, H. Sayama, and Y. Bar-Yam. Negotiating complex contracts. *IEEE Intelligent Systems*, 8(6):32–38, 2003.
- [43] S. Kraus. *Strategic Negotiation in Multiagent Environments*. The MIT Press, Cambridge, Massachusetts, 2001.
- [44] S. Kraus and D. Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171, 1995.
- [45] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Negotiation under time constraints. *Artificial Intelligence Journal*, 75(2):297–345, 1995.
- [46] Sarit Kraus, Katia Sycara, and Amir Evenchik. Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, 104(1–2):1–69, 1998.
- [47] D. M. Kreps. *Game Theory and Economic Modeling*. Oxford: Clarendon Press, 1993.
- [48] K. Larson and T. Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001.
- [49] Roy J. Lewicki, David M. Saunders, John W. Minton, and Bruce Barry. *Negotiation*. McGraw-Hill/Irwin, New York NY, USA, fourth edition, 2003.
- [50] R. Lin and S. Kraus. Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88, 2010.

- [51] R. Lin, S. Kraus, Y. Oshrat, Y.K. Gal, et al. Facilitating the evaluation of automated negotiators using peer designed agents. *Proc. of the 24th Association for the Advancement of Artificial Intelligence (AAAI-2010)*, 2010.
- [52] I. Marsa-Maestre, M. Lopez-Carmona, J. Velasco, T. Ito, M. Klein, and K. Fujita. Balancing utility and deal probability for auction-based negotiations in highly non-linear utility spaces. In *Proc. of the Twenty-First Int. Joint Conf. on Artificial Intelligence*, pages 214–219, 2009.
- [53] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, 1995.
- [54] R.B. Myerson. Two-person bargaining problems and comparable utility. *Econometrica*, 45(7):1631–1637, 1977.
- [55] J. F. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.
- [56] J. F. Nash. Two-person cooperative games. *Econometrica*, 21:128–140, 1953.
- [57] Simon Parsons, Carles Sierra, and Nick Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
- [58] C. Ponsati and J. Watson. Multiple-issue bargaining and axiomatic solutions. *International Journal of Game Theory*, 26:501–524, 1997.
- [59] D. G. Pruitt. *Negotiation Behavior*. Academic Press, 1981.
- [60] I. Rahwan and K. Larson. Argumentation and game theory. *Argumentation in Artificial Intelligence*, pages 321–339, 2009.
- [61] I. Rahwan and K. Larson. Logical mechanism design. *Knowledge Engineering Review*, 26(1):61–69, 2011.
- [62] I. Rahwan, P. Pasquier, L. Sonenberg, and F. Dignum. A formal analysis of interest-based negotiation. *Annals of Mathematics and Artificial Intelligence*, 55(3):253–276, 2009.
- [63] I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *Knowledge Engineering Review*, 18(4):343–375, 2003.
- [64] H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, Cambridge, USA, 1982.
- [65] S. D. Ramchurn, C. Sierra, L. Godo, and N. R. Jennings. Negotiating using rewards. *Artificial Intelligence*, 171(10–15):805–837, 2007.

- [66] J. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge MA, USA, 1994.
- [67] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994.
- [68] A. Roth. *Axiomatic Models of Bargaining*. Springer-Verlag, Berlin, 1979.
- [69] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.
- [70] A. Rubinstein. A bargaining model with incomplete informations about time preferences. *Econometrica*, 53:1151–1172, January 1985.
- [71] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: An abductive approach. In K. Stathis and M. Schroeder, editors, *Proceedings of the AISB 2001 Symposium on Information Agents for E-Commerce*, 2001.
- [72] T.W. Sandholm and V.R.T. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1-2):99–137, 1997.
- [73] T. Sandholm and N. Vulkan. Bargaining with deadlines. In *AAAI-99*, pages 44–51, Orlando, FL, 1999.
- [74] Tuomas Sandholm. eMediator: A next generation electronic commerce server. *Computational Intelligence*, 18(4):656–676, 2002.
- [75] T. C. Schelling. *The Strategy of Conflict*. Oxford University Press, 1960.
- [76] A. Shaked and J. Sutton. Two-person bargaining problems with incomplete information. *Econometrica*, 52:461–488, 1984.
- [77] I. Stahl. *Bargaining Theory*. Economics Research Institute, Stockholm School of Economics, Stockholm, 1972.
- [78] A. Tversky and D. Kahneman. The framing of decisions and the psychology of choice. *Science*, 211:453–458, 1981.

Chapter 5

Argumentation among Agents

Iyad Rahwan

1 Introduction

The theory of argumentation is a rich, interdisciplinary area of research spanning philosophy, communication studies, linguistics, and psychology [53]. Argumentation can be seen as a reasoning process consisting of the following four steps:

1. Constructing *arguments* (in favor of/against a “statement”) from available information.
2. Determining the different *conflicts* among the arguments.
3. Evaluating the *acceptability* of the different arguments.
4. Concluding, or defining, the *justified conclusions*.

The argumentation metaphor has found a wide range of applications in both theoretical and practical branches of artificial intelligence and computer science [5, 44]. These applications range from specifying semantics for logic programs [12], to natural language text generation [14], to supporting legal reasoning [4], to medical decision-support [16].

In multiagent systems, argumentation has been used both for automating individual agent reasoning, as well as multiagent interaction. In this chapter, I focus on the latter. I refer the reader to other articles for examples of argumentation-based single-agent reasoning, such as belief revision [15] or planning [3, 41]. For

a more comprehensive overview of argumentation in AI, the reader may refer to the recent books on the subject [7, 46]. Due to space limitations, my coverage cannot do justice to the broad literature on argumentation in multiagent systems. Instead, I chose to give some representative approaches to give the reader an overview of the main ideas and insights.

2 What Is an Argument?

Among argumentation theorists in philosophy, the term “argument” usually refers to “the giving of reasons to support or criticize a claim that is questionable, or open to doubt” [59]. This distinguishes argumentation from deductive mathematical inference, in which the conclusions follow necessarily from the premises. Here, I give a very brief overview of the major approaches to formalizing this notion in the AI literature.

2.1 Arguments as Chained Inference Rules

Formalizations of argumentation using logically constructed structures have their roots in the work of Pollock [34], Lin and Shoham [23], and Vreeswijk [56]. Other approaches include those building on classical logic [6] or logic programming formalisms [8, 49].

A number of recent attempts have been made to provide a general, unifying definition. In this section, I give a very brief overview of Prakken’s recent unifying framework [36], since it is quite general and highlights most important concepts. Prakken defines an *argumentation system* as a tuple $(\mathcal{L}, -, \mathcal{R}_s, \mathcal{R}_d, \leq)$, consisting of a logical language \mathcal{L} , two disjoint sets of strict rules \mathcal{R}_s and defeasible rules \mathcal{R}_d , and a partial order \leq over \mathcal{R}_d . The *contrariness function* $- : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ captures conflict between formulas, with classical negation \neg being captured by $\neg\phi \in \bar{\phi}$ and $\phi \in \neg\bar{\phi}$.

A particular knowledge base is a pair (\mathcal{K}, \leq') , with $\mathcal{K} \subseteq \mathcal{L}$ divided into the following disjoint sets: \mathcal{K}_n are the necessary *axioms* (cannot be attacked); \mathcal{K}_p are the *ordinary premises*; \mathcal{K}_a are the *assumptions*; \mathcal{K}_i are the *issues*. Finally, \leq' is a partial order on $\mathcal{K} \setminus \mathcal{K}_n$.

From a knowledge base, arguments are built by applying *inference rules* to subsets of \mathcal{K} . The left-hand side of the rule ϕ_1, \dots, ϕ_n is called the *premises* (or *antecedents*), while the right-hand side ϕ is called the *conclusion* (or *consequent*). A *strict* rule of the form $\phi_1, \dots, \phi_n \rightarrow \phi$ stands for classical implication, while a *defeasible* rule of the form $\phi_1, \dots, \phi_n \Rightarrow \phi$ means that ϕ follows *presumably* from the premises. Functions $Perm(A)$, $Conc(A)$ and $Sub(A)$ return premises,

conclusion, and *subarguments* of argument A , respectively. Omitting some details (see [36]), an argument is intuitively any of the following structures:

- $\varphi \in \mathcal{K}$, where $Prem(A) = \{\varphi\}$, $Conc(A) = \varphi$, and $Sub(A) = \{\varphi\}$.
- $A_1, \dots, A_n \rightarrow \psi$, where A_1, \dots, A_n are arguments, and there exists in \mathcal{R}_s a strict rule $Conc(A_1), \dots, Conc(A_n) \rightarrow \psi$.
- $A_1, \dots, A_n \Rightarrow \psi$, where A_1, \dots, A_n are arguments, and there exists in \mathcal{R}_d a defeasible rule $Conc(A_1), \dots, Conc(A_n) \rightarrow \psi$.

In the second and third cases, we define $Prem(A) = Prem(A_1) \cup \dots \cup Prem(A_n)$, that is, the premises of an argument are the union of the premises of its constituents. Likewise, $Sub(A) = Sub(A_1) \cup \dots \cup Sub(A_n) \cup \{A\}$. The following example is illustrative:

Example 5.1 (from [36]) Consider a knowledge base in an argumentation system with $\mathcal{R}_s = \{p, q \rightarrow s; u, v \rightarrow w\}$, $\mathcal{R}_d = \{p \Rightarrow t; s, r, t \Rightarrow v\}$, $\mathcal{K}_n = \{q\}$, $\mathcal{K}_p = \{p, u\}$, $\mathcal{K}_a = \{r\}$. We can construct the following arguments:

$$\begin{array}{ll} A_1 : p & A_5 : A_1 \Rightarrow t \\ A_2 : q & A_6 : A_1, A_2 \rightarrow s \\ A_3 : r & A_7 : A_3, A_5, A_6 \Rightarrow v \\ A_4 : u & A_8 : A_4, A_7 \rightarrow w \end{array}$$

with $Prem(A_8) = \{p, q, r, u\}$, $Conc(A_8) = \{w\}$, and $Sub(A_8) = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$.

The example above shows how an argument can form part of (the *support* of) another argument. The other possibility is when an argument *attacks* another argument. Various notions of attack have been explored in the literature. For example, an argument can *undercut* another argument by showing that a defeasible rule cannot be applied.

Example 5.2 In Example 5.1, argument A_8 can be undercut by an argument with conclusion $\overline{A_5}$, since argument A_5 is constructed with a defeasible rule.

Another way to attack an argument is to *rebut* with an argument that has the opposite conclusion.

Example 5.3 In Example 5.1, argument A_8 can be rebutted on A_5 with an argument with conclusion \bar{t} , or rebutted on A_7 with an argument with conclusion \bar{v} .

Rebutters and undercutters were first formalized by Pollock [34]. Yet another way to attack an argument is to *undermine* it by attacking one of its premises (also called *premise attack*), introduced by Vreeswijk [56].

Example 5.4 In Example 5.1, argument A_8 can be undermined by an argument with conclusion \bar{p} , \bar{r} , or \bar{u} .

Finally, we say that an argument A *defeats* argument B if the former attacks the latter, and is also preferred to it according to some preference relation \prec , which is itself based on the nature of the attack as well as the preference relations \leq and \leq' over the formulas involved. Details are beyond this introductory chapter, and can be found elsewhere [36].

2.2 Argument as an Instance of a Scheme

Argumentation schemes are forms (or categories) of argument, representing stereotypical ways of drawing inferences from particular patterns of premises to conclusions in a particular domain (e.g., reasoning about action). Prior to the development of formal models of argumentation, a number of such informal schemes have been proposed [33, 51]. One of the most comprehensive classifications was presented by Walton [58]. For each category, Walton specifies the scheme informally by listing the general form of the premises and conclusion, together with a set of critical questions that can be used to scrutinize the argument by questioning explicit or implicit premises. For example, Walton’s “sufficient condition scheme for practical reasoning” may be described as follows [1]:

In the current circumstances R
 We should perform action A
 Which will result in new circumstances S
 Which will realize goal G
 Which will promote some value V .

And its associated critical questions include:

- CQ1:** Are the believed circumstances true?
- CQ2:** Does the action have the stated consequences?
- CQ3:** Assuming the circumstances and that the action has the stated consequences, will the action bring about the desired goal?
- CQ4:** Does the goal realize the value stated?
- CQ5:** Are there alternative ways of realizing the same consequences?

A number of formalizations of Walton’s schemes have been attempted. For example, Atkinson and Bench-Capon [1] formalize the above scheme using Action-based Alternating Transition Systems (AATS), which are a variant of alternating time logic [52]. Walton’s schemes have also been influential in the proposed Argument Interchange Format (AIF), which is a community-based effort to produce

an extensible ontology for describing argument structures [11]. Yet another formalization of Walton's scheme was presented by Gordon et al. in their Carneades model [18].

2.3 Abstract Arguments

In the preceding sections, each argument had an explicit internal structure, and relationships between arguments were defined carefully, using various notions of attack they pose or critical questions they raise. In a seminal article, Dung showed that many fundamental aspects of argumentation can be studied without paying attention to the internal structure of the arguments themselves [12]. Thus, an argument can be seen as a node in an argument graph, in which directed arcs capture defeat between arguments. Despite its simplicity, this model is surprisingly powerful.

Definition 5.1 (Argumentation framework) *An argumentation framework is a pair $AF = \langle \mathcal{A}, \rightarrow \rangle$ where \mathcal{A} is a finite set of arguments and $\rightarrow \subseteq \mathcal{A} \times \mathcal{A}$ is a defeat relation. We say that an argument α defeats an argument β if $(\alpha, \beta) \in \rightarrow$ (sometimes written $\alpha \rightarrow \beta$).*

An argumentation framework can be represented as a directed graph in which vertices are arguments and directed arcs characterize defeat among arguments. An example of an argument graph is shown in Figure 5.1. Argument α_1 has two defeaters (i.e., counterarguments) α_2 and α_4 , which are themselves defeated by arguments α_3 and α_5 , respectively.

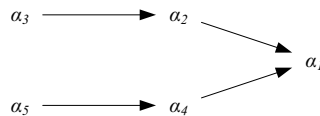


Figure 5.1: A simple argument graph.

3 Evaluating an Argument

To evaluate whether an argument is acceptable (according to some logical semantics), we need to take into account how it interacts with other arguments. This turns out to be a non-trivial problem, and the source of much research [2] and controversy [9]. Let $S^+ = \{\beta \in \mathcal{A} \mid \alpha \rightarrow \beta \text{ for some } \alpha \in S\}$. Also let

$\alpha^- = \{\beta \in \mathcal{A} \mid \beta \rhd \alpha\}$. We first characterize the fundamental notions of conflict-free and defense.

Definition 5.2 (Conflict-free, defense) Let $\langle \mathcal{A}, \rhd \rangle$ be an argumentation framework and let $S \subseteq \mathcal{A}$ and let $\alpha \in \mathcal{A}$.

- S is conflict-free if $S \cap S^+ = \emptyset$.
- S defends argument α if $\alpha^- \subseteq S^+$. We also say that argument α is acceptable with respect to S .

Intuitively, a set of arguments is *conflict-free* if no argument in that set defeats another. A set of arguments *defends* a given argument if it defeats all its defeaters. In Figure 5.1, for example, $\{\alpha_3, \alpha_5\}$ defends α_1 . We now look at different semantics that characterize the *collective acceptability* of a set of arguments.

Definition 5.3 (characteriztic function) Let $AF = \langle \mathcal{A}, \rhd \rangle$ be an argumentation framework. The characteriztic function of AF is $\mathcal{F}_{AF}: 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ such that, given $S \subseteq \mathcal{A}$, we have $\mathcal{F}_{AF}(S) = \{\alpha \in \mathcal{A} \mid S \text{ defends } \alpha\}$.

When there is no ambiguity, we use \mathcal{F} instead of \mathcal{F}_{AF} . The characteriztic function allows us to capture the collective criteria of admissibility.

Definition 5.4 (Admissible set) Let S be a conflict-free set of arguments in framework $\langle \mathcal{A}, \rhd \rangle$. S is *admissible* if it is conflict-free and defends every element in S (i.e., if $S \subseteq \mathcal{F}(S)$).

Intuitively, a set of arguments is *admissible* if it is a conflict-free set that defends itself against any defeater – in other words, if it is a conflict-free set in which each argument is acceptable with respect to the set itself.

Example 5.5 In Figure 5.1, the sets \emptyset , $\{\alpha_3\}$, $\{\alpha_5\}$, and $\{\alpha_3, \alpha_5\}$ are all *admissible* simply because they do not have any defeaters. The set $\{\alpha_1, \alpha_3, \alpha_5\}$ is also *admissible* since it defends itself against both defeaters α_2 and α_4 .

An admissible set S is a *complete extension* if and only if all arguments defended by S are also in S (that is, if S is a fixed point of the operator \mathcal{F}).

Definition 5.5 (Complete extensions) Let S be a conflict-free set of arguments in framework $\langle \mathcal{A}, \rhd \rangle$. S is a *complete extension* if $S = \mathcal{F}(S)$.

Example 5.6 In Figure 5.1, the admissible set $\{\alpha_3, \alpha_5\}$ is not a *complete extension*, since it defends α_1 but does not include α_1 . Similarly, sets $\{\alpha_3\}$ and $\{\alpha_5\}$ are not *complete extensions*, since $\mathcal{F}(\{\alpha_3\}) = \{\alpha_3, \alpha_5\}$ and $\mathcal{F}(\{\alpha_5\}) = \{\alpha_3, \alpha_5\}$. The admissible set $\{\alpha_1, \alpha_3, \alpha_5\}$ is the only *complete extension*, since $\mathcal{F}(\{\alpha_1, \alpha_3, \alpha_5\}) = \{\alpha_1, \alpha_3, \alpha_5\}$.

Before moving to further refinements of the complete extension, I discuss an equivalent way to characterize complete extensions using *argument labeling* [10]. A labeling specifies which arguments are accepted (labeled in), which ones are rejected (labeled out), and which ones whose acceptance or rejection could not be decided (labeled undec). Labelings must satisfy the condition that an argument is in if and only if all of its defeaters are out. An argument is out if and only if at least one of its defeaters is in. Otherwise, it is undecided.

Definition 5.6 (Argument labeling) Let $AF = \langle \mathcal{A}, \rightarrow \rangle$. A labeling is a total function $L : \mathcal{A} \rightarrow \{\text{in}, \text{out}, \text{undec}\}$ such that:

- $\forall \alpha \in \mathcal{A} : (L(\alpha) = \text{out} \equiv \exists \beta \in \mathcal{A} \text{ such that } (\beta \rightarrow \alpha \text{ and } L(\beta) = \text{in}))$
- $\forall \alpha \in \mathcal{A} : (L(\alpha) = \text{in} \equiv \forall \beta \in \mathcal{A} : (\text{if } \beta \rightarrow \alpha \text{ then } L(\beta) = \text{out}))$

Otherwise, $L(\alpha) = \text{undec}$ (since L is a total function).

As it turns out, for any labeling satisfying the conditions above, those arguments that happen to be labeled in form a complete extension as per Definition 5.5.

If the argument graph contains cycles, there may be more than one complete extension (i.e., more than one legal labeling), each corresponding to a particular consistent and self-defending viewpoint. Consider the following.

Example 5.7 Consider the graph in Figure 5.2. Here, we have three complete extensions: $\{\alpha_3\}$, $\{\alpha_1, \alpha_3\}$, and $\{\alpha_2, \alpha_3\}$.

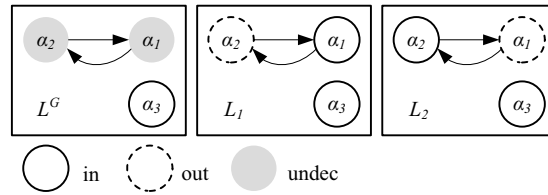


Figure 5.2: Graph with three labelings/complete extensions.

Now, we turn to further refinements of the notion of complete extension.

Definition 5.7 (Refinements of the complete extension) Let S be a conflict-free set of arguments in framework $\langle \mathcal{A}, \rightarrow \rangle$.

- S is a grounded extension if it is the minimal (w.r.t. set-inclusion) complete extension (or, alternatively, if S is the least fixed-point of $\mathcal{F}(\cdot)$).

- S is a preferred extension if it is a maximal (w.r.t. set-inclusion) complete extension (or, alternatively, if S is a maximal admissible set).
- S is a stable extension if $S^+ = \mathcal{A} \setminus S$.
- S is a semi-stable extension if S is a complete extension of which $S \cup S^+$ is maximal.

A *grounded extension* contains all the arguments which are not defeated, as well as the arguments which are defended directly or indirectly by non-defeated arguments. This can be seen as a non-committal view (characterized by the *least* fixed point of \mathcal{F}). As such, there always exists a unique grounded extension. Dung [12] showed that in finite argumentation systems, the grounded extension can be obtained by an iterative application of the characteriztic function \mathcal{F} to the empty set. For example, in Figure 5.1 the grounded extension (and the only complete extension) can be obtained as follows:

- $\mathcal{F}^1(\emptyset) = \{\alpha_3, \alpha_5\};$
- $\mathcal{F}^2(\emptyset) = \mathcal{F}(\mathcal{F}^1(\emptyset)) = \{\alpha_1, \alpha_3, \alpha_5\};$
- $\mathcal{F}^3(\emptyset) = \mathcal{F}(\mathcal{F}^2(\emptyset)) = \{\alpha_1, \alpha_3, \alpha_5\} = \mathcal{F}^2(\emptyset).$

Similarly, in Figure 5.2, the grounded extension is $\{\alpha_3\}$, which is the minimal complete extension w.r.t. set inclusion.

A *preferred extension* is a bolder, more committed position that cannot be extended – by accepting more arguments – without causing inconsistency. Thus a preferred extension can be thought of as a maximal consistent set of hypotheses. There may be multiple preferred extensions, and the grounded extension is included in all of them.

Example 5.8 In Figure 5.1, $\{\alpha_1, \alpha_3, \alpha_5\}$ is the only preferred extension. But in Figure 5.2, there are two preferred extensions: $\{\alpha_1, \alpha_3\}$ and $\{\alpha_2, \alpha_3\}$, which are the maximal complete extension w.r.t. set inclusion.

Finally, a set of arguments is a *stable extension* if it is a preferred extension that defeats every argument that does not belong to it. As expected, stable extensions may now always exist. An alternative definition is a *semi-stable extension*, which satisfies the weaker condition that the set of arguments defeated is maximal.

Caminada [10] established a correspondence between properties of labelings and the different extensions. Let $AF = \langle \mathcal{A}, \rightarrow \rangle$ be an argumentation framework, and L a labeling over AF . Define $\text{in}(L) = \{\alpha \in \mathcal{A} \mid L(\alpha) = \text{in}\}$; $\text{out}(L) = \{\alpha \in \mathcal{A} \mid L(\alpha) = \text{out}\}$; and $\text{undec}(L) = \{\alpha \in \mathcal{A} \mid L(\alpha) = \text{undec}\}$. These are summarized in Table 5.1.

Extensions	Restrictions on labelings
complete	all labelings
grounded	minimal in, or equivalently minimal out, or equivalently maximal undec
preferred	maximal in, or equivalently maximal out
semi-stable	minimal undec
stable	empty undec

Table 5.1: The relationships between extensions and labelings.

Now that the acceptability of sets of arguments is defined, we can define the status of any individual argument.

Definition 5.8 (Argument status) Let $\langle \mathcal{A}, \rightarrow \rangle$ be an argumentation system, and $\mathcal{E}_1, \dots, \mathcal{E}_n$ its extensions under a given semantics. Let $\alpha \in \mathcal{A}$.

1. α is skeptically accepted iff $\alpha \in \mathcal{E}_i, \forall \mathcal{E}_i$ with $i = 1, \dots, n$.
2. α is credulously accepted iff $\exists \mathcal{E}_i$ such that $\alpha \in \mathcal{E}_i$.
3. α is rejected iff $\nexists \mathcal{E}_i$ such that $\alpha \in \mathcal{E}_i$.

An argument is *skeptically accepted* if it belongs to all extensions under the adopted semantics. Intuitively, an argument is skeptically accepted if it can be accepted without making any hypotheses beyond what is surely self-defending. On the other hand, an argument is *credulously accepted* on the basis that it belongs to at least one extension. Intuitively, an argument is credulously accepted if there is a possible consistent set of hypotheses in which it is consistent. If an argument is neither skeptically nor credulously accepted, there is no basis for accepting it, and it is therefore *rejected*.

4 Argumentation Protocols

So far, I outlined some methods for evaluating an argument given an existing collection of arguments. In a multiagent system, however, the arguments are not all available a priori. Instead, they are presented by the agents during their argumentative dialogue. This raises the question of how such argumentation dialogues are to be regulated. For example, one should not be able to make statements that are completely irrelevant, or to contradict oneself. An argumentation protocol is, therefore, a set of rules that govern the argumentation process.

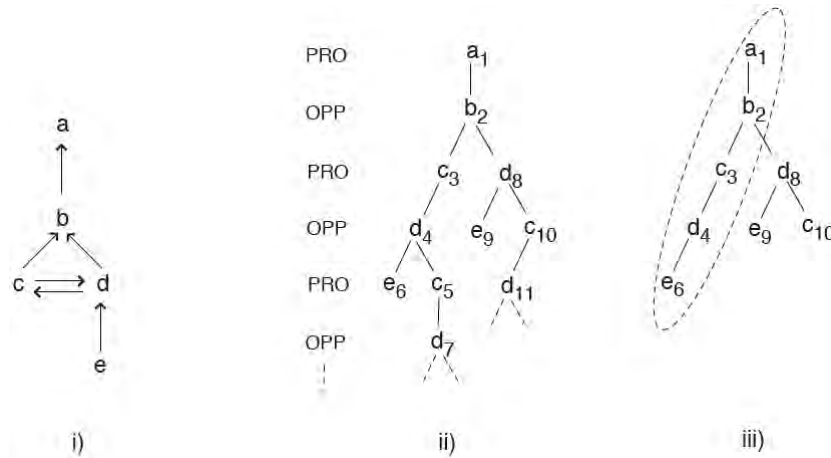


Figure 5.3: Argumentation framework and dispute tree. (i) shows an argumentation framework, (ii) shows the dispute tree induced in a , and (iii) shows the dispute tree induced by a under protocol G , with the winning strategy encircled.

4.1 Abstract Argument Games

Rules for governing argumentation can be specified without regard to the specific internal structure of the arguments. These protocols typically assume the presence of two agents, one PRO (the proponent) and the other OPP (the opponent). Dialogue begins with PRO asserting an argument x . Then PRO and OPP take turns, in a sequence of moves called a *dispute*, where each player makes an argument that attacks its counterpart's last move. A player wins a dispute if its counterpart cannot make a counterattack. But the counterpart may try a different line of attack, creating a new dispute. This results in a *dispute tree* structure that represents the dialogue.

The idea is that there is an implicit abstract argument graph, distributed in the agents' heads, so to speak. By following a specific protocol, the two agents arrive at an outcome (about the status of a particular argument) that corresponds to a particular semantics over the implicit graph.

Example 5.9 Consider two agents arguing the argument graph shown in Figure 5.3, taken from [30]. In Figure 5.3(i), we see the underlying (usually implicit) argument graph. Figure 5.3(ii) shows the corresponding dispute tree in which PRO presents argument a , OPP counters with argument b , PRO then counters once with argument c and once with argument d , and so on. Note that this dispute tree is infinite, since agents are able to repeat counterarguments due to the presence of cycles in the argument graph. As shown in the figure, arguments in the dispute tree can be indexed to capture repetition of the same argument from the graph.

We now present a definition of *winning strategy* adapted from [30].

Definition 5.9 (Winning strategy) *Given argument graph $\langle \mathcal{A}, \rhd \rangle$ and a dispute tree T with root a . A subtree T' is a winning strategy for a iff:*

1. *The set of disputes $D_{T'}$ in T' is a non-empty and finite set such that each $d \in D_{T'}$ is finite and is won by PRO (terminates in an argument moved by PRO).*
2. *$\forall d \in D_{T'}, \forall d'$ such that d' is a subdispute (i.e., sub-sequence with the same head) of d , and the last move in d' is argument x played by PRO, then for any y such that $y \rhd x$, there is $d'' \in D_{T'}$ such that d' appended with y is a subdispute of d'' .*

PRO is guaranteed to win if it plays the moves described in the winning strategy subtree. The second requirement ensures that all possible objections that can possibly be raised by OPP in one dispute can be neutralized successfully by PRO in the same or some other dispute in the subtree.

Note that creating a dispute tree (or subtree) only requires that every argument presented is a defeater of an argument that has already been presented. Adding further restrictions on the moves of different players can be captured by subtrees of the dispute tree. By adding such restrictions carefully, we can generate dialogue outcomes that correspond to well-known semantics. To illustrate the above, consider the following simple protocol:

Definition 5.10 (Protocol G) *Given argument graph $\langle \mathcal{A}, \rhd \rangle$, and a dispute D whose tail is argument x . Let $PRO(D)$ be the arguments uttered by PRO.*

- *If the dispute length is odd (next move is by OPP), then the possible next moves are $\{y \mid y \rhd x\}$.*
- *If the dispute length is even (next move is by PRO), then the possible next moves are $\{y \mid y \rhd x \text{ and } y \notin PRO(D)\}$.*

By simply prohibiting PRO from repeating himself, we ensure that the dispute tree is finite. A more striking consequence of this simple restriction is that PRO can only win if the argument at the root is in the grounded extension.

Theorem 5.1 ([30]) *Let $\langle \mathcal{A}, \rhd \rangle$ be an argument graph. There exists a winning strategy T for x under protocol G such that the set of arguments uttered by PRO in T is conflict-free, if and only if x is in the grounded extension of $\langle \mathcal{A}, \rhd \rangle$.*

Going back to Figure 5.3, note that the grounded extension of the graph in Figure 5.3(i) is $\{a, c, e\}$. Figure 5.3(iii) shows the dispute tree induced by argument a under protocol G , with the winning strategy $\{(a_1 - b_2 - c_3 - d_4 - e_6)\}$.

As it turns out, it is possible to use different protocol restrictions in order to capture different semantics. Suppose we wish to create dialogues in which PRO wins if and only if the argument in question is in at least one preferred extension (i.e., is credulously accepted under preferred semantics, recalling Definition 5.8). This can be achieved by a protocol in which two restrictions apply. OPP is not allowed to repeat its arguments. And PRO is allowed to repeat its arguments but cannot present a self-attacking argument, or an argument that conflicts (attacks or is attacked by) with another argument it already stated in the dispute. For a more comprehensive discussion of abstract argument games, including protocols that implement other semantics, refer to [30].

4.2 Dialogue Systems

Another approach to specifying argumentation protocols is the so-called *dialogue systems* approach (also known as *dialogue games*). This approach was initiated by Australian logician Charles L. Hamblin [19]. Unlike abstract argument games, this approach relies on having the explicit contents of the arguments presented.

As an example, Table 5.2 shows a specification of a persuasion protocol introduced by Prakken [37]. The left column shows the list of *speech acts* that can be used by agents (the notion of speech acts dates back to Searle [48]). An agent can assert a proposition by uttering “*claim* ϕ ”, and can retract its own claims by uttering “*retract* ϕ ”. An agent can also concede a proposition claimed by its counterpart by uttering “*concede* ϕ ”, or challenge such a claim by uttering “*why* ϕ ”. Finally, an agent can support its claims, if challenged, by uttering “ ϕ *since* S ”. With each speech act, Prakken associates two legal replies, one corresponding to surrender (e.g., by conceding to a claim made by the counterpart), and the other corresponding to attack (e.g., by challenging a claim or elements of an argument S presented by the counterpart).

This protocol generates a tree structure in which each utterance (i.e., move) is a node, and its possible responses are its children. Termination is defined in terms of the *dialogical status* of each move. A move is *in* if it is surrendered or if all its attacking replies are *out*. Conversely, a move is *out* if it has a reply that is *in*.¹ Whether the proponent or the opponent wins depends on whether the initial move is *in* or *out*. It is possible to also impose the requirement that moves must be *relevant*, meaning it would make its speaker the current winner. To illustrate,

¹Note that assigning dialogical status bears resemblance (but is not identical) to the labeling of abstract argument graphs discussed in Section 3.

Acts	Intuitive Meaning	Attacks	Surrenders
<i>claim</i> ϕ	Assert ϕ is true	<i>why</i> ϕ	<i>concede</i> ϕ
ϕ <i>since</i> S	Support ϕ by argument S	<i>why</i> ψ ($\psi \in S$) ϕ' <i>since</i> S' (defeats ϕ <i>since</i> S)	<i>concede</i> ψ ($\psi \in S$) <i>concede</i> ϕ
<i>why</i> ϕ	Challenge ϕ	ϕ <i>since</i> S	<i>retract</i> ϕ
<i>concede</i> ϕ	Concede ϕ claimed by other		
<i>retract</i> ϕ	Take back own claim ϕ		

Table 5.2: An example L_c in Prakken's framework [35].

suppose agent P has knowledge base $\{p, p \rightarrow_{r_1} q, p \rightarrow_{r_2} r, p \wedge s \rightarrow_{r_3} r_2 > r_4\}$ and agent O has knowledge base $\{t, t \rightarrow_{r_4} \neg r\}$. These knowledge bases are specified in Prakken and Sartor's argument-based, prioritized, extended logic programming language, in which rules are annotated, allowing for rules that support preferences between other rules [38].² The following dialogue is consistent with the above protocol [37], with the target of each move indicated between square brackets:

$P_1[-]:$ <i>claim</i> r $P_3[O_2]:$ r <i>since</i> $q, q \Rightarrow r$ $P_5[O_4]:$ q <i>since</i> $p, p \Rightarrow q$	$O_2[P_1]:$ <i>why</i> r $O_4[P_3]:$ <i>why</i> q $O_6[P_5]:$ <i>concede</i> $p \Rightarrow q$ $O_7[P_5]:$ <i>why</i> p
--	--

Note that at this point, player P has many possible moves. It can retract its claim or premises of its argument, or give an argument in support of p . It was also possible for player O to make the following against P 's original claim:

$O_7[P_3]: \neg r$ *since* $t, t \Rightarrow \neg r$

To this, P may respond with a priority argument, supporting the claim that rule r_2 takes precedence over rule r_4 , thus showing that P_3 strictly defeats O_7 :

$P_8[O_7]: r_2 > r_4$ *since* $p, s, p \wedge s \Rightarrow r_2 > r_4$

At this point, P_1 is *in*, but the dialogue may continue, with O conceding or making further challenges, and so on.

Various other dialogue systems have been presented in the literature, such as Walton and Krabbe's PDD [57] and McBurney and Parsons's Agent Dialogue Framework [28]. A longer discussion and comparison of different dialogue systems for persuasion was presented by Prakken [35].

The above approaches are related to the earlier work on so-called *game seman-*

²Note that these implications are not classical, thus they do not satisfy contraposition.

tics for logic, which was pioneered by logicians such as Paul Lorenzen [24] and Jaakko Hintikka [20]. Although many specific instantiations of this notion have been presented in the literature, the general idea is as follows. Given some specific logic, the truth value of a formula is determined through a special-purpose, multi-stage dialogue game between two players, the *verifier* and *falsifier*. The formula is considered true precisely when the verifier has a winning strategy, while it will be false whenever the falsifier has the winning strategy. Similar ideas have been used to implement dialectical proof-theories for defeasible reasoning (e.g., by Prakken and Sartor [38]).

It is worth mentioning that, in addition to the generic protocols presented in the last two sections, various domain-specific protocols have appeared in the literature. These protocols are usually more complex, and are linked to argumentation schemes from specific domains, enabling agents to argue about what action to take [1], the risks involved in various decisions [27], or about the properties of items involved in negotiation [29].

5 Strategic Argumentation and Game Theory

In all of the approaches above, we saw a specification of the argumentation protocol, specifying the set of possible moves that can be made by the agents. When it comes to the behavior of the agents who use this protocol, some protocols were extremely prescriptive, giving exactly a single option at a time. Other protocols (e.g., Prakken's persuasion protocol) gave agents some choice of what to do. The behavior of an agent, its so-called *strategy*, significantly influences the outcome of the dialogue (e.g., who wins), as well as its dynamics (e.g., whether it will terminate in a short number of moves).

To address this issue, researchers started exploring strategies for agents to choose their next moves. For example, in their dialogue system, Parsons et al. [32] defined a number of so-called *attitudes* of an agent, which specify the criteria according to which it evaluates and asserts arguments. These attitudes assume the availability of a preference relation over arguments, which can be used to compare their relative strength. In terms of asserting arguments, a *confident* agent can assert any proposition for which it can construct an argument, while a *careful* agent can do so only if it can construct such an argument and cannot construct a stronger counterargument. A *thoughtful* agent, on the other hand, can assert a proposition only if it can construct an acceptable argument for the proposition. Conversely, when it comes to evaluating arguments presented by the opponent, an agent can have one of the following attitudes. A *credulous* agent accepts a proposition if it can construct an argument for it, while a *cautious* agent does so only if it is also unable to construct a stronger counterargument. A *skeptical* agent,

however, accepts an argument only if it can construct an acceptable argument for the proposition.

Another approach to designing heuristics is to use aspects related to social constructs such as rights and obligations [21], or mental states of agents such as their beliefs, desires, and intentions [22]. A more detailed discussion of some of these issues can be found elsewhere [45].

While heuristic approaches to designing argumentation strategies are informative, they typically only address a subset of the possible strategies. A more comprehensive study of strategic argumentation must rely on a more systematic approach, and the appropriate framework for doing so is provided by the theory of games (or *game theory*), which was pioneered by von Neuman and Morgenstern [55]. A setting of strategic interaction is modeled as a *game*, which consists of a set of players, a set of actions available to them, and a rule that determines the outcome given players' chosen actions. In an argumentation scenario, the set of actions are typically the set of argumentative moves (e.g., asserting a claim or challenging a claim), and the outcome rule is the criterion by which arguments are evaluated (e.g., the judge's attitude). Generally, game theory can be used to achieve two goals:

1. undertake precise analysis of interaction in particular strategic settings, with a view to predicting the outcome;
2. design rules of the game in such a way that self-interested agents behave in some desirable way (e.g., tell the truth); this is called *mechanism design*.

Both these approaches are quite useful for the study of argumentation in multi-agent systems. On the one hand, an agent may use game theory to analyze a given argumentative situation in order to choose the best strategy. On the other hand, we may use mechanism design to design the rules (e.g., argumentation protocol) in such a way as to promote good argumentative behavior.

5.1 Glazer and Rubinstein's Model

One of the earliest attempts at game-theoretic analysis of argumentation was presented by microeconomists Glazer and Rubinstein [17]. The authors explore the mechanism design problem of constructing rules of debate that maximize the probability that a listener reaches the right conclusion given arguments presented by two debaters. They study a very restricted setting, in which the world state is described by a vector $\omega = (w_1, \dots, w_5)$, where each "aspect" w_i has two possible values: 1 and 2. If $w_i = j$ for $j \in \{1, 2\}$, we say that aspect w_i supports outcome O_j . Presenting an argument amounts to revealing the value of some w_i . The setting is modeled as an extensive-form game and analyzed. In particular,

the authors investigate various combinations of *procedural rules* (stating in which order and what sorts of arguments each debater is allowed to state) and *persuasion rules* (stating how the outcome is chosen by the listener). In terms of procedural rules, the authors explore: (1) *one-speaker debate* in which one debater chooses two arguments to reveal; (2) *simultaneous debate* in which the two debaters simultaneously reveal one argument each; and (3) *sequential debate* in which one debater reveals one argument followed by one argument by the other. Glazer and Rubinstein investigate a variety of persuasion rules. For example, in one-speaker debate, one rule analyzed by the authors states that “a speaker wins if and only if he presents two arguments from $\{a_1, a_2, a_3\}$ or $\{a_4, a_5\}$.” In a sequential debate, one persuasion rule states that “if debater D_1 argues for aspect a_3 , then debater D_2 wins if and only if he counter-argues with aspect a_4 .”

Note that these kinds of rules are arbitrary and do not follow an intuitive notion of persuasion (e.g., like skepticism). Hence, there is no concept of how the logical structure of the information presented by different players constrains the decision made by the judge. In the remainder of this section, I present a game-theoretic account of strategic argumentation based on abstract argument graphs.

5.2 Game Theory Background

This section gives a brief background on key game-theoretic concepts [25]. Readers already familiar with game theory may skip this section. The field of game theory studies strategic interactions of self-interested agents. We assume that there is a set of self-interested agents, denoted by I . We let $\theta_i \in \Theta_i$ denote the *type* of agent i , which is drawn from some set of possible types Θ_i . The type represents the private information and preferences of the agent. An agent’s preferences are over *outcomes* $o \in \mathcal{O}$, where \mathcal{O} is the set of all possible outcomes. We assume that an agent’s preferences can be expressed by a utility function $u_i(o, \theta_i)$, which depends on both the outcome, o , and the agent’s type, θ_i . Agent i prefers outcome o_1 to o_2 when $u_i(o_1, \theta_i) > u_i(o_2, \theta_i)$.

When agents interact, we say that they are playing *strategies*. A strategy for agent i , $s_i(\theta_i)$, is a plan that describes what actions the agent will take for every decision that the agent might be called upon to make, for each possible piece of information that the agent may have at each time it is called to act. That is, a strategy can be thought of as a complete contingency plan for an agent. We let Σ_i denote the set of all possible strategies for agent i , and thus $s_i(\theta_i) \in \Sigma_i$. When it is clear from the context, we will drop the θ_i in order to simplify the notation. We let *strategy profile* $s = (s_1(\theta_1), \dots, s_I(\theta_I))$ denote the outcome that results when each agent i is playing strategy $s_i(\theta_i)$. As a notational convenience we define

$$s_{-i}(\theta_{-i}) = (s_1(\theta_1), \dots, s_{i-1}(\theta_{i-1}), s_{i+1}(\theta_{i+1}), \dots, s_I(\theta_I))$$

and thus $s = (s_i, s_{-i})$. We then interpret $u_i((s_i, s_{-i}), \theta_i)$ to be the utility of agent i with type θ_i when all agents play strategies specified by strategy profile $(s_i(\theta_i), s_{-i}(\theta_{-i}))$. Similarly, we also define:

$$\theta_{-i} = (\theta_1, \dots, \theta_{i-1}, \theta_{i+1}, \dots, \theta_I)$$

Since the agents are all self-interested, they will try to choose strategies that maximize their own utility. Since the strategies of other agents also play a role in determining the outcome, the agents must take this into account. The *solution concepts* in game theory determine the outcomes that will arise if all agents are rational and strategic. The most well-known solution concept is the *Nash equilibrium*. A Nash equilibrium is a strategy profile in which each agent is following a strategy that maximizes its own utility, given its type and the strategies of the other agents.

Definition 5.11 (Nash equilibrium) A strategy profile $s^* = (s_1^*, \dots, s_I^*)$ is a Nash equilibrium if no agent has an incentive to change its strategy, given that no other agent changes. Formally, $\forall i, \forall s'_i, u_i(s_i^*, s_{-i}^*, \theta_i) \geq u_i(s'_i, s_{-i}^*, \theta_i)$

Although the Nash equilibrium is a fundamental concept in game theory, it does have several weaknesses. First, there may be multiple Nash equilibria and so agents may be uncertain as to which equilibrium they should play. Second, the Nash equilibrium implicitly assumes that agents have perfect information about all other agents, including the other agents' preferences.

A stronger solution concept in game theory is the *dominant-strategy equilibrium*. A strategy s_i is said to be *dominant* if by playing it, the utility of agent i is maximized no matter what strategies the other agents play.

Definition 5.12 (Dominant strategy) A strategy s_i^* is dominant if

$$\forall s_{-i}, \forall s'_i, u_i(s_i^*, s_{-i}, \theta_i) \geq u_i(s'_i, s_{-i}, \theta_i).$$

Sometimes, we will refer to a strategy satisfying the above definition as *weakly dominant*. If the inequality is strict (i.e., $>$ instead of \geq), we say that the strategy is *strictly dominant*. A *dominant-strategy equilibrium* is a strategy profile where each agent is playing a dominant strategy. This is a very robust solution concept since it makes no assumptions about what information the agents have available to them, nor does it assume that all agents know that all other agents are being rational (i.e., trying to maximize their own utility). However, there are many strategic settings where no agent has a dominant strategy.

5.2.1 Mechanism Design

The problem that mechanism design studies is how to ensure that a desirable system-wide outcome or decision is made when there is a group of self-interested agents who have preferences over the outcomes. In particular, we often want the outcome to depend on the preferences of the agents. This is captured by a *social choice function*.

Definition 5.13 (Social choice function) A social choice function is a rule $f : \Theta_1 \times \dots \times \Theta_I \rightarrow \mathcal{O}$, that selects some outcome $f(\theta) \in \mathcal{O}$, given agent types $\theta = (\theta_1, \dots, \theta_I)$.

The challenge, however, is that the types of the agents (the θ_i 's) are private and known only to the agents themselves. Thus, in order to select an outcome with the social choice function, one has to rely on the agents to reveal their types. However, for a given social choice function, an agent may find that it is better off if it does not reveal its type truthfully, since by lying it may be able to cause the social choice function to choose an outcome that it prefers. Instead of trusting the agents to be truthful, we use a *mechanism* to try to reach the correct outcome.

A mechanism $\mathcal{M} = (\Sigma, g(\cdot))$ defines the set of allowable strategies that agents can choose, with $\Sigma = \Sigma_1 \times \dots \times \Sigma_I$ where Σ_i is the strategy set for agent i , and an outcome function $g(s)$ that specifies an outcome o for each possible strategy profile $s = (s_1, \dots, s_I) \in \Sigma$. This defines a game in which agent i is free to select any strategy in Σ_i , and, in particular, will try to select a strategy that will lead to an outcome that maximizes its own utility. We say that a mechanism *implements* social choice function f if the outcome induced by the mechanism is the same outcome that the social choice function would have returned if the true types of the agents were known.

Definition 5.14 (Implementation) Mechanism $\mathcal{M} = (\Sigma, g(\cdot))$ implements social choice function f if there exists an equilibrium s^* s.t. $\forall \theta \in \Theta, g(s^*(\theta)) = f(\theta)$.

While the definition of a mechanism puts no restrictions on the strategy spaces of the agents, an important class of mechanisms are the *direct-revelation mechanisms* (or simply *direct mechanisms*).

Definition 5.15 (Direct-revelation mechanism) A direct-revelation mechanism is a mechanism in which $\Sigma_i = \Theta_i$ for all i , and $g(\theta) = f(\theta)$ for all $\theta \in \Theta$.

In other words, a direct mechanism is one where the strategies of the agents are to announce a type, θ_i' , to the mechanism. While it is not necessary that $\theta_i' = \theta_i$, the important *revelation principle* (see below for more details) states that if a social choice function, $f(\cdot)$, can be implemented, then it can be implemented by a direct

mechanism where every agent reveals its true type [25]. In such a situation, we say that the social choice function is *incentive compatible*.

Definition 5.16 (Incentive compatible) *The social choice function $f(\cdot)$ is incentive compatible (or truthfully implementable) if the direct mechanism $\mathcal{M} = (\Theta, g(\cdot))$ has an equilibrium s^* such that $s_i^*(\theta_i) = \theta_i$.*

If the equilibrium concept is the dominant-strategy equilibrium, then the social choice function is *strategyproof*. In this chapter we will on occasion call a mechanism incentive compatible or strategyproof. This means that the social choice function that the mechanism implements is incentive compatible or strategyproof.

5.2.2 The Revelation Principle

Determining whether a particular social choice function can be implemented, and in particular, finding a mechanism that implements a social choice function appears to be a daunting task. In the definition of a mechanism, the strategy spaces of the agents are unrestricted, leading to an infinitely large space of possible mechanisms. However, the *revelation principle* states that we can limit our search to a special class of mechanisms [25, Ch. 14].

Theorem 5.2 (Revelation principle) *If there exists some mechanism that implements social choice function f in dominant strategies, then there exists a direct mechanism that implements f in dominant strategies and is truthful.*

The intuitive idea behind the revelation principle is fairly straightforward. Suppose that you have a possibly very complex mechanism, \mathcal{M} , which implements some social choice function, f . That is, given agent types $\theta = (\theta_1, \dots, \theta_I)$ there exists an equilibrium $s^*(\theta)$ such that $g(s^*(\theta)) = f(\theta)$. Then, the revelation principle states that it is possible to create a new mechanism, \mathcal{M}' , which, when given θ , will then execute $s^*(\theta)$ on behalf of the agents and then select outcome $g(s^*(\theta))$. Thus, each agent is best off revealing θ_i , resulting in \mathcal{M}' being a truthful, direct mechanism for implementing social choice function f .

The revelation principle is a powerful tool when it comes to studying implementation. Instead of searching through the entire space of mechanisms to check whether one implements a particular social choice function, the revelation principle states that we can restrict our search to the class of truthful, direct mechanisms. If we cannot find a mechanism in this space that implements the social choice function of interest, then there does not exist any mechanism that will do so.

5.3 Argumentation Mechanism Design

In this section I define the mechanism design problem for abstract argumentation, leading to so-called *argumentation mechanism design* (ArgMD) (this work was introduced in [39, 42]).

Let $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework with a set of arguments \mathcal{A} and a binary defeat relation \mathcal{R} . I define a mechanism with respect to AF and semantics \mathcal{S} , and I assume that there is a set of I self-interested agents. I define an agent's type to be its set of arguments.

Definition 5.17 (Agent type) *Given an argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$, the type of agent i , $\mathcal{A}_i \subseteq \mathcal{A}$, is the set of arguments that the agent is capable of putting forward.*

There are two things to note about this definition. First, an agent's type can be seen as a reflection of its expertise or domain knowledge. For example, medical experts may only be able to comment on certain aspects of forensics in a legal case, whereas a defendant's family and friends may be able to comment on its character. Also, such expertise may overlap, so agent types are not necessarily disjoint. For example, two medical doctors might have some identical argument.

The second thing to note about the definition is that agent types do not include the defeat relation. In other words, I implicitly assume that the notion of defeat is common to all agents. That is, given two arguments, no agent would dispute whether one attacks another. This is a reasonable assumption in systems where agents use the same logic to express arguments or at least multiple logics for which the notion of defeat is accepted by everyone (e.g., conflict between a proposition and its negation). Disagreement over the defeat relation itself requires a form of hierarchical (meta) argumentation, which is a powerful concept, but can be reduced to a standard argument graph [31].

Given the agents' types (argument sets), a social choice function f maps a type profile into a subset of arguments: $f : 2^{\mathcal{A}} \times \dots \times 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$. Denote by $Acc(\langle \mathcal{A}, \mathcal{R} \rangle, \mathcal{S}) \subseteq \mathcal{A}$ the set of acceptable arguments according to semantics \mathcal{S} .³ It is possible to define *argument acceptability* social choice functions.

Definition 5.18 (Argument acceptability social choice functions) *Given an argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$ with semantics \mathcal{S} , and given an agent type profile $(\mathcal{A}_1, \dots, \mathcal{A}_I)$, the argument acceptability social choice function f is defined as the set of acceptable arguments given the semantics \mathcal{S} . That is,*

$$f(\mathcal{A}_1, \dots, \mathcal{A}_I) = Acc(\langle \mathcal{A}_1 \cup \dots \cup \mathcal{A}_I, \mathcal{R} \rangle, \mathcal{S}).$$

³Here, assume that \mathcal{S} specifies both the classical semantics used (e.g., grounded, preferred, stable) as well as the acceptance attitude used (e.g., skeptical or credulous).

As is standard in the mechanism design literature, assume that agents have preferences over the outcomes $o \in 2^A$, and we represent these preferences using utility functions where $u_i(o, \mathcal{A}_i)$ denotes agent i 's utility for outcome o when its type is argument set \mathcal{A}_i .

Agents may not have an incentive to reveal their true type because they may be able to influence the final argument status assignment by lying, and thus obtain higher utility. There are two ways that an agent can lie in our model. On the one hand, an agent might create new arguments that it does not have in its argument set. In the rest of the chapter I will assume that there is an *external verifier* that is capable of checking whether it is possible for a particular agent to actually make a particular argument. Informally, this means that presented arguments, while still possibly defeasible, must at least be based on some sort of demonstrable “plausible evidence.” If an agent is caught making up arguments, then it will be removed from the mechanism. For example, in a court of law, any act of perjury by a witness is punished, at the very least, by completely discrediting all evidence produced by the witness. Moreover, in a court of law, arguments presented without any plausible evidence are normally discarded (e.g., “*I did not kill him, since I was abducted by aliens at the time of the crime!*”). For all intents and purposes this assumption (also made by Glazer and Rubinstein [17]) removes the incentive for an agent to make up facts.

A more insidious form of manipulation occurs when an agent decides to *hide* some of its arguments. By refusing to reveal certain arguments, an agent might be able to break defeat chains in the argument framework, thus changing the final set of acceptable arguments. For example, a witness may hide evidence that implicates the defendant if the evidence also undermines the witness's own character. I will focus on this form of lying in this chapter.

As mentioned earlier, a strategy of an agent specifies a complete plan that describes what action the agent takes for every decision that a player might be called upon to make, for every piece of information that the player might have at each time that it is called upon to act. In our model, the actions available to an agent involve announcing sets of arguments. Thus a strategy $s_i \in \Sigma_i$ for agent i would specify for each possible subset of arguments that could define its type, what set of arguments to reveal. For example, a strategy might specify that an agent should reveal only half of its arguments without waiting to see what other agents are going to do, while another strategy might specify that an agent should wait and see what arguments are revealed by others, before deciding how to respond. We can therefore define an argumentation mechanism.

Definition 5.19 (Argumentation mechanism) *Given an argumentation framework $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ and semantics \mathcal{S} , an argumentation mechanism is defined as $\mathcal{M}_{AF}^{\mathcal{S}} = (\Sigma_1, \dots, \Sigma_I, g(\cdot))$, where Σ_i is an argumentation strategy space of agent i*

MD Concept	ArgMD Instantiation
Agent type $\theta_i \in \Theta_i$	Agent's arguments $\theta_i = \mathcal{A}_i \subseteq \mathcal{A}$
Outcome $o \in \mathcal{O}$	Accepted arguments $Acc(\cdot) \subseteq \mathcal{A}$
Utility $u_i(o, \theta_i)$	Preferences over $2^{\mathcal{A}}$ (what arguments end up being accepted)
Social choice function $f : \Theta_1 \times \dots \times \Theta_I \rightarrow \mathcal{O}$	$f(\mathcal{A}_1, \dots, \mathcal{A}_I) = Acc(\langle \mathcal{A}_1 \cup \dots \cup \mathcal{A}_I, \mathcal{R} \rangle, \mathcal{S})$ by some argument acceptability criterion
Mechanism $\mathcal{M} = (\Sigma, g(\cdot))$ where $\Sigma = \Sigma_1 \times \dots \times \Sigma_I$ and $g : \Sigma \rightarrow \mathcal{O}$	Σ_i is an argumentation strategy, $g : \Sigma \rightarrow 2^{\mathcal{A}}$
Direct mechanism: $\Sigma_i = \Theta_i$	$\Sigma_i = 2^{\mathcal{A}}$ (every agent reveals a set of arguments)
Truth revelation	Revealing \mathcal{A}_i

Table 5.3: Abstract argumentation as a mechanism.

and $g : \Sigma_1 \times \dots \times \Sigma_I \rightarrow 2^{\mathcal{A}}$.

Note that in the above definition, the notion of dialogue strategy is broadly construed and would depend on the protocol used. In a *direct* mechanism, however, the strategy spaces of the agents are restricted so that they can only reveal a subset of arguments. Due to the revelation principle, this will be sufficient for the analysis in the rest of the chapter.

Definition 5.20 (Direct argumentation mechanism) *Given an argumentation framework $AF = \langle \mathcal{A}, \mathcal{R} \rangle$ and semantics \mathcal{S} , a direct argumentation mechanism is defined as $\mathcal{M}_{AF}^{\mathcal{S}} = (\Sigma_1, \dots, \Sigma_I, g(\cdot))$, where $\Sigma_i = 2^{\mathcal{A}}$ and $g : \Sigma_1 \times \dots \times \Sigma_I \rightarrow 2^{\mathcal{A}}$.*

Table 5.3 summarizes the mapping of multiagent abstract argumentation as an instance of a mechanism design problem [42].

5.4 Case Study: Implementing the Grounded Semantics

In this section, I demonstrate the power of our ArgMD approach by showing how it can be used to systematically analyze the strategic incentives imposed by a well-established argument evaluation criterion [39, 43]. In particular, I specify a direct-revelation argumentation mechanism, in which agents' strategies are to reveal sets of arguments, and where the mechanism calculates the outcome using skeptical (grounded) semantics.⁴ I show that, in general, this mechanism gives rise to strategic manipulation. However, under some conditions, this mechanism turns out to be strategyproof.

In a direct argumentation mechanism, each agent i 's available actions are $\Sigma_i = 2^{\mathcal{A}}$. We will refer to a specific action (i.e., set of declared arguments) as $\mathcal{A}_i^o \in \Sigma_i$. The following mechanism calculates the grounded extension given the union of all arguments revealed by agents.

⁴In the remainder of the chapter, I will use the term *skeptical* to refer to *skeptical grounded*.

Definition 5.21 (Grounded direct argumentation mechanism) A grounded direct argumentation mechanism for argumentation framework $\langle \mathcal{A}, \mathcal{R} \rangle$ is $\mathcal{M}_{AF}^{grnd} = (\Sigma_1, \dots, \Sigma_I, g(\cdot))$ where:

- $\Sigma_i \in 2^{\mathcal{A}}$ is the set of strategies available to each agent;
- $g : \Sigma_1 \times \dots \times \Sigma_I \rightarrow 2^{\mathcal{A}}$ is an outcome rule defined as: $g(\mathcal{A}_1^\circ, \dots, \mathcal{A}_I^\circ) = \text{Acc}(\langle \mathcal{A}_1^\circ \cup \dots \cup \mathcal{A}_I^\circ, \mathcal{R} \rangle, \mathcal{S}^{grnd})$ where \mathcal{S}^{grnd} denotes skeptical grounded acceptability semantics.

To simplify our analysis, let us assume that agents can only lie by hiding arguments, and not by making up arguments. Formally, this means that $\forall i, \Sigma_i \in 2^{\mathcal{A}_i}$. For the sake of illustration, consider a particular family of preferences that agents may have. According to these preferences, every agent attempts to maximize the number of arguments in \mathcal{A}_i that end up being accepted. We call this preference criteria the *individual acceptability maximizing preference*.

Definition 5.22 (Acceptability maximizing preferences) An agent i has individual acceptability maximizing preferences if and only if $\forall o_1, o_2 \in \mathcal{O}$ such that $|o_1 \cap \mathcal{A}_i| \geq |o_2 \cap \mathcal{A}_i|$, we have $u_i(o_1, \mathcal{A}_i) \geq u_i(o_2, \mathcal{A}_i)$.

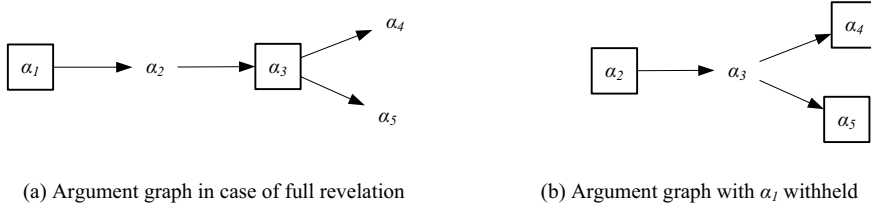


Figure 5.4: Hiding an argument is beneficial (case of acceptability maximizers).

The following example explores incentives with mechanism \mathcal{M}_{AF}^{grnd} .

Example 5.10 Consider grounded direct argumentation mechanism with three agents x , y , and z with types $\mathcal{A}_x = \{\alpha_1, \alpha_4, \alpha_5\}$, $\mathcal{A}_y = \{\alpha_2\}$, and $\mathcal{A}_z = \{\alpha_3\}$, respectively. And suppose that the defeat relation is defined as follows: $\mathcal{R} = \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_3), (\alpha_3, \alpha_4), (\alpha_3, \alpha_5)\}$. If each agent reveals its true type (i.e., $\mathcal{A}_x^\circ = \mathcal{A}_x$; $\mathcal{A}_y^\circ = \mathcal{A}_y$; and $\mathcal{A}_z^\circ = \mathcal{A}_z$), then we get the argument graph depicted in Figure 5.4(a). The mechanism outcome rule produces the outcome $o = \{\alpha_1, \alpha_3\}$. If agents have individual acceptability maximizing preferences, with utilities equal to the number of arguments accepted, then: $u_x(o, \{\alpha_1, \alpha_4, \alpha_5\}) = 1$; $u_y(o, \{\alpha_3\}) = 1$; and $u_z(o, \{\alpha_2\}) = 0$.

It turns out that the mechanism is susceptible to strategic manipulation, even if we suppose that agents do not lie by making up arguments (i.e., they may only withhold some arguments). In this case, for both agents y and z , revealing their true types weakly dominates revealing nothing at all. However, it turns out that agent x is better off revealing $\{\alpha_4, \alpha_5\}$. By withholding α_1 , the resulting argument network becomes as depicted in Figure 5.4(b), for which the output rule produces the outcome $o' = \{\alpha_2, \alpha_4, \alpha_5\}$. This outcome yields utility 2 to agent x , which is better than the truth-revealing strategy. Thus, given an arbitrary argumentation framework AF and agents with acceptability maximizing preferences, mechanism \mathcal{M}_{AF}^{grnd} is *not* strategyproof.

The following theorem provides a full characterization of strategyproof mechanisms for skeptical argumentation frameworks for agents with acceptability maximizing preferences.

Theorem 5.3 *Let AF be an arbitrary argumentation framework, and let $\mathcal{E}_{\mathcal{GR}}(AF)$ denote its grounded extension. Mechanism \mathcal{M}_{AF}^{grnd} is strategyproof for agents with acceptability maximizing preferences if and only if AF satisfies the following condition: $\forall i \in I, \forall S \subseteq \mathcal{A}_i$ and $\forall \mathcal{A}_{-i}$, we have $|\mathcal{A}_i \cap \mathcal{E}_{\mathcal{GR}}(\langle \mathcal{A}_i \cup \mathcal{A}_{-i}, \mathcal{R} \rangle)| \geq |\mathcal{A}_i \cap \mathcal{E}_{\mathcal{GR}}(\langle (\mathcal{A}_i \setminus S) \cup \mathcal{A}_{-i}, \mathcal{R} \rangle)|$.*

Although the theorem gives us a full characterization, it is difficult to apply in practice. In particular, the theorem does not give us an indication of how agents (or the mechanism designer) can identify whether the mechanism is strategyproof for a class of argumentation frameworks by appealing to their graph-theoretic properties. Below, we provide an intuitive, graph-theoretic condition that is sufficient to ensure that \mathcal{M}_{AF}^{grnd} is strategyproof when agents have focal arguments.

Let $\alpha, \beta \in \mathcal{A}$. We say that α *indirectly defeats* β , written $\alpha \hookrightarrow \beta$, if and only if there is an odd-length path from α to β in the argument graph.

Theorem 5.4 *Suppose agents have individual acceptability maximizing preferences. If each agent's type corresponds to a conflict-free set of arguments that does not include (in)direct defeats (formally $\forall i \nexists \alpha_1, \alpha_2 \in \mathcal{A}_i$ such that $\alpha_1 \hookrightarrow \alpha_2$), then \mathcal{M}_{AF}^{grnd} is strategyproof.*

Note that \hookrightarrow is over all arguments in \mathcal{A} . Intuitively, the condition in the theorem states that *all* arguments of every agent must be conflict-free (i.e., consistent), both explicitly and implicitly. Explicit consistency implies that no argument defeats another. Implicit consistency implies that other agents cannot possibly present a set of arguments that reveal an indirect defeat among one's own arguments. More concretely, in Example 5.10 and Figure 5.4, while agent x 's argument set $\mathcal{A}_x = \{\alpha_1, \alpha_4, \alpha_5\}$ is conflict-free, when agents y and z presented their own ar-

guments α_2 and α_3 , they revealed an implicit conflict in x 's arguments. In other words, they showed that x contradicts itself (i.e., committed a *fallacy*).

In addition to characterizing a sufficient graph-theoretic condition for strategy-proofness, Theorem 5.4 is useful for individual agents. As long as the agent knows that it is *not* possible for a path to be created that causes an (in)direct defeat among its arguments (i.e., a fallacy to be revealed), then the agent is best off revealing all its arguments. The agent only needs to know that no argument imaginable can reveal conflicts among its own arguments.

Is the *sufficient* condition in Theorem 5.4 also *necessary* for revealing all arguments truthfully? Example 5.11 shows that this is not the case. In particular, for certain argumentation frameworks, an agent may have truth-telling as a dominant strategy despite the presence of indirect defeats among its own arguments.

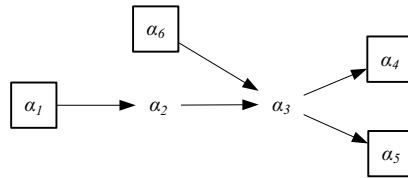


Figure 5.5: Argument graph (see Example 5.11).

Example 5.11 Consider the variant of Example 5.10 with the additional argument α_6 and defeat (α_6, α_3) . Let the agent types be $\mathcal{A}_x = \{\alpha_1, \alpha_4, \alpha_5, \alpha_6\}$, $\mathcal{A}_y = \{\alpha_2\}$, and $\mathcal{A}_z = \{\alpha_3\}$, respectively. The full argument graph is depicted in Figure 5.5. Under full revelation, the mechanism outcome rule produces the outcome $o = \{\alpha_1, \alpha_4, \alpha_5, \alpha_6\}$.

Note that in Example 5.11, truth revelation is now a dominant strategy for x (since it gets all its arguments accepted) despite the fact that $\alpha_1 \hookrightarrow \alpha_4$ and $\alpha_1 \hookrightarrow \alpha_5$. This hinges on the presence of an argument (namely α_5) that cancels out the negative effect of the (in)direct self-defeat among x 's own arguments.

6 The Argument Interchange Format

To facilitate argumentation among agents in an open system, it is essential to have a common language for argument representation. One community-led effort toward this is the *Argumentation Interchange Format (AIF)*. Here, I give a very brief overview of the AIF, and point the reader elsewhere for further details [11].

The core AIF has two types of nodes: *information nodes* (or *I-nodes*) and *scheme nodes* (or *S-nodes*). These are represented by two disjoint sets, $\mathcal{N}_I \subset \mathcal{N}$ and $\mathcal{N}_S \subset \mathcal{N}$, respectively. Information nodes are used to represent *passive* information contained in an argument, such as a claim, premise, data, etc. S-nodes capture the application of *schemes* (i.e., patterns of reasoning). Such schemes may be domain-independent patterns of reasoning, which resemble rules of inference in deductive logics but broadened to include non-deductive inference. The schemes themselves belong to a class, \mathcal{S} , and are classified into the types: *rule of inference scheme*, *conflict scheme*, and *preference scheme*. We denote these using the disjoint sets \mathcal{S}^R , \mathcal{S}^C , and \mathcal{S}^P , respectively. The predicate ($\text{uses} : \mathcal{N}_S \times \mathcal{S}$) is used to express the fact that a particular scheme node uses (or instantiates) a particular scheme. The AIF thus provides an ontology for expressing schemes and instances of schemes, and constrains the latter to the domain of the former via the function *uses*, i.e., $\forall n \in \mathcal{N}_S, \exists s \in \mathcal{S}$ such that $\text{uses}(n, s)$.

The present ontology has three different types of scheme nodes: *rule of inference application nodes* (or *RA-nodes*), *preference application nodes* (or *PA-nodes*) and *conflict application nodes* (or *CA-nodes*). These are represented as three disjoint sets: $\mathcal{N}_S^{RA} \subseteq \mathcal{N}_S$, $\mathcal{N}_S^{PA} \subseteq \mathcal{N}_S$, and $\mathcal{N}_S^{CA} \subseteq \mathcal{N}_S$, respectively. The word “application” on each of these types was introduced in the AIF as a reminder that these nodes function as instances, not classes, of possibly generic inference rules. Intuitively, \mathcal{N}_S^{RA} captures nodes that represent (possibly non-deductive) rules of inference, \mathcal{N}_S^{CA} captures applications of criteria (declarative specifications) defining conflict (e.g., among a proposition and its negation, etc.), and \mathcal{N}_S^{PA} are applications of (possibly abstract) criteria of preference among evaluated nodes.

The AIF specification does not type its edges. The (informal) semantics of edges can be inferred from the types of nodes they connect. One of the restrictions is that no outgoing edge from an I-node can be directed directly to another I-node. This ensures that the type of any relationship between two pieces of information must be specified explicitly via an intermediate S-node.

Definition 5.23 (Argument network) An argument network Φ is a graph with: (i) a set $\mathcal{N} = \mathcal{N}_I \cup \mathcal{N}_S$ of vertices (or nodes); and (ii) a binary relation $\xrightarrow{\text{edge}} : \mathcal{N} \times \mathcal{N}$ representing edges, where $\nexists (i, j) \in \xrightarrow{\text{edge}}$ where both $i \in \mathcal{N}_I$ and $j \in \mathcal{N}_I$.

A simple argument can be represented by linking a set of premises to a conclusion.

Definition 5.24 (Simple argument) A simple argument, in network Φ and schemes \mathcal{S} , is a tuple $\langle P, \tau, c \rangle$ where: (i) $P \subseteq \mathcal{N}_I$ is a set of nodes denoting premises; (ii) $\tau \in \mathcal{N}_S^{RA}$ is a rule of inference application node; (iii) $c \in \mathcal{N}_I$ is a node denoting the conclusion, such that $\tau \xrightarrow{\text{edge}} c$, $\text{uses}(\tau, s)$ where $s \in \mathcal{S}$, and $\forall p \in P$ we have $p \xrightarrow{\text{edge}} \tau$.

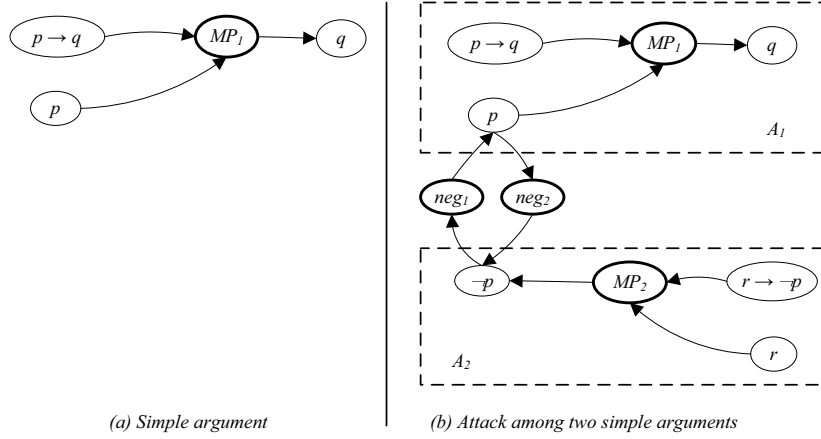


Figure 5.6: Examples of simple arguments. S-nodes denoted with a thicker border.

Following is a description of a simple argument in propositional logic, depicted in Figure 5.6(a).

Example 5.12 (Simple argument) The tuple $A_1 = \langle \{p, p \rightarrow q\}, MP_1, q \rangle$ is a simple argument in propositional language \mathcal{L} , where $p, (p \rightarrow q) \in \mathcal{N}_I$ are nodes representing premises, and $q \in \mathcal{N}_I$ is a node representing the conclusion. In between them, the node $MP_1 \in \mathcal{N}_S^{RA}$ is a rule of inference application node (i.e., RA-node) that uses the modus ponens natural deduction scheme, which can be formally written as follows: $\text{uses}(MP_1, \forall A, B \in \mathcal{L} \frac{A \quad A \rightarrow B}{B})$.

An attack or conflict from one information or scheme node to another information or scheme node is captured through a CA-node, which captures the type of conflict. The attacker is linked to the CA-node, and the CA-node is subsequently linked to the attacked node. Note that since edges are directed, each CA-node captures attack in one direction. Symmetric attack would require two CA-nodes, one in each direction. The following example describes a conflict between two simple arguments (see Figure 5.6(b)).

Example 5.13 (Simple arguments in conflict) Recall the simple argument $A_1 = \langle \{p, p \rightarrow q\}, MP_1, q \rangle$. And consider another simple argument $A_2 = \langle \{r, r \rightarrow \neg p\}, MP_2, \neg p \rangle$. Argument A_2 undermines A_1 by supporting the negation of the latter's premise. This (symmetric) propositional conflict is captured through two CA-nodes: neg_1 and neg_2 , both of which instantiate a conflict scheme based on propositional contraries.

Note that the AIF language is less abstract than Dung's abstract argument graphs, but is more abstract than approaches that describe the internal contents of

logical arguments. Moreover, the AIF was deliberately given only semi-formal semantics, allowing for it to be adapted according to one's need, for example with a particular language for describing the internal contents of information nodes, or by committing to edges with specific formal semantics. It has been shown that the AIF can be adapted for creating ontologies using Semantic Web standards for annotating natural language arguments using Walton-style schemes (see for example [47]). Having said that, the AIF is still a young effort in need of further refinement and proof-of-concept applications.

7 Conclusion

I gave an overview of the emerging field of argumentation in multiagent systems. I introduced some basic definitions of the argument and relationships between arguments. I then gave an overview of protocols that govern argumentation dialogues among agents, before moving to the issue of strategic argumentation. I closed the discussion with a brief overview of efforts toward a common ontology for enabling the exchange of arguments.

This field of study is still in its infancy. While much is understood about the properties of argumentation semantics [2] and the termination and complexity properties of dialogue protocols [13], strategic aspects are still underexplored. Game-theoretic tools have proven indispensable to the understanding of other forms of interaction in multiagent systems, such as auction and voting protocols, as illustrated elsewhere in this book. Yet the connection between argumentation processes and game theory still has a long way to go.

Another important challenge is understanding how computational models of argument relate to how people actually argue. This is crucial to the task of programming agents capable of arguing with, and successfully persuading people [26]. The models explored in this chapter mostly overlook questions of psychological plausibility [40]. There is an opportunity for cross-fertilization between computational argumentation and human reasoning research [50].

Acknowledgment

Some of the contents of this chapter are based on excerpts from my previous work. I would like to thank all my coauthors on those articles, especially Kate Larson and Chris Reed. I would also like to apologize for the omission of much important work, which I simply did not have space for due to a limited number of pages.

8 Exercises

1. **[Level 1]** Consider the argument framework $\langle \mathcal{A}, \rightarrow \rangle$ with $\mathcal{A} = \{a, b, c, d\}$ and $\rightarrow = \{(a, b), (b, a), (a, c), (b, c), (c, d), (d, c)\}$. Draw the argument graph, then produce all legal labelings of this graph. List all complete extensions, and identify which of these is grounded, preferred, stable (if it exists), and semi-stable.
2. **[Level 1]** Consider the argument framework $\langle \mathcal{A}, \rightarrow \rangle$ with $\mathcal{A} = \{a, b, c\}$ and $\rightarrow = \{(a, b), (b, c), (c, a)\}$. Draw the argument graph, then produce all legal labelings of this graph. List all complete extensions. Think about the peculiar nature of odd-length cycles in argument graphs.
3. **[Level 1]** Pick an argument from today's newspaper, and try to model it using the argument interchange format. Are there multiple ways to do this?
4. **[Level 2]** Consider the following situation involving the couple Alice (A) and Brian (B), who want to decide on an activity for the day. Brian thinks they should go to a soccer match (argument α_1) while Alice thinks they should attend the ballet (argument α_2). There is time for only one activity, however (hence α_1 and α_2 defeat one another). Moreover, while Alice prefers the ballet to the soccer, she would still rather go to a soccer match than stay at home. Likewise, Brian prefers the soccer match to the ballet, but also prefers the ballet to staying home. Formally, we can write $u_A(\text{ballet}) > u_A(\text{soccer}) > u_A(\text{home})$ and $u_B(\text{soccer}) > u_B(\text{ballet}) > u_B(\text{home})$. Alice has a strong argument which she may use against going to the soccer match, namely by claiming that she is too sick to be outdoors (argument α_3). Brian simply cannot attack this argument (without compromising his marriage at least). Likewise, Brian has an irrefutable argument against the ballet; he could claim that his ex-wife will be there too (argument α_4). Alice cannot stand her! Draw the corresponding abstract argument graph, and identify the strategic (normal-form) game being played, together with the equilibria of this game.
5. **[Level 3]** Recall that Definition 5.10 provides a dialogue protocol such that the proponent wins the dialogue if and only if the argument at the root is in the grounded set. Produce variants of this protocol corresponding to the skeptical and credulous version of all semantics described in Section 3.
6. **[Level 3]** Program a web-based system that enables people to author arguments represented using the Argument Interchange Format, link to other people's arguments, and navigate argument structures.

7. **Level 3** In recent years, a number of web-based games have been developed, providing people with entertainment while, as a bi-product, producing useful content, such as agent image labels or formalized commonsense facts [54]. Design and implement a web-based game in which, given plain text snippets, human players produce annotated arguments according to an appropriate argument scheme.
8. **Level 4** Using the concepts of argumentation mechanism design, characterize conditions under which mechanisms based on various semantics are dominant-strategy incentive compatible.
9. **Level 4** Conduct an experimental study, with human participants and natural language arguments, to compare the psychological plausibility of different argumentation semantics. Explore the role of single argument structure, argument graph structure, and type of attack relation on the way humans evaluate arguments.
10. **Level 4** Investigate the design of a software agent capable of conducting successful persuasion dialogues with a human. Explore the role of game-theoretic reasoning in designing such an agent, and compare it with more heuristic behavior inspired by the psychological literature.
11. **Level 4** Using tools from automated natural language processing, program a system capable of annotating natural language arguments.

References

- [1] K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171(10-15):855–874, 2007.
- [2] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence*, 171(10–15):675–700, 2007.
- [3] A. Belesiotis, M. Rovatsos, and I. Rahwan. Agreeing on plans through iterated disputes. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 765–772. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
- [4] Trevor J. M. Bench-Capon. Argument in artificial intelligence and law. *Artificial Intelligence and Law*, 5(4):249–261, 1997.

- [5] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10–15):619–641, 2007.
- [6] P. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128(1-2):203–235, 2001.
- [7] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, Cambridge MA, USA, 2008.
- [8] A. Bondarenko, P.M. Dung, R.A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial intelligence*, 93(1-2):63–101, 1997.
- [9] M. Caminada and L. Amgoud. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(5-6):286–310, 2007.
- [10] Martin W. A. Caminada. On the issue of reinstatement in argumentation. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.
- [11] C. I. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.
- [12] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
- [13] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artificial Intelligence*, 171(10-15):701–729, 2007.
- [14] M. Elhadad. Using argumentation in text generation. *Journal of Pragmatics*, 24:189–220, 1995.
- [15] M.A. Falappa, G. Kern-Isberner, and G.R. Simari. Explanations, belief revision and defeasible reasoning. *Artificial Intelligence*, 141(1-2):1–28, 2002.
- [16] John Fox, David Glasspool, Dan Grecu, Sanjay Modgil, Matthew South, and Vivek Patkar. Argumentation-based inference and decision making – a medical perspective. *IEEE Intelligent Systems*, 22(6):34–41, 2007.
- [17] Jacob Glazer and Ariel Rubinstein. Debates and decisions: On a rationale of argumentation rules. *Games and Economic Behavior*, 36:158–173, 2001.
- [18] Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10–15):875–896, 2007.

- [19] Charles L. Hamblin. *Fallacies*. Methuen, London, UK, 1970.
- [20] Jaakko Hintikka and Gabriel Sandu. Game-theoretical semantics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 361–410. Elsevier, Amsterdam, The Netherlands, 1997.
- [21] Nishan C. Karunatilake, Nicholas R. Jennings, Iyad Rahwan, and Peter McBurney. Dialogue games that agents play within a society. *Artificial Intelligence*, 173(9-10):935–981, 2009.
- [22] Sarit Kraus, Katia Sycara, and Amir Evenchik. Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, 104(1-2):1–69, 1998.
- [23] F. Lin and Y. Shoham. A logic of knowledge and justified assumptions. *Artificial Intelligence*, 57(2-3):271–289, 1992.
- [24] Paul Lorenzen. Ein dialogisches konstruktivitätskriterium. In *Infinitistic Methods*, pages 193–200. Pergamon Press, Oxford, UK, 1961.
- [25] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York NY, USA, 1995.
- [26] Irene Mazzotta, Fiorella de Rosis, and Valeria Carofiglio. Portia: A user-adapted persuasion system in the healthy-eating domain. *IEEE Intelligent Systems*, 22(6):42–51, 2007.
- [27] P. McBurney and S. Parsons. Risk agoras: Using dialectical argumentation to debate risk. *Risk Management*, 2(2):17–27, 2000.
- [28] P. McBurney and S. Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.
- [29] Peter McBurney, Rogier M. van Eijk, Simon Parsons, and Leila Amgoud. A dialogue-game protocol for agent purchase negotiations. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(3):235–273, 2003.
- [30] S. Modgil and M. Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer, 2009.
- [31] Sanjay Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9-10):901–934, 2009.
- [32] Simon Parsons, Michael J. Wooldridge, and Leila Amgoud. Properties and complexity of formal inter-agent dialogues. *Journal of Logic and Computation*, 13(3):347–376, 2003.

- [33] Chaim Perelman and Lucie Olbrechts-Tyteca. *The New Rhetoric: A Treatise on Argumentation*. University of Notre Dame Press, 1969.
- [34] J.L. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.
- [35] H. Prakken. Models of persuasion dialogue. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 281–300. Springer, 2009.
- [36] H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124, 2010.
- [37] Henry Prakken. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation*, 15(6):1009–1040, 2005.
- [38] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.
- [39] I. Rahwan and K. Larson. Argumentation and game theory. *Argumentation in Artificial Intelligence*, pages 321–339, 2009.
- [40] I. Rahwan, M.I. Madakkatel, J.F. Bonnefon, R.N. Awan, and S. Abdallah. Behavioral experiments for assessing the abstract argumentation semantics of reinstatement. *Cognitive Science*, 34(8):1483–1502, 2010.
- [41] Iyad Rahwan and Leila Amgoud. An argumentation-based approach for practical reasoning. In Gerhard Weiss and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 347–354, 2006.
- [42] Iyad Rahwan and Kate Larson. Mechanism design for abstract argumentation. In L. Padgham, D. Parkes, J. Mueller, and S. Parsons, editors, *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1031–1038, 2008.
- [43] Iyad Rahwan, Kate Larson, and Fernando Tohmé. A characterisation of strategy-proofness for grounded argumentation semantics. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 251–256, 2009.
- [44] Iyad Rahwan and Peter McBurney. Guest editors’ introduction: Argumentation technology. *IEEE Intelligent Systems*, 22(6):21–23, 2007.
- [45] Iyad Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, Peter McBurney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *Knowledge Engineering Review*, 18(4):343–375, 2003.

- [46] Iyad Rahwan and Guillermo R. Simari, editors. *Argumentation in Artificial Intelligence*. Springer, 2009.
- [47] Iyad Rahwan, Fouad Zablith, and Chris Reed. Laying the foundations for a world wide argument web. *Artificial Intelligence*, 171(10–15):897–921, 2007.
- [48] John Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, New York, USA, 1969.
- [49] G.R. Simari and R.P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial intelligence*, 53(2-3):125–157, 1992.
- [50] Keith Stenning and Michiel van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, Cambridge MA, USA, 2008.
- [51] Stephen Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.
- [52] W. van der Hoek, M. Roberts, and M. Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis. *Synthese*, 156(1):1–19, 2007.
- [53] Frans H. van Eemeren, Rob Flanery Grootendorst, and Francisca Snoeck Henkemans, editors. *Fundamentals of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Applications*. Lawrence Erlbaum Associates, Mahwah NJ, USA, 1996.
- [54] L. von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.
- [55] J. von Neuman and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, Princeton NJ, USA, 1944.
- [56] G.A.W. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90(1-2):225–279, 1997.
- [57] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press, Albany NY, USA, 1995.
- [58] D.N. Walton, D.C. Reed, and F. Macagno. *Argumentation Schemes*. Cambridge university press, 2008.
- [59] Douglas Walton. *Fundamentals of Critical Argumentation*. Cambridge University Press, New York, USA, 2006.

Part III

Basic Coordination

Chapter 6

Computational Social Choice

Felix Brandt, Vincent Conitzer, and Ulle Endriss

1 Introduction

Social choice theory concerns the design and formal analysis of methods for aggregating the preferences of multiple agents. Examples of such methods include voting procedures, which are used to aggregate the preferences of voters over a set of candidates standing for election to determine which candidate should win the election (or, more generally, to choose an alternative from a set of alternatives), or protocols for deciding on a fair allocation of resources given the preferences of a group of stakeholders over the range of bundles they might receive. Originating in economics and political science, social choice theory has since found its place as one of the fundamental tools for the study of multiagent systems. The reasons for this development are clear: if we view a multiagent system as a “society” of autonomous software agents, each of which has different objectives, is endowed with different capabilities, and possesses different information, then we require clearly defined and well-understood mechanisms for aggregating their views so as to be able to make collective decisions in such a multiagent system.

Computational social choice, the subject of this chapter, adds an algorithmic perspective to the formal approach of social choice theory. More broadly speaking, computational social choice deals with the application of methods usually associated with computer science to problems of social choice.

1.1 Introductory Example

Let us begin with a simple example. We shall discuss it at length, in order to introduce some of the key concepts that will be treated more formally later in the chapter. Consider the following situation in which there are four Dutchmen, three Germans, and two Frenchmen who have to decide which drink will be served for lunch (only a single drink will be served to all).¹ The Dutchmen prefer milk to wine to beer, the Germans prefer beer to wine to milk, and the Frenchmen prefer wine to beer to milk. These preferences can be conveniently represented in a table where each group of agents is represented by one column.

4	3	2
milk	beer	wine
wine	wine	beer
beer	milk	milk

Now, which drink should be served based on these individual preferences? Milk could be chosen on the grounds that it has the most agents ranking it first (the Dutch). That is, it is the winner according to the *plurality rule*, which only considers how often each alternative is ranked in first place. However, a majority of agents (the Germans and the French) will be dissatisfied with this choice as they prefer *any* other drink to milk. In fact, it turns out that wine is preferred to both beer and milk by a 6:3 and a 5:4 majority of voters, respectively. An alternative with this property (defeating every other alternative in pairwise majority comparisons) is called a *Condorcet winner*. Yet another method of determining a collective choice would be to successively eliminate those beverages that are ranked first by the lowest number of agents (known as *Single Transferable Vote*, or STV). This would result in wine being eliminated first because only two agents (the French) rank it first. Between the remaining two options, beer is ranked higher by the Germans and the French, and will eventually be chosen. In summary, this example shows that collective choice is not a trivial matter, as different, seemingly reasonable, voting rules can yield very different results.

Another important lesson that can be learned from this example concerns *strategic manipulation*. Assume the collective choice is determined using the plurality rule. Since preferences are private and each agent only knows its own preferences with certainty, nobody can prevent the Germans from *claiming* that their most-preferred drink is wine. This will result in a more preferable outcome to them than reporting their preferences truthfully, because they get wine rather than milk, their least-preferred alternative. A seminal result in social choice theory, the

¹This is based on an example used by Donald G. Saari at a conference in Rotterdam, where only milk was served for lunch.

Gibbard-Satterthwaite theorem (discussed in detail in Section 3.2.1), states that *every* reasonable voting rule is susceptible to this type of manipulation.

While the example was carefully set up to avoid this, plurality and STV, as well as many other rules, can, in general, result in multiple alternatives ending up tied. If a social decision must be made, then we need to break this tie in some way – for example, by flipping a coin (resulting in a randomized rule), lexicographically according to the names of the alternatives, or by using another voting rule as a tie-breaking rule (whose own ties may yet again need to be broken). Another option is simply to “pass the buck” and declare all the tied alternatives to be winners, so that the output is now a subset of the alternatives. For obvious reasons, we will generally require this subset to be non-empty. This sounds trivial, but, for example, as we will see later in the chapter, a given election may not have a Condorcet winner at all. As a consequence, the Condorcet winner method is not even a well-defined voting rule. Many voting rules, however, are so-called *Condorcet extensions*, which means that they choose the Condorcet winner whenever one exists. This is sometimes also called the Condorcet principle. Our example above shows that neither plurality nor STV are Condorcet extensions. An example of a rule that is a Condorcet extension is *Copeland’s rule*, which chooses those alternatives that win the most pairwise majority comparisons. If no Condorcet winner exists, Copeland’s rule still yields one or more winners. A disadvantage of Copeland’s rule, in contrast to, say, the plurality rule when applied to elections with many voters and few alternatives, is that ties seem more likely here. In general, we prefer to end up with as small a set of winners as possible, but as we will see, this needs to be traded off against other properties.

We may even be a bit more ambitious and attempt to not only choose the winner(s), but rather to rank all the alternatives, representing “society’s preferences” over them. This can be useful, for example, if we are worried that some alternatives may turn out to be unavailable and we need to quickly switch to another one. We may also be interested in the aggregate ranking for other reasons; for example, consider the problem of running a single query on multiple Internet search engines, and trying to aggregate the results into a single ranking. In the example above, we can simply rank the alternatives according to their pairwise majority comparisons: wine defeats both beer and milk in their pairwise comparisons and so should be ranked first, and beer defeats milk and so should be ranked second. As we will see later, however, this approach can result in cycles. A simple approach to ranking alternatives is to use a rule that gives each alternative a score – such as plurality or Copeland – and sort the alternatives by aggregate score. (Note that if the pairwise majority approach does not result in cycles, then Copeland will agree with it.) For STV, one possibility is to sort the alternatives in inverse order of elimination. A generally applicable approach is to take the winners, rank

them first, then vote again over the remaining alternatives, and to continue in this fashion until all alternatives have been ranked.

We shall revisit several of these ideas again later on, when we define the frameworks for social choice outlined here in more formal detail.

1.2 History of the Field

There are a number of historical cases showing that the intricacies of social choice have occupied people's minds for a very long time [160]. Examples include the writings of Pliny the Younger, a senator in ancient Rome around the turn of the 1st century A.D.; the thirteenth century Catalan philosopher, alchemist, and missionary Ramon Llull; and the Marquis de Condorcet, a public intellectual who was active around the time of the French Revolution.

Social choice theory *as a scientific discipline* with sound mathematical foundations came into existence with the publication of the Ph.D. thesis of Kenneth J. Arrow in 1951 [5], who introduced the axiomatic method into the study of aggregation methods and whose seminal *Impossibility Theorem* shows that any such method that satisfies a list of seemingly basic fairness requirements must in fact amount to a dictatorial rule. Since then, much of the work in classical social choice theory has focused on results concerning the formal possibility and impossibility of aggregation methods that combine certain desirable properties – like *Pareto optimality*, *monotonicity*, or *non-manipulability* – without resulting in an unacceptable concentration of power. Some of the landmark results include Sen's characterization of preference domains allowing for consistent majority decisions [197] and the Gibbard-Satterthwaite theorem [124, 191] mentioned earlier, which establishes the impossibility of devising a reasonable, general voting rule that is immune to strategic manipulation.

The first clear examples of work in *computational* social choice are a series of papers by Bartholdi, Orlin, Tovey, and Trick, published around 1990 [17, 19, 20]. They argued that complexity theory, as studied in theoretical computer science, is relevant to social choice. For instance, they analyzed the complexity of determining the winners in an intricate voting rule due to C. L. Dodgson, better known as Lewis Carroll, the author of *Alice in Wonderland*.² They also fielded the fundamental idea that complexity barriers might provide a means of protection against strategic manipulation and other undesirable behavior. That is, while classical social choice theory showed that it is a mathematical impossibility to devise a voting rule that cannot be manipulated, computer science might provide the tools

²It was shown later that determining the winners according to Dodgson's rule is complete for the complexity class Θ_2^P [130]. This is remarkable as Θ_2^P was considered to lack “natural” complete problems and Dodgson's rule was proposed long before complexity theory existed.

for making this unwanted behavior so difficult that it can be neglected in practice.³

This groundbreaking work was followed by a small number of isolated publications throughout the 1990s. In the first few years of the twenty-first century, as the relevance of social choice to artificial intelligence, multiagent systems, and electronic commerce became apparent, the frequency of contributions on problems related to social choice with a computational flavor suddenly intensified. Although the field was still lacking a name, by 2005 contributions in what we would now call “computational social choice” had become a regular feature at several of the major conferences in artificial intelligence. The first workshop specifically dedicated to computational social choice, and the first event to explicitly use this name, took place in 2006 [102]. Around the same time, Chevaleyre et al. [56] attempted the first classification of research in the area by distinguishing (a) the nature of the social choice problem addressed, and (b) the type of formal or computational technique used.

1.3 Applications

Social choice theory was originally developed as an abstraction of problems that arise in political science and economics. More generally, social choice theory provides a useful theoretical framework for the precise mathematical study of the normative foundations of collective decision making, in a wide range of areas, involving not only human decision makers but also autonomous software agents. This chapter will focus on the theoretical foundations of computational social choice. But before we delve into the theory, let us briefly cite a few examples of actual and potential application domains, going beyond political elections and collective decision making in multiagent systems, where the methods we shall cover in this chapter can be put to good use.

The first such example comes from the domain of Internet search engines. Imagine you want to design a *metasearch engine* that combines the search results of several engines. This problem has a lot in common with preference aggregation. Aggregating preferences means asking each individual agent for a ranking over the set of alternatives and then amalgamating this information into a single such ranking that adequately represents the preferences of the group. For the metasearch engine, we ask each individual search engine for a ranking of its own, say, 20 top results and then have to aggregate this information to produce our metaranking. Of course, the problems are not exactly the same. For instance, some website may not have been ranked at all by one search engine, but be in the

³As we shall see, this approach of using computational complexity as a barrier against strategic manipulation has its limitations, but conceptually this has nevertheless been an important idea that has inspired a good deal of exciting research.

top five for another. Also, the general principles that we might want to adhere to when performing the aggregation might differ: in preference aggregation, fairness will play an important role; when aggregating search results fairness is not a goal in itself. Nevertheless, it is clear that insights from social choice theory can inform possible approaches for designing our metasearch engine. In fact, this situation is rather typical in computational social choice: for many modern applications, we can rely on some of the basic insights from social choice theory, but to actually develop an adequate solution, we do have to alter some of the classical assumptions.

There is also a less obvious application of principles of social choice to search engines. One way of measuring the *importance* of a web page is the number of other web pages linking to it. In fact, this is a recursive notion: the importance of our web page also depends on the importance of the pages linking to it, which in turn depends on the importance of the pages linking to those. This idea is the basis for the PAGERANK algorithm at the core of Google's search engine [170]. We may think of this as an election where the set of the voters and the set of the candidates coincide (both are the set of all web pages). In this sense, the ranking of the importance of web pages may be considered as a social choice problem. This perspective has led to a deeper understanding of the problem, for instance, by providing an axiomatic characterization of different ranking algorithms [3].

Another example of an application domain for which the perspective of social choice theory can provide fruitful new insights is that of *recommender systems*. A recommender system is a tool for helping users choose attractive products on the basis of choices made by other users in the past. An important technique in this field is *collaborative filtering*. By reinterpreting collaborative filtering as a process of preference aggregation, the axiomatic method developed in social choice theory has proven helpful in assessing and comparing the quality of different collaborative filtering approaches [171].

Yet another example is the problem of *ontology merging*, which arises in the context of the *Semantic Web*. Suppose different information providers on the Semantic Web provide us with different ontologies describing the same set of concepts. We would like to combine this information so as to arrive at the best possible ontology representing the available knowledge regarding the problem domain. This is a difficult problem that will require a combination of different techniques. Social choice theory can make a contribution in those cases where we have little information regarding the reliability of the individual providers and can only resort to aggregating whatever information they provide in a "fair" (and logically consistent) manner [174].

We shall allude to further areas of application along the way. However, our focus will be on theoretical foundations from here on.

1.4 Chapter Outline

In this chapter we review the foundations of social choice theory and introduce the main research topics in computational social choice that have been identified to date. Specifically, Section 2 introduces the axiomatic framework for studying preference aggregation and discusses the most important seminal result in the field, Arrow's theorem, in detail. Section 3 is an introduction to voting theory. We present the most important voting rules and then focus on the problem of strategic manipulation. This includes a discussion of the Gibbard-Satterthwaite theorem and a number of possible avenues for circumventing the impossibility it is pointing to. Section 4 focuses on voting scenarios where the set of alternatives to choose from has a combinatorial structure, as is the case when we have to elect a committee (rather than a single official) or more generally when we have to collectively decide on an instantiation of several variables. In Section 5 we turn our attention to the problem of fairly allocating a number of goods to a group of agents and discuss the problems that are characteristic of this particular type of social choice problem. Section 6 concludes with a brief discussion of related research topics in computational social choice not covered in this chapter and with a number of recommendations for further reading.

2 Preference Aggregation

One of the most elementary questions in social choice theory is how the preference relations of individual agents over some abstract set of alternatives can be aggregated into one collective preference relation. Apart from voting, this question is of broad interest in the social sciences, because it studies whether and how a society of autonomous agents can be treated as a single rational decision maker. As we point out in Section 2.1, results in this framework are very discouraging.

In many practical settings, however, one is merely interested in a set of socially acceptable alternatives rather than a collective preference relation. In Section 2.2, we discuss the relationship between both settings and present some positive results for the latter framework.

2.1 Social Welfare Functions

We start by investigating social welfare functions, the simplest and perhaps most elegant framework of preference aggregation. A social welfare function (SWF) aggregates preferences of individual agents into collective preferences. More formally, we consider a finite set $N = \{1, \dots, n\}$ of at least two agents (sometimes also called individuals or voters) and a finite universe U of at least two alterna-

tives (sometimes also called candidates). Each agent i entertains preferences over the alternatives in U , which are represented by a transitive and complete preference relation \succsim_i . *Transitivity* requires that $a \succsim_i b$ and $b \succsim_i c$ imply $a \succsim_i c$ for all $a, b, c \in U$, and *completeness* requires that any pair of alternatives $a, b \in U$ is comparable, i.e., it holds that either $a \succsim_i b$ or $b \succsim_i a$ or both. In some cases, we will assume preferences to be linear, i.e., also satisfying antisymmetry ($a \succsim_i b$ and $b \succsim_i a$ imply that $a = b$), but otherwise we impose no restrictions on preference relations. We have $a \succsim_i b$ denote that agent i likes alternative a at least as much as alternative b and write \succ_i for the strict part of \succsim_i , i.e., $a \succ_i b$ if $a \succsim_i b$ but not $b \succsim_i a$. Similarly, \sim_i denotes i 's indifference relation, i.e., $a \sim_i b$ if both $a \succsim_i b$ and $b \succsim_i a$. The set of all preference relations over the universal set of alternatives U will be denoted by $\mathcal{R}(U)$. The set of *preference profiles*, associating one preference relation with each individual agent, is then given by $\mathcal{R}(U)^n$.

Economists often also consider *cardinal* (rather than *ordinal*) preferences, which are usually given in the form of a utility function that assign numerical values to each alternative. It is easy to show that, for a finite number of alternatives, a preference relation can be represented by a utility function if and only if it satisfies transitivity and completeness (see Exercise 1). Still, a utility function may encode much more information than a preference relation, such as the intensity of preferences. In the absence of a common numeraire such as money, the meaning of individual utility values and especially the interpersonal comparisons between those values is quite controversial. Therefore, the ordinal model based on preference relations is the predominant model in abstract social choice theory. In special domains such as fair division (see Section 5), however, cardinal preferences are also used.

A social welfare function is a function that maps individual preference relations to a collective preference relation.

Definition 6.1 A social welfare function (SWF) is a function $f : \mathcal{R}(U)^n \rightarrow \mathcal{R}(U)$.

For a given preference profile $R = (\succsim_1, \dots, \succsim_n)$, the resulting *social* preference relation will be denoted by \succsim .

It was the Marquis de Condorcet who first noted that the concept of a social preference relation can be problematic. When there are just two alternatives, common sense and several axiomatic characterizations, such as May's Theorem [157], suggest that alternative a should be socially preferred to alternative b if and only if there are more voters who strictly prefer a to b than b to a . This concept is known as *majority rule*. Since the majority rule provides us with social pairwise comparisons, it appears to be a natural candidate for an SWF. However, as demonstrated by the *Condorcet paradox* [86], the majority rule can result in cycles when there

are more than two alternatives. To see this, consider the preference relations of three voters given in Figure 6.1. A majority of voters (two out of three) prefers a to b . Another majority prefers b to c and yet another one c to a . Clearly, the pairwise majority relation in this example is cyclic and therefore not a well-formed preference relation. Hence, the majority rule does not constitute an SWF.

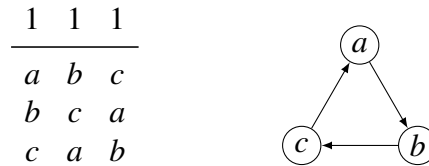


Figure 6.1: Condorcet's paradox [86]. The left-hand side shows the individual preferences of three agents such that the pairwise majority relation, depicted on the right-hand side, is cyclic.

In what is perhaps the most influential result in social choice theory, Arrow [5] has shown that this “difficulty in the concept of social welfare” (as he calls it) is not specific to the majority rule, but rather applies to a very large class of SWFs. Arrow's theorem states that a seemingly innocuous set of desiderata cannot be simultaneously met when aggregating preferences. These desiderata are *Pareto optimality*, *independence of irrelevant alternatives*, and *non-dictatorship*; they are defined as follows.

- An SWF satisfies *Pareto optimality* if strict unanimous agreement is reflected in the social preference relation. Formally, Pareto optimality requires that $a \succ_i b$ for all $i \in N$ implies that $a \succ b$.
- An SWF satisfies *independence of irrelevant alternatives (IIA)* if the social preference between any pair of alternatives only depends on the individual preferences restricted to these two alternatives. Formally, let R and R' be two preference profiles and a and b be two alternatives such that $R|_{\{a,b\}} = R'|_{\{a,b\}}$, i.e., the pairwise comparisons between a and b are identical in both profiles. Then, IIA requires that a and b are also ranked identically in \succsim , i.e., $\succsim|_{\{a,b\}} = \succsim'|_{\{a,b\}}$.
- An SWF is *non-dictatorial* if there is no agent who can dictate a strict ranking no matter which preferences the other agents have. Formally, an SWF is non-dictatorial if there is no agent i such that for all preference profiles R and alternatives a, b , $a \succ_i b$ implies that $a \succ b$.

Theorem 6.1 (Arrow, 1951) *There exists no SWF that simultaneously satisfies IIA, Pareto optimality, and non-dictatorship whenever $|U| \geq 3$.*

According to Paul Samuelson, who is often considered the founding father of modern economics, Arrow's theorem is one of the significant intellectual achievements of the twentieth century [188]. A positive aspect of such a negative result is that it provides boundaries on what can actually be achieved when aggregating preferences. In particular, Arrow's theorem shows that at least one of the required conditions has to be omitted or relaxed in order to obtain a positive result. For instance, if $|U| = 2$, IIA is trivially satisfied by *any* SWF and reasonable SWFs (such as the majority rule) also satisfy the remaining conditions. In a much more elaborate attempt to circumvent Arrow's theorem, Young [231] proposed to replace IIA with *local IIA* (LIIA), which only requires IIA to hold for consecutive pairs of alternatives in the social ranking. By throwing in a couple of other conditions (such as anonymity and neutrality, which will be defined in Section 3) and restricting attention to linear individual preferences, Young completely characterizes an aggregation function known as *Kemeny's rule*.

Kemeny's rule. Kemeny's rule [140] yields all strict rankings that agree with as many pairwise preferences of the agents as possible. That is, it returns

$$\arg \max_{\succ} \sum_{i \in N} |\succ \cap \succ_i|.$$

Since there can be more than one ranking that satisfies this property, Kemeny's rule is not really an SWF but rather a multi-valued SWF. (Young refers to these as *social preference functions*.) Alternatively, Kemeny's rule can be characterized using maximum likelihood estimation [231, 232].⁴ Over the years, Kemeny's rule has been reinvented by many scholars in different fields. It is also known as the *median* or *linear ordering* procedure [15, 53]. Kemeny's rule is not only very interesting from an axiomatic but also from a computational point of view. The problem of computing a Kemeny ranking, as well as the closely related problem of computing a *Slater* ranking (a ranking that agrees with the outcomes of as many pairwise elections as possible), correspond to a computational problem on graphs known as the *minimum feedback arc set problem* (in the case of Kemeny's rule, the weighted version of this problem). It has been shown that computing a Kemeny

⁴This is done under a model where there exists a "correct" ranking of the alternatives, and the agents' preferences are noisy estimates of this correct ranking. This result relies on a particular noise model; if the noise model is changed, the maximum likelihood solution can result in other SWFs, though for yet other SWFs, it can be proved that no noise model would yield that SWF as the solution [70, 76, 90, 209]. However, the Kemeny result is robust to some other generalizations of the model [66, 222].

ranking is NP-hard [20], even when there are just four voters [97]. Moreover, deciding whether a given alternative is ranked first in a Kemeny ranking is Θ_2^P -complete [131]. Nevertheless, under certain conditions, there is a polynomial-time approximation scheme (PTAS) for the Kemeny problem [141]. For further details on these problems, we refer to the works of Alon [2], Betzler et al. [23, 26], Brandt et al. [48], Charon and Hudry [53], Conitzer [61], Conitzer et al. [73], Davenport and Kalagnanam [84], Hudry [135, 136], and Ali and Meila [1].⁵

Rather than relaxing the explicit conditions in Arrow's theorem, one may call its implicit assumptions into question. For instance, in many applications, a full social preference relation is not needed; rather, we just wish to identify the socially most desirable alternatives. This corresponds to the framework considered in the following section.⁶

2.2 Social Choice Functions

The central objects of study in this section are *social choice functions*, i.e., functions that map the individual preferences of the agents and a feasible subset of the alternatives to a set of socially preferred alternatives, the choice set. Throughout this chapter, the set of possible feasible sets $\mathcal{F}(U)$ is defined as the set of all non-empty subsets of U . A feasible set (or agenda) defines the set of possible alternatives in a specific choice situation at hand. The reason for allowing the feasible set to vary is that we will later define properties that relate choices from different feasible sets to each other [see also 200, 206].

Definition 6.2 A social choice function (SCF) is a function $f : \mathcal{R}(U)^n \times \mathcal{F}(U) \rightarrow \mathcal{F}(U)$ such that $f(R, A) \subseteq A$ for all R and A .

2.2.1 The Weak Axiom of Revealed Preference

Arrow's theorem can be reformulated for SCFs by appropriately redefining Pareto optimality, IIA, and non-dictatorship and introducing a new property called the weak axiom of revealed preference, as follows.

⁵In 1995, Peyton Young predicted “that the time will come when [Kemeny's rule] is considered a standard tool for political and group decision making” [232]. This has not yet happened, but the website www.vote-fair.org provides an interface to use Kemeny's rule for surveys, polls, and elections at no charge.

⁶This effectively reduces the codomain of the aggregation function. As we will see in Section 3.2.2, a common technique to avoid negative results in social choice theory is to reduce the *domain* of the function.

Pareto optimality now requires that an alternative should not be chosen if there exists another feasible alternative that *all* agents unanimously prefer to the former – more precisely, $a \notin f(R, A)$ if there exists some $b \in A$ such that $b \succ_i a$ for all $i \in N$. An SCF f is non-dictatorial if there is no agent i such that for all preference profiles R and alternatives a , $a \succ_i b$ for all $b \in A \setminus \{a\}$ implies $a \in f(R, A)$.⁷ Independence of irrelevant alternatives reflects the idea that choices from a set of feasible alternatives should not depend on preferences over alternatives that are infeasible, i.e., $f(R, A) = f(R', A)$ if $R|_A = R'|_A$. Interestingly, in the context of SCFs, IIA constitutes no more than a framework requirement for social choice and is not the critical assumption it used to be in the context of SWFs.

Finally, the *weak axiom of revealed preference* (WARP) demands that choice sets from feasible sets are strongly related to choice sets from feasible subsets. Let A and B be feasible sets such that $B \subseteq A$. WARP requires that the choice set of B consists precisely of those alternatives in B that are also chosen in A , whenever this set is non-empty. Formally, for all feasible sets A and B and preference profiles R ,

$$\text{if } B \subseteq A \text{ and } f(R, A) \cap B \neq \emptyset \text{ then } f(R, A) \cap B = f(R, B). \quad (\text{WARP})$$

We are now ready to state a variant of Arrow's theorem for SCFs.

Theorem 6.2 (Arrow, 1951, 1959) *There exists no SCF that simultaneously satisfies IIA, Pareto optimality, non-dictatorship, and WARP whenever $|U| \geq 3$.*

As the Arrovian conditions – Pareto optimality, IIA, non-dictatorship, and WARP – cannot be satisfied by any SCF, at least one of them needs to be excluded or relaxed to obtain positive results. Clearly, dropping non-dictatorship is unacceptable and, as already mentioned, IIA merely states that the SCF represents a reasonable model of preference aggregation [see, e.g., 29, 193]. Wilson [215] has shown that without Pareto optimality only SCFs that are constant (i.e., completely unresponsive) or fully determined by the preferences of a single agent are possible. Moreover, it could be argued that not requiring Pareto optimality runs counter to the very idea of *social* choice. Accordingly, the only remaining possibility is to relax WARP.

2.2.2 Contraction and Expansion Consistency

Building on earlier work by Sen [198], Bordes [28] factorized WARP into two separate conditions by splitting the equality in the consequence of the definition of WARP into two inclusions. The resulting conditions are known as *contraction* and *expansion*.

⁷Theorem 6.2 holds for an even weaker notion of non-dictatorship in which a dictator can enforce that $\{a\} = f(R, A)$.

Contraction prescribes that an alternative that is chosen from some feasible set will also be chosen from all subsets in which it is contained. Formally, SCF f satisfies contraction if for all A, B and R ,

$$\text{if } B \subseteq A \text{ then } B \cap f(R, A) \subseteq f(R, B). \quad (\text{contraction})$$

The intuition behind *expansion* is that if alternative a is chosen from some set that contains another alternative b , then it will also be chosen in all supersets in which b is chosen. Formally, SCF f satisfies expansion if for all A, B and R ,

$$\text{if } B \subseteq A \text{ and } B \cap f(R, A) \neq \emptyset \text{ then } f(R, B) \subseteq B \cap f(R, A). \quad (\text{expansion})$$

One possibility to escape the haunting impossibility of social choice is to require only contraction or expansion but not both at the same time. It turns out that contraction and even substantially weakened versions of it give rise to impossibility results that retain Arrow's spirit [199]. As an example, consider the preference profile given in Figure 6.1. All of the voting rules mentioned in the introduction (plurality, STV, and Copeland) will yield a tie between all three alternatives. Hence, if any of these rules were to satisfy contraction, they would need to yield both available alternatives in every two-element subset of $\{a, b, c\}$. However, this is not the case for *any* of these subsets as each of them has a single winner according to all three rules (in fact, it is a 2:1 majority in each case, so this would be the case for almost any natural rule).

Expansion consistency conditions, on the other hand, are much less restrictive. In fact, a number of appealing SCFs can be characterized using weakenings of expansion and inclusion minimality. Inclusion minimality is quite natural in this context as one is typically interested in choice sets that are as small as possible.⁸ These characterizations are nice examples of how positive results can be obtained by using the axiomatic method.

In the following, an SCF is said to be *majoritarian* if choice sets only depend on the majority rule within the feasible set. For technical reasons, we furthermore assume that n is odd and that the preferences of the voters are strict, which guarantees that the majority rule is asymmetric.

Top cycle. The top cycle is the smallest majoritarian SCF satisfying expansion [28]. It consists of the maximal elements of the transitive closure of the weak majority relation [45, 88] and can be computed in linear time by using standard algorithms for identifying strongly connected components in digraphs such as those due to Kosaraju or Tarjan [see, e.g., 80].

⁸Moreover, due to the inclusive character of expansion consistency conditions, they are easily satisfied by very undiscriminatory SCFs. For instance, the trivial SCF, which always yields all feasible alternatives, trivially satisfies expansion (and all of its weakenings).

Uncovered set. The uncovered set is the smallest majoritarian SCF satisfying a weak version of expansion [164]. Interestingly, the uncovered set consists precisely of those alternatives that reach every other alternative on a majority rule path of length at most two [202]. Based on this characterization, computing the uncovered set can be reduced to matrix multiplication and is thus feasible in almost linear time [42, 134].

Banks set. The Banks set is the smallest majoritarian SCF satisfying a weakening of weak expansion, called strong retentiveness [40]. In contrast to the previous two SCFs, the Banks set cannot be computed in polynomial time unless P equals NP. Deciding whether an alternative is contained in the Banks set is NP-complete [47, 216]. Interestingly, some alternatives (and thus subsets) of the Banks set can be found in linear time [133]. A very optimized (exponential-time) algorithm for computing the Banks set was recently proposed by Gaspers and Mnich [122].⁹

Two other SCFs, namely the *minimal covering set* [95] and the *bipartisan set* [147], have been axiomatized using a variant of contraction, which is implied by WARP [43]. While the bipartisan set can be computed using a single linear program, the minimal covering set requires a slightly more sophisticated, yet polynomial-time algorithm [42]. In addition to efficient computability, the minimal covering set and the bipartisan set satisfy a number of other desirable properties [39, 151] (see also Section 3.2.5).

3 Voting

In the previous section, we started our formal treatment of social choice and encountered some of the fundamental limitations that we face. The purpose of presenting these limitations at the outset is of course not to convince the reader that social choice is hopeless and we should give up on it; it is too important for that. (One is reminded of Churchill's quote that "democracy is the worst form of government except for all those other forms that have been tried from time to time.") Rather, it is intended to get the reader to think about social choice in a precise manner and to have realistic expectations for what follows. Now, we can move on to some more concrete procedures for making decisions based on the preferences of multiple agents.

⁹Another SCF, the tournament equilibrium set [194], was, for more than 20 years, conjectured to be the unique smallest majoritarian SCF satisfying retentiveness, a weakening of strong retentiveness. This was recently disproven by Brandt et al. [49]. Deciding whether an alternative is contained in the tournament equilibrium set of a tournament is NP-hard [47]. This problem is not known to be in NP and may be significantly harder.

3.1 Voting Rules

We begin by defining voting rules.

Definition 6.3 A voting rule is a function $f : \mathcal{R}(U)^n \rightarrow \mathcal{F}(U)$.

Of course, every SCF can also be seen as a voting rule. There are two reasons we distinguish SCFs from voting rules. First, from a technical perspective, the SCFs defined in the previous section were axiomatized using variable feasible sets in order to salvage some degree of collective rationality. Second, some of these SCFs (e.g., the top cycle) can hardly be considered voting rules because they are not discriminatory enough. Of course, the latter is merely a gradual distinction, but there have been attempts to formalize this [see, e.g., 10, 117, 195, 207]. When ignoring all conditions that relate choices from different feasible sets with each other, we have much more freedom in defining aggregation functions. For simplicity, we assume throughout this section that preferences are linear, i.e., there are no ties in individual preference relations.

An important property that is often required of voting rules in practice, called *resoluteness*, is that they should always yield a unique winner. Formally, a voting rule f is *resolute* if $|f(R)| = 1$ for all preference profiles R . Two natural symmetry conditions are anonymity and neutrality. *Anonymity* requires that the outcome of a voting rule is unaffected when agents are renamed (or more formally, when the individual relations within a preference profile are permuted). In a similar vein, *neutrality* requires that a voting rule is invariant under renaming alternatives.

Unfortunately, in general, anonymous and neutral voting rules cannot be single-valued. The simplest example concerns two agents and two alternatives, each of which is preferred by one of the voters. Clearly, a single alternative can only be chosen by breaking anonymity or neutrality.¹⁰

In the remainder of this section, we will define some of the most common voting rules.

3.1.1 Scoring Rules

A common objection to the plurality rule is that an alternative ought to get some credit for being ranked, say, in second place by a voter. Under a (positional) scoring rule, each time an alternative is ranked i th by some voter, it gets a particular score s_i . The scores of each alternative are then added and the alternatives with the highest cumulative score are selected. Formally, for a fixed number of alternatives m , we define a *score vector* as a vector $s = (s_1, \dots, s_m)$ in \mathbb{R}^m such

¹⁰Moulin [163] has shown that anonymous, neutral, and resolute voting rules exist if and only if $|U|$ can be written as the sum of non-trivial divisors of n .

that $s_1 \geq \dots \geq s_m$ and $s_1 > s_m$. Three well-known examples of scoring rules are *Borda's rule*, the *plurality rule*, and the *anti-plurality rule*.

Borda's rule. Under *Borda's rule* alternative a gets k points from voter i if i prefers a to k other alternatives, i.e., the score vector is $(|U| - 1, |U| - 2, \dots, 0)$. Borda's rule takes a special place within the class of scoring rules as it chooses those alternatives with the *highest average rank* in individual rankings. While Borda's rule is not a Condorcet extension, it is the *only* scoring rule that never gives a Condorcet winner the lowest accumulated score [204]. Another appealing axiomatic characterization of Borda's rule was given by Young [228].

Plurality rule. The score vector for the plurality rule is $(1, 0, \dots, 0)$. Hence, the cumulative score of an alternative equals the number of voters by which it is ranked first.

Anti-plurality rule. The score vector for the anti-plurality rule (which is sometimes also called *veto*) is $(1, \dots, 1, 0)$. As a consequence, it chooses those alternatives that are least-preferred by the lowest number of voters.

Due to their simplicity, scoring rules are among the most used voting rules in the real world. Moreover, there are various elegant characterizations of scoring rules. In Section 2, we introduced axioms that impose consistency restrictions on choice sets when the set of feasible alternatives varies. Alternatively, one can focus on changes in the set of voters. A very natural consistency property with respect to a variable electorate, often referred to as *reinforcement*, was suggested independently by Smith [204] and Young [228]. It states that all alternatives that are chosen simultaneously by two disjoint sets of voters (assuming that there is at least one alternative with this property) should be precisely the alternatives chosen by the union of both sets of voters. When also requiring anonymity, neutrality, and a mild technical condition, Smith [204] and Young [229] have shown that scoring rules are the *only* voting rules satisfying these properties simultaneously.

A voting procedure, popularized by Brams and Fishburn [36], that is closely related to scoring rules is *approval voting*. In approval voting, every voter can approve any number of alternatives and the alternatives with the highest number of approvals win. We deliberately called approval voting a voting procedure, because technically it is not really a voting rule (unless we impose severe restrictions on the domain of preferences by making them dichotomous). Various aspects of approval voting (including computational ones) are analyzed in a recent compendium by Laslier and Sanver [152].

3.1.2 Condorcet Extensions

As mentioned in Section 1, a Condorcet winner is an alternative that beats every other alternative in pairwise majority comparisons. We have already seen in the Condorcet paradox that there are preference profiles that do not admit a Condorcet winner. However, whenever a Condorcet winner does exist, it obviously has to be unique. Many social choice theorists consider the existence of Condorcet winners to be of great significance and therefore call any voting rule that picks a Condorcet winner whenever it exists a *Condorcet extension*. For aficionados of Condorcet's criterion, scoring rules present a major disappointment: every scoring rule fails to select the Condorcet winner for some preference profile [118]. This is shown by using one universal example given in Figure 6.2.

6	3	4	4	
<hr/>				
<i>a</i>	<i>c</i>	<i>b</i>	<i>b</i>	<i>a</i> 's score : $6 + 7s_2$
<i>b</i>	<i>a</i>	<i>a</i>	<i>c</i>	<i>b</i> 's score : $8 + 6s_2$
<i>c</i>	<i>b</i>	<i>c</i>	<i>a</i>	<i>c</i> 's score : $3 + 4s_2$

Figure 6.2: Example due to Fishburn [118], which shows that no scoring rule is a Condorcet extension. Scores for the score vector $(1, s_2, 0)$ are given on the right-hand side.

It is easily verified that alternative *a* is a Condorcet winner as 9 out of 17 voters prefer *a* to *b* and 10 out of 17 voters prefer *a* to *c*. Now, consider an arbitrary scoring rule with score vector (s_1, s_2, s_3) . Due to the linearity of scores, we may assume without loss of generality that $s_1 = 1$ and that $s_3 = 0$. The resulting scores for each alternative are given in Figure 6.2. Since $s_2 \in [0, 1]$, the score of alternative *b* always exceeds that of alternatives *a* and *c*. In other words, *b* is the unique winner in any scoring rule, even though *a* is the Condorcet winner.

We now give some examples of rules that do satisfy Condorcet's criterion. This list is far from complete, but it already shows the wide variety of Condorcet extensions.

Copeland's rule. We have already mentioned *Copeland's rule*: an alternative gets a point for every pairwise majority win, and some fixed number of points between 0 and 1 (say, $1/2$) for every pairwise tie. The winners are the alternatives with the greatest number of points.

Maximin. Under the *maximin* rule, we consider the magnitude of pairwise election results (by how many voters one alternative was preferred to the other). We evaluate every alternative by its worst pairwise defeat by another alternative; the winners are those who lose by the lowest margin in their worst pairwise defeats. (If there are any alternatives that have no pairwise defeats, then they win.)

Dodgson's rule. *Dodgson's rule* yields all alternatives that can be made a Condorcet winner by interchanging as few adjacent alternatives in the individual rankings as possible. Deciding whether an alternative is a Dodgson winner is Θ_2^P -complete and thus computationally intractable [20, 130]. Various computational properties of Dodgson's rule such as approximability and fixed-parameter tractability have been studied [see, e.g., 24, 51, 52, 158]. Unfortunately, Dodgson's rule violates various mild axioms that almost all other Condorcet extensions satisfy [see, e.g., 38].

Young's rule. *Young's rule* is based on removing voters in order to obtain a Condorcet winner. More precisely, it yields all alternatives that can be made a Condorcet winner by removing as few voters as possible. Deciding whether an alternative is a winner according to Young's rule is Θ_2^P -complete [185].¹¹ Further computational results for Young's rule were obtained by Betzler et al. [24], Caragiannis et al. [51]

Nanson's rule. *Nanson's rule* is a runoff method similar to STV as described in Section 1.1. In Nanson's original definition, a series of Borda elections is held and all alternatives who at any stage have no more than the average Borda score are excluded unless all alternatives have identical Borda scores, in which case these candidates are declared the winners. There exist two variants of Nanson's rule due to Fishburn and Schwartz, which exclude candidates with the *lowest* Borda score (also known as *Baldwin's rule*) and candidates whose Borda score is *less than* the average score, respectively [167].

Ranked pairs. The *ranked pairs* rule generates a ranking of all alternatives (and the first-ranked alternative can be considered the winner). It first sorts all pairwise elections by the magnitude of the margin of victory. Then, starting with the pairwise election with the largest margin, it “locks in” these results in this order, so that the winner of the current pairwise election must be ranked above the loser in the final ranking – unless this would create a cycle due to previously locked-in

¹¹Young [230] actually defined his rule using *weak* Condorcet winners (see Exercise 15). Brandt et al. [46] have shown that the hardness result by Rothe et al. [185] carries over to Young's original definition.

results, in which case we move on to the next pairwise election. A similar voting rule was proposed by Schulze [192].

All SCFs mentioned in Section 2.2 (e.g., the top cycle, the uncovered set, and the Banks set) also happen to be Condorcet extensions. This is because the Condorcet criterion can be seen as a very weak variant of expansion consistency: whenever an alternative is chosen in all two-element subsets, then it should also be chosen from the union of all these sets. Many of the proposed Condorcet extensions can be seen as refinements of these SCFs because they always yield elements of, say, the top cycle or the uncovered set. Other prominent Condorcet extensions are Kemeny's rule and Slater's rule (see Section 2.1).

3.1.3 Other Rules

While scoring rules and Condorcet extensions are two important classes of voting rules, many other rules that do not fit in either class have been proposed over the years. Two examples are STV and Bucklin's rule.

STV. We have already mentioned the *STV rule*: it looks for the alternatives that are ranked in first place the least often, removes them from all voters' ballots (so that some of them may now rank a different alternative first), and repeats. The alternatives removed in the last round (which results in no alternatives being left at all) win.

Bucklin's rule. In the (simple version of) *Bucklin's rule*, we first check whether there is any alternative that is ranked first by more than half the voters; if so, this alternative wins. If not, we check whether there are any alternatives that are ranked in either first or second place by more than half the voters; if so, they win. If not, we consider the first three positions, etc. When multiple alternatives cross the $n/2$ threshold simultaneously, it is common to break ties by the margin by which they crossed the threshold.

In order to gain more insight into the huge zoo of voting rules, various axioms that may or may not be satisfied by a voting rule have been put forward. Sometimes a certain set of axioms completely characterizes a single voting rule (such as the SCFs proposed in Section 2.2.2) or an interesting class of voting rules (such as the class of scoring rules in Section 3.1.1). Another stream of research studies the rationalization of voting rules by measuring the *distance* (according to various metrics) of a given preference profile to the nearest preference profile that satisfies certain consensus properties (e.g., being completely unanimous or admitting a Condorcet winner). This approach goes back to Dodgson's voting rule

mentioned in Section 3.1.2 and covers many of the rules proposed in this section [99, 100, 161].

3.2 Manipulation

So far, we have assumed that the preferences of all voters are known. In reality, generally the voters need to *report* their preferences. A significant problem is that a voter may be incentivized to report preferences other than its true ones. For example, consider a plurality election between three alternatives, a , b , and c . Consider voter i with preferences $a \succ_i b \succ_i c$. Moreover, suppose that voter i believes that almost nobody else will rank a first, but it will be a close race between b and c . Then, i may be best off casting a vote in which b is ranked first: it has little hope of getting a to win, so it may be better off focusing on ensuring that at least b will win.

One may wonder why manipulation is something to be avoided. First, the possibility of manipulation leads to fairness issues since manipulative skills are usually not spread evenly across the population. Second, energy and resources are wasted on determining how best to manipulate. Third, it makes it difficult to evaluate whether the resulting outcome is in fact one that makes sense with respect to the true preferences (as opposed to the reported ones). As we will see, the question of *how* to manipulate is not only computationally but also conceptually problematic. It raises various fundamental game-theoretic questions and makes it very difficult to make predictions or theoretical statements about election outcomes.¹² There is also a result in the theory of mechanism design known as the *revelation principle*, which can be very informally described as saying that anything that can be achieved by a mechanism in which agents play strategically, can also be achieved by a mechanism in which agents are best off telling the truth, underlining again the importance of truthful voting (for more details see the chapter on “Mechanism Design and Auctions” in this book). Unfortunately, as we shall see, the problem of manipulation cannot be avoided in general as *every* single-valued

¹²One reason for this is that voting games can have many different *equilibria*. For example, in a plurality election, it can be an equilibrium for all voters to vote for either b or c , even though *all* voters rank a first in their true preferences! This is so because if nobody else is expected to vote for a , then it does not make sense to waste one’s vote on a . If such an equilibrium seems artificial, imagine a society in which two parties dominate the political scene and put forward candidates b and c , whereas a is a third-party candidate. Of course, there are other equilibria as well, which will in general result in different winners. This makes it difficult to make any predictions about strategic voting. One context in which we can make a sharp game-theoretic prediction of the winner is the one in which the agents vote in sequence, one after the other, observing what the earlier agents have voted (see also Section 4.2). Unfortunately, in this context, paradoxical results can be exhibited where the game-theoretic outcome does not reflect the voters’ true preferences well. For more detail, see the work of Desmedt and Elkind [89] and Xia and Conitzer [220].

voting rule for more than two alternatives is susceptible to manipulation.

3.2.1 The Gibbard-Satterthwaite Impossibility

In order to formally capture whether voting rule f can be manipulated by voter i , we initially make the following assumptions. First, since voters just have preferences over single alternatives we assume that f is resolute (i.e., single-valued). And second, we assume that i knows the preferences of all other voters. This latter assumption is not entirely unreasonable in some settings (e.g., decision making in committees) and actually makes all statements about non-manipulability particularly strong because it guarantees non-manipulability even when all preferences are known.

Formally, a resolute voting rule f is *manipulable* by voter i if there exist preference profiles R and R' such that $R_j = R'_j$ for all $j \neq i$ and $f(R') \succ_i f(R)$. (Recall that \succ_i corresponds to the strict part of \succsim_i (not \succsim'_i .) A voting rule is *strategyproof* if it is not manipulable.

Just like an SCF, a voting rule f is *non-dictatorial* if there is no voter i such that for all preference profiles R , $a \in f(R)$ where a is voter i 's most preferred alternative.¹³ Finally, we need a technical condition, even weaker than Pareto optimality, that ensures that at least three different alternatives can be returned by the voting rule, as follows. A voting rule is *non-imposing* if its image contains all singletons of $\mathcal{F}(U)$, i.e., every single alternative is returned for some preference profile.

Theorem 6.3 (Gibbard, 1973; Satterthwaite, 1975) *Every non-imposing, strategyproof, resolute voting rule is dictatorial when $|U| \geq 3$.*

Just as for Arrow's theorem, we will now consider different ways to circumvent this impossibility by calling some of its explicit and implicit assumptions into question.

3.2.2 Restricted Domains of Preferences

One of the implicit assumptions of the Gibbard-Satterthwaite theorem is that the voting rule needs to be defined for *all* possible preference profiles. An important stream of research has consequently studied which restricted domains of preferences allow for strategyproof voting rules.

An important observation by Moulin [165] is that in every restricted domain that always admits a Condorcet winner, the SCF that uniquely chooses the Condorcet winner is strategyproof (see Exercise 9). There are many examples of such

¹³For *resolute* voting rules, $a \in f(R)$ obviously implies $\{a\} = f(R)$.

domains. The best-known among these is the domain of *single-peaked* preferences. Suppose there is a linear ordering $<$ on the alternatives, signifying which alternatives are “smaller” than others. For example, the voters may be voting over what the tax rate should be. In this case, the set of alternatives may be $\{20\%, 30\%, 40\%, 50\%, 60\%\}$, and clearly, $20\% < 30\% < \dots < 60\%$. They may also be voting over possible dates for a deadline, in which case earlier dates could be considered smaller; they may be voting over a location along a road at which to build a building, in which case locations further to the west could be considered “smaller”; or, more abstractly, they could be voting over political candidates, in which case candidates further to the left of the political spectrum could be considered “smaller.”

Imposing an ordering $<$ on the alternatives, in and of itself, of course does not restrict the preferences yet; we must say something about how the preferences *relate* to the order over the alternatives. Preferences are said to be single-peaked with respect to the order $<$ if the following holds: for every voter, as we move away (according to $<$) from the voter’s most-preferred alternative, the alternatives will become less preferred for that voter. Formally, a preference profile R is single-peaked if for every $x, y, z \in U$, it holds that

if $(x < y < z)$ or $(z < y < x)$, then $x \succ_i y$ implies $y \succ_i z$ for every $i \in N$.

When preferences are single-peaked and there is an odd number of voters, there is always a unique Condorcet winner. If we sort voters by $<$ according to their most-preferred alternative (breaking ties arbitrarily), then the $((n+1)/2)$ th voter is called the *median voter*. Its top choice is always identical to the Condorcet winner, as was first observed by Black [27]. (The reason that the median voter’s most-preferred alternative c is always the Condorcet winner is simple. Consider any other alternative c' ; without loss of generality, suppose $c' < c$. Then, by single-peakedness, all the voters whose most-preferred alternative is equal to or greater than c will prefer c to c' , and these constitute more than half the voters.) Hence, to determine a Condorcet winner, it suffices to know every voter’s top choice, even though the voters’ preference relations contain more information than just their most-preferred alternatives.

Now, what is the relation of all of this to the Gibbard-Satterthwaite theorem? The answer is that the median-voter rule, in spite of clearly allowing every alternative the possibility of winning and not being a dictatorial rule, is strategyproof when we restrict attention to preference profiles that are single-peaked. This follows immediately from the more general result by Moulin [165], which we stated earlier, which says that any Condorcet extension is strategyproof when we only consider profiles with a Condorcet winner. Still, it is instructive to explain the reasons for strategyproofness directly. To do so, consider what a voter who did not get its most preferred alternative elected could do. Suppose the winner a is “smaller”

than its own top choice b . If it manipulates and instead of truthfully declaring b as its top choice decides to report a “smaller” alternative b' , then either the winner will not change or the winner will become even “smaller” and thus even less attractive. On the other hand, if it reports a “larger” alternative b'' instead, then it will not affect the median (and thus the winner) at all. Hence, any form of manipulation will either damage its interests or have no effect at all. The exact same argument would continue to hold even if instead of choosing the median, or 50th-percentile, voter, we chose the (say) 60th-percentile voter, even though this clearly would not necessarily choose the Condorcet winner. The argument is also easily modified to prove the stronger property of *group-strategyproofness* (where a group of agents can join forces in attempting to manipulate the outcome).

Single-peakedness has also been studied from a computational point of view. It is very easy to check whether a preference profile is single-peaked according to a specific given ordering $<$. However, it is less obvious whether it can be checked efficiently whether a preference profile is single-peaked according to *some* ordering $<$. Building on previous work by Bartholdi, III and Trick [18], Escoffier et al. [106] proposed a linear-time algorithm for this problem. In other work, Conitzer [63] and Farfel and Conitzer [116] investigated how to elicit the voters’ preferences by asking as few queries as possible when preferences are known to be single-peaked (with the latter paper focusing on settings where agents have most-preferred *ranges* of alternatives). The computational hardness of manipulation (which will be introduced in the next section) for voting rules other than median voting has also been examined in the context of single-peaked preferences [46, 111, 213].

Another important domain of restricted preferences is that of *value-restricted* preferences, which also guarantees the existence of a Condorcet winner and subsumes many other domains such as that of single-peaked preferences [197, 201].

3.2.3 Computational Hardness of Manipulation

The positive results for restricted preferences discussed above are encouraging for settings where we can expect these restrictions to hold. Unfortunately, in many settings we would *not* expect them to hold. For example, while placing political candidates on a left-to-right spectrum may give us some insight into what the voters’ preferences are likely to be, we would still expect many of their preferences to be not exactly single-peaked: a voter may rank a candidate higher because the candidate is especially charismatic, or perhaps voters are somewhat more sophisticated and they really consider two spectra, a social one and an economic one. This is perhaps the main downside of the approach of restricting the voters’ preferences: we generally have no control over whether the preferences actually satisfy the restriction, and if they do not, then there is little that we can do.

We now discuss another approach to circumventing impossibility results such as the Gibbard-Satterthwaite theorem. Here, the idea is that the mere theoretical *possibility* of manipulation need not be a problem if in practice, opportunities for manipulation are computationally too hard to *find*. So, how hard is it to find effective manipulations? For this, we first need to clearly define the manipulation problem as a computational problem. The best-known variant is the following, which takes the perspective of a single voter who (somehow) already knows all the other votes, and wishes to determine whether it can make a particular alternative the winner.

Definition 6.4 *In the manipulation problem for a given resolute voting rule, we are given a set of alternatives, a set of (unweighted) votes, and a preferred alternative p . We are asked whether there exists a single vote that can be added so that p wins.*¹⁴

One may object to various aspects of this definition. First of all, one may argue that what the manipulator seeks to do is not to make a given alternative the winner, but rather to get an alternative elected that is as high in its true ranking as possible. This does not pose a problem: if the manipulator can solve the above problem, it can simply check, for every alternative, whether it can make that alternative win, and subsequently pick the best of those that can win. (Conversely, to get an alternative elected that is as high in its true ranking as possible, the manipulator needs to check first of all whether it can make its most-preferred alternative win, which comes down to the above problem.) Another objection is that the manipulator generally does not know the votes of all the other voters. This is a reasonable objection, though it should be noted that as long as it is *possible* that the manipulator knows the votes of all the other voters, the above problem remains a special case (and thus, any (say) NP-hardness results obtained for the above definition still apply).

Inspired by early work by Bartholdi, III et al. [19], recent research in computer science has investigated how to use computational hardness – primarily NP-hardness – as a barrier against manipulation [see, e.g., 68, 74, 98, 110, 129]. Finding a beneficial manipulation is known to be NP-hard for several rules, including second-order Copeland [19], STV [17], ranked pairs [224], and Nanson and Baldwin's rules [166]. Many variants of the manipulation problem have also been considered. In the *coalitional* manipulation problem, the manipulators can cast multiple votes in their joint effort to make p win. Because the single-manipulator

¹⁴Often, the problem is defined for irresolute voting rules; in this case, the question is either whether p can be made one of the winners, or whether p can be made the unique winner. These questions can be interpreted to correspond to the cases where ties are broken in favor of p , and where they are broken against p , respectively.

case is a special case of this problem where the coalition happens to have size 1, this problem is NP-hard for all the rules mentioned above. However, other rules are also NP-hard to manipulate in this sense, including Copeland [109, 115], maximin [224], and Borda [25, 85]. Finally, in the *weighted* version of this problem, weights are associated with the voters, including the manipulators (a vote of weight k counts as k unweighted votes). Here, many rules become NP-hard to manipulate even when the number of alternatives is fixed to a small constant [74, 129, 166].

In the *destructive* version of the problem, the goal is not to make a given alternative a win, but rather to make a given alternative a *not* win [74]. For contrast, the regular version is called the *constructive* version. If the constructive version is easy, then so is the destructive version, because to solve the destructive version it suffices to solve the constructive version for every alternative other than a ; but in some cases, the destructive version is easy while the constructive version is not.

Computational hardness has also been considered as a way of avoiding other undesirable behavior. This includes *control* problems, where the chair of the election has (partial) control over some aspects of the election (such as which alternatives are in the running or which voters get to participate) and tries to use this to get a particular alternative to win [16, 75, 113, 132, 179]. Another example is the *bribery* problem, where some interested party attempts to bribe the voters to bring about a particular outcome [108, 112, 113].

One downside of using NP-hardness to prevent undesirable behavior – whether it be manipulation, control, or bribery, but let us focus on manipulation – is that it is a *worst-case* measure of hardness. This means that if the manipulation problem is NP-hard, it is unlikely that there is an efficient algorithm that solves *all* instances of the manipulation problem. However, there may still be an efficient algorithm that solves *many* of these instances fast. If so, then computational hardness provides only partial protection to manipulation, at best. It would be much more desirable to show that manipulation is *usually* hard. Recent results have cast doubt on whether this is possible at all. For instance, it was shown that when preferences are single-peaked, many of the manipulation problems that are known to be NP-hard for general preferences become efficiently solvable [46, 111]. In other work on certain distributions of unrestricted preferences, both theoretical and experimental results indicate that manipulation is often computationally easy [e.g., 71, 176, 177, 214, 217, 218]. Extending a previous result by Friedgut et al. [119], Isaksson et al. [139] have recently shown that efficiently manipulable instances are ubiquitous under fairly general conditions.

Theorem 6.4 (Isaksson et al., 2010) *Let f be a neutral resolute voting rule and assume that preferences are uniformly distributed. The probability that a random preference profile can be manipulated by a random voter by submitting random*

preferences is at most polynomially small in $|U|$ and n .

As a consequence, for efficiently computable, neutral, and resolute voting rules, a manipulable preference profile with a corresponding manipulation can easily be found by repeated random sampling. The current state of affairs on using computational hardness to prevent manipulation is surveyed by Faliszewski and Procaccia [107].

3.2.4 Probabilistic Voting Rules

Perhaps the only weakness of the Gibbard-Satterthwaite theorem is that it is restricted to *resolute* voting rules [see, e.g., 206]. As we have seen in Section 3, resoluteness is at variance with elementary fairness conditions such as anonymity and neutrality. The most natural way to break ties yielded by an irresolute voting rule that comes to mind is to pick a single winner at random according to some probability distribution. In order to formalize this, Gibbard [125] proposed an extension of voting rules called *social decision schemes (SDSs)*, which map preference profiles to probability distributions (so-called lotteries) over alternatives.

Of course, the introduction of lotteries raises the question of how voters compare lotteries with each other. The standard approach chosen by Gibbard [125] and subsequent papers [e.g., 11, 126] is to use an expected utility model. In this context, an SDS is strategyproof if, for any utility function that the voter may have over the alternatives, the voter is best off reporting the ordering of the alternatives that corresponds to its true utility function.

Standard examples of non-manipulable SDSs are *random dictator* rules, in which the most preferred alternative of a randomly selected voter is chosen according to a probability distribution that does *not* depend on the voters' preferences. While these rules are clearly fairer than rules with a fixed dictator, they are still not entirely desirable. Unfortunately, when requiring *non-imposition*, i.e., every alternative may be chosen with probability 1 under *some* circumstances (e.g., when all voters unanimously agree), random dictatorships are the only non-manipulable SDSs.

Theorem 6.5 (Gibbard, 1977; Hylland, 1980) *Every non-imposing, non-manipulable SDS is a random dictatorship when $|U| \geq 3$.*

While this might appear like the natural equivalent of the Gibbard-Satterthwaite theorem, it may be argued that non-imposition is rather strong in this context. Gibbard [125] provides a different characterization that uses so-called *duple rules*, in which the outcome is always restricted to two randomly chosen alternatives (e.g., by applying the majority rule to a random pair of alternatives), which no longer seems so unreasonable. Following along these lines,

Barberà [12] and Procaccia [175] provide further examples and characterizations. However, all of these SDSs require an extreme degree of randomization.¹⁵

3.2.5 Irresolute Voting Rules

The definition of manipulability for SDSs rests on strong assumptions with respect to the voters' preferences. In contrast to the traditional setup in social choice theory, which typically only involves ordinal preferences, this model relies on the axioms of von Neumann and Morgenstern in order to compare lotteries over alternatives. The gap between the Gibbard-Satterthwaite theorem for resolute voting rules and Gibbard's theorem for social decision schemes has been filled by a number of impossibility results for *irresolute* voting rules with varying underlying notions of how to compare sets of alternatives with each other [see, e.g., 13, 206].

How preferences over sets of alternatives relate to or depend on preferences over individual alternatives is a fundamental issue that goes back to the foundations of decision making. There is no single correct answer to this question. Much depends on the particular setting considered, the nature of the alternatives, and what we can assume about the personal inclinations of the agent entertaining the preferences. In the context of social choice the alternatives are usually interpreted as mutually exclusive candidates for a unique final choice. For instance, assume a voter prefers a to b , b to c , and – by transitivity – a to c . What can we reasonably deduce from this about the voter's preferences over the subsets of $\{a, b, c\}$? It stands to reason to assume that it would strictly prefer $\{a\}$ to $\{b\}$, and $\{b\}$ to $\{c\}$. If a single alternative is eventually chosen from each choice set, it is safe to assume that the voter also prefers $\{a\}$ to $\{b, c\}$, but whether it prefers $\{a, b\}$ to $\{a, b, c\}$ already depends on (its knowledge about) the final decision process. In the case of a lottery over all preselected alternatives according to a known *a priori* probability distribution with full support, it would prefer $\{a, b\}$ to $\{a, b, c\}$.¹⁶ This assumption is, however, not sufficient to separate $\{a, b\}$ and $\{a, c\}$. Based on a sure-thing principle, which prescribes that alternatives present in both choice sets can be ignored, it would be natural to prefer the former to the latter. Finally, whether the voter prefers $\{a, c\}$ to $\{b\}$ depends on its attitude towards risk: it might be an *optimist* and hope for its most-preferred alternative, or a *pessimist* and fear that its least-preferred alternative will be chosen. One of the most influential negative results for irresolute rules is the *Duggan-Schwartz impossibility* [92].

¹⁵An important extension of this model studies SDSs in which the von Neumann-Morgenstern utility functions of the voters rather than their preference relations are aggregated [see, e.g., 14, 96, 137].

¹⁶The posterior distribution is obtained by conditioning on the selected subset. This rules out inconsistent lotteries like always picking b from $\{a, b\}$ and a from $\{a, b, c\}$.

Theorem 6.6 (Duggan and Schwartz, 2000)

Every non-imposing, non-dictatorial voting rule can be manipulated by an optimist or pessimist when $|U| \geq 3$.

However, for weaker (incomplete) preference relations over sets, more positive results can be obtained [e.g., 39, 41]. Brandt [39], for instance, has shown that the minimal covering set and the bipartisan set (mentioned in Section 2.2.2) are non-manipulable when one set of alternatives is preferred to another if and only if everything in the former is preferred to everything in the latter.

3.3 Possible and Necessary Winners

Two commonly studied computational problems in voting are the *possible winner problem* and the *necessary winner problem* [142]. The input to these problems is a partially specified profile of votes and a distinguished alternative c ; we are asked whether there exists some completion of the profile that results in c winning (possible winner) or whether c will in fact win no matter how the profile is completed (necessary winner).¹⁷

There are several important motivations for studying these problems. One derives from the *preference elicitation problem*, where we repeatedly query voters for parts of their preferences until we know enough to determine the winner. The necessary winner problem is of interest here, because at an intermediate stage in the elicitation process, we will know the profile partially and may wish to know whether we can safely terminate and declare c the winner. (The computational problem of determining whether elicitation is done was explicitly studied by Conitzer and Sandholm [67].) The possible and necessary winner problems also have as a special case the problems faced by a set of manipulators who know the subprofile of the other voters (when they wish to make a given alternative win or prevent a given alternative from winning, respectively). This latter observation allows us to easily transfer hardness results for the manipulation problem to the possible and necessary winner problems. In general, however, the possible and necessary winner problems can be even harder than the corresponding manipulation problems, because the possible and necessary winner problems generally also allow individual votes to be only partially specified (which makes sense under the elicitation interpretation, because we may so far have asked voters to compare only certain pairs of alternatives, and not others).¹⁸

¹⁷Other work has considered this problem in the setting where instead of uncertainty about the profile, there is uncertainty about the voting rule [150].

¹⁸Recent work has also studied a version of the manipulation problem where the profile of nonmanipulator votes is only partially known to the manipulator [79], which is another problem that is closely related to the possible/necessary winner problem.

A natural way of expressing partial knowledge about the voters' preferences is to have a *partial order* over the alternatives associated with every voter. The idea is that we know that the voters' preferences must be some linear order that extends that partial order. The computational complexity of this problem for various voting rules has been determined by Xia and Conitzer [221]. The possible winner problem is NP-complete for rules including STV, scoring rules including Borda and *k*-approval,¹⁹ Copeland, maximin, Bucklin, and ranked pairs. The necessary winner problem is coNP-complete for all these except scoring rules, maximin, and Bucklin, for which it can be solved in polynomial time. For plurality and anti-plurality, both problems can be solved in polynomial time.

4 Combinatorial Domains

So far we have presented the classical mathematical framework for studying different variants of the problem of social choice and we have seen examples of questions regarding the computational properties of this framework. Next, we will consider a social choice problem where computational considerations already play a central role at the level of *defining* the formal framework to study this problem. The problem in question is the problem of *social choice in combinatorial domains*. To simplify matters, we will focus specifically on *voting* in combinatorial domains.

Let us begin with an example. Suppose three agents need to agree on a menu for dinner. The options for the starter are *salad* and *oyster*; the options for the main course are *trout* and *veal*; and the options for the wine are *red* and *white*. The favorite menus of our three agents are as follows.

Agent 1: *salad-trout-white*

Agent 2: *salad-veal-red*

Agent 3: *oyster-veal-white*

Agent 1 likes trout and naturally wants to combine this with a white wine; agent 2 likes veal (which may be paired with either red or white wine) and has a preference for red wine; and agent 3 likes oyster and veal, which calls for a white wine. Now, what menu should our agents choose as a group, and how should they make that choice? Maybe the most natural approach is to use the *plurality rule* on each of the three issues: there is a majority for salad, there is a majority for veal, and there is a majority for white wine. That is, the group menu will be *salad-veal-white*. But this very conceivably could be one of the worst possible choices for our agents: like agent 2, they may very well all prefer to have a red wine with salad and veal.

¹⁹The complexity of the possible winner problem for scoring rules has been completely characterized by Betzler and Dorn [22] and Baumeister and Rothe [21].

What went wrong here? The problem is that the preferences of the agents over the choices made for each of the three issues are not *independent*. For instance, our little story suggested that for all of them their preferred choice of wine depends on what starter and main course they will actually get served. But voting issue-by-issue completely ignores this dependency, and so we should not be too surprised if we get a paradoxical outcome.

Note also that the next most obvious approach, which would be to directly vote on full menus, does not work very well either. If we ask each agent only for its most preferred menu (as we have done above), we will typically get three different answers, and the best we can do is to randomly select one of the three. We could refine this approach further, and ask, say, for their five most preferred menus and apply, say, the Borda rule. This might lead to an acceptable solution in our little example, but imagine we are dealing with a choice problem with 10 binary issues and thus $2^{10} = 1,024$ alternatives: the most preferred alternatives of our three agents might very well be entirely disjoint again.

A full description of our example should actually list the full preferences of each of our three agents over the *combinatorial domain* $D = \{\text{salad}, \text{oyster}\} \times \{\text{trout}, \text{veal}\} \times \{\text{red}, \text{white}\}$, i.e., over a set of eight alternatives. Note that the number of alternatives is *exponential* in the number of issues. But this means that even for examples with a slightly larger number of issues it can quickly become practically infeasible for the agents to rank all the alternatives and communicate this ranking. That is, there is a fundamental computational challenge hidden at the very heart of voting in combinatorial domains: even a small problem description immediately gives rise to a very large choice problem.

In our little example there actually is a good solution: For all three agents, their preferences regarding the wine depend on the choices made for the starter and the main course, while their preferences for those two issues do not depend on anything else (we have not actually described our example in enough detail before to be sure about the latter fact, but let us now assume that this is indeed the case). We can use these dependencies to determine a good order in which to vote on each of the three issues *in sequence*. As long as we vote on the wine at the very end, there will not be any paradoxical outcome (nor will there be any computational difficulty).²⁰

So, if we first use the plurality rule to choose a starter and a main course, our agents are likely to choose the salad and the veal. If we then fix these choices and ask the agents to vote on the wine, they will select the red wine, yielding an outcome (*salad-veal-red*) that is ideal for agent 2 and not unreasonable for the other two.

²⁰This is assuming that agents do not vote strategically; we will discuss this point more at the end of Section 4.2.

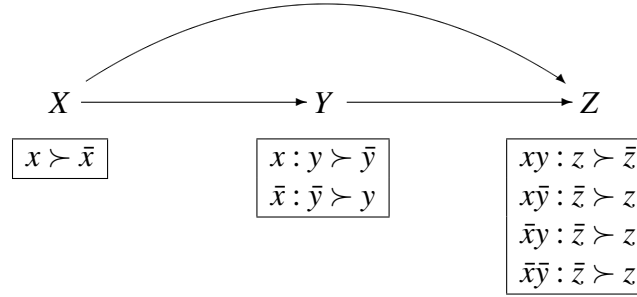
The kind of paradox we have seen has long been observed and studied in political science, typically under the name of “multiple-election paradoxes” [35]. As a problem that is inherently computational in nature it was first formulated by Lang [148].

As the representation of an agent’s preferences plays a central role in social choice in combinatorial domains, we will first review the most important knowledge representation languages that have been used in the literature to this end. We will then focus on two types of promising approaches: sequential voting and voting by means of compactly represented preferences.

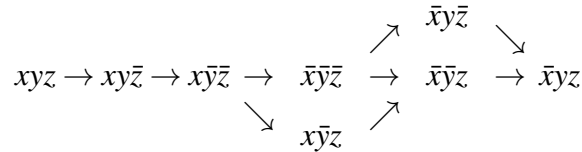
4.1 Preference Representation

Suppose we want to model an agent’s preferences over a combinatorial domain defined by ℓ binary variables, i.e., over a domain with 2^ℓ alternatives. There are $(2^\ell)!$ different linear orders that might represent the agent’s preferences, and there are even more possibilities if we want to also consider weak or partial orders. Encoding such a linear order thus requires at least $\log(2^\ell!)$, i.e., $O(\ell \cdot 2^\ell)$, bits. If we use an *explicit representation* that specifies for each pair of alternatives whether or not our agent prefers the first of them over the second, we even need $O(2^\ell \cdot 2^\ell)$ bits (one per pair). This will generally not be feasible in practice. Instead, we require a *compact* preference representation language that will allow us to model those preference structures that we can expect to occur in a given application scenario using short expressions in that language. When choosing such a language, we should consider its *expressive power* (which preference structures can it express?), its relative *succinctness* (can it do so using significantly less space than a given rival language?), its *complexity* (how hard is it to reason about preferences expressed in the language?), its *elicitation-friendliness* (does it support efficient elicitation of preferences from the agents?), and its *cognitive adequacy* (is it a “natural” form of describing preferences?) [54].

The most widely used language for compact preference representation used in computational social choice is *conditional preference networks*, or CP-nets [31]. The basic idea is to induce a preference order from statements of the form “if condition C holds, then – everything else being equal – I prefer variable X to take value x rather than value \bar{x} ”. A CP-net is based on a directed graph on the set of variables defining the combinatorial domain in question. Every vertex in the graph is annotated with a table that specifies, for each possible instantiation of the variables corresponding to the parents of that vertex, a preference order over the possible values of the variable corresponding to that vertex. Let us consider an example. Suppose our domain is defined by means of three variables: X (with possible values x and \bar{x}), Y (with possible values y and \bar{y}), and Z (with possible values z and \bar{z}). A CP-net for this domain might look like this:



A CP-net induces a partial order: if two alternatives differ only in the instantiation of a single variable, then we can look up the corresponding entry in the table for that variable to find how the two alternatives should be ranked. The full partial order is the transitive closure of the relations we obtain by interpreting the individual preference statements in this manner. For instance, given the CP-net above, we prefer $xy\bar{z}$ to $x\bar{y}\bar{z}$, because the two differ only in their assignment to Y , and the first statement in the table for variable Y says that when $X = x$, then we should prefer $Y = y$ over $Y = \bar{y}$, everything else being equal (i.e., $Z = \bar{z}$ in both cases). The full preference relation induced by the CP-net above is the following partial order (where an arrow represents \succ and the rankings obtained by transitivity are not shown explicitly):



Note that, for instance, $x\bar{y}\bar{z}$ and $x\bar{y}z$ are *incomparable*: the CP-net does not specify which of the two the agent prefers.

Another important family of languages for preference representation is that of *prioritized goals* [81, 148]. Prioritized goals are applicable when each of the variables defining the combinatorial domain has exactly two possible values (e.g., *true* and *false*, or 1 and 0). The basic idea is to describe the goals of the agent whose preferences we are modeling as formulas of propositional logic. For example, the formula $X \vee Y$ expresses the goal of having at least one of the variables X and Y take the value *true*, while $X \rightarrow \neg(Y \wedge Z)$ says that whenever X is *true*, then it should not be the case that both Y and Z are *true* as well. Usually not all goals will be satisfiable. An agent can indicate the importance of each of its goals by labeling it with a number, its priority level (suppose a higher number indicates higher importance). Different interpretations of this kind of language are possible. One choice is the *lexicographic* interpretation, under which we prefer alternative a to alternative b if there exists a k such that for every priority level above k both

a and b satisfy the same number of goals, while a satisfies more goals of priority level k than b does. For example, if an agent has the goals X and $\neg Y$ and the former has higher priority than the latter, then this induces the preference order $xy \succ xy \succ \bar{x}\bar{y} \succ \bar{x}y$.

Both CP-nets and prioritized goals define preferences that are ordinal: either linear or weak orders, as commonly used in classical social choice theory, or partial orders and preorders, which can be particularly appropriate for applications in multiagent systems, where we may want to model explicitly the fact that an agent has bounded rationality and is lacking either the computational resources or the necessary information to completely rank all possible pairs of alternatives. In Section 5.1, in the context of discussing fair division problems, we will see further examples of preference representation languages, namely languages for modeling cardinal preferences (i.e., valuation functions).

4.2 Sequential Voting

In the example above, we already mentioned the idea of voting over the issues one at a time, in sequence. This is a very natural idea and requires relatively little communication. A downside of sequential voting is that an agent's preferences for the current issue may depend on issues on which we have not yet decided. (Above, avoiding such a situation was our motivation for deciding on the wine last.) CP-nets allow us to formalize this idea. We say that a CP-net is *legal* for an order over the issues if its graph does not have any edges pointing from issues that are later in the order to ones that are earlier (which immediately implies that the graph is acyclic). As a result, if all the agents' CP-nets are legal for the order in which we vote over issues, then each agent's CP-net will always unambiguously specify the agent's preferences over the current issue, because we will have already decided on the values of the issues on which these preferences depend. This also means that the agents do not actually need to vote on each issue separately; they can just submit their CP-nets, and then leave. These ideas and the properties of sequential voting are discussed in detail by Lang and Xia [149].

It is of course still possible to force agents to use sequential voting even if their preferences for earlier issues do depend on later issues, but in this case it is no longer clear how they should vote. One possibility is to assume that agents vote strategically, thinking ahead toward what is likely to happen regarding later issues (which may also depend on how they vote on the current issue). It should be noted that this will, in general, change how the agents vote even when their preferences for earlier issues do not depend on later issues. Unfortunately, as we have discussed earlier, even when there is only a single issue, strategic voting is quite complicated when that issue can take three or more possible values – for example, the corresponding game has multiple equilibria. On the other hand, if

we assume that each issue can take only two possible values and that the agents' true preferences are common knowledge, then it is clear how to vote strategically. This is because in the last round (when voting over the last issue) the agents are effectively voting over the two remaining alternatives, so each agent is best off voting for its preferred one; based on this, in the second-to-last round, the agents can predict which alternative will end up chosen in the last round as a function of which value the current (second-to-last) issue ends up taking, so effectively the agents are deciding between the corresponding two alternatives; and so on. Specifically, this means that under these circumstances, strategic sequential voting is bound to result in the election of the Condorcet winner, whenever it exists [145].

On the other hand, Xia et al. [226] show that, unfortunately, for some profiles without a Condorcet winner, strategic sequential voting results in very poor outcomes; in fact, this happens even when the agents' preferences for earlier issues never depend on the agents' preferences for later issues, because they will not necessarily vote truthfully. (Incidentally, the strategic sequential voting process is a special case of *multistage sophisticated voting* [128, 159, 162].) Xia et al. [226] also show that the outcome can be very sensitive to the order in which the issues are voted on, potentially giving the agenda setter a significant amount (or even complete) control over the outcome. The complexity of this control problem has been studied in a non-strategic context [75].

4.3 Voting with Compactly Represented Preferences

Considerations of strategic voting aside, sequential voting is an appealing procedure when each agent's preferences are represented by a CP-net that is legal with respect to the same ordering. But what if this is not the case? For one, the agents may require different orders. For example, consider one agent who prefers veal to trout regardless of which wine is chosen, but whose preferred wine depends on which meal is served; and another agent who prefers red to white wine regardless of which meal is served, but whose preferred meal depends on which wine is served. For the former agent, it would be ideal to vote on the meal first, but for the latter, it would be ideal to vote on the wine first. Clearly, there is no way to order the issues that will make both agents comfortable voting on the first issue without knowing what value the second issue will take. Moreover, it is not necessarily the interaction between multiple agents' preferences that causes trouble: it is even possible for a single agent's preferences to conflict with the idea of sequential voting. For example, consider an agent who mostly cares that the wine fits the meal, and ranks the different meal combinations $(trout, white) \succ (veal, red) \succ (veal, white) \succ (trout, red)$. For this agent, no ordering of the issues is satisfactory, because its preference for each issue depends on the other issue – its CP-net is cyclic.

How can we address such preferences, without falling back on making the agents rank all the exponentially many alternatives, but rather by making use of a compact preference representation language, such as CP-nets? This problem has been introduced by Lang [148] under the name of “combinatorial voting,” i.e., voting by means of ballots that are compactly represented preference structures. Unfortunately, the computational complexity of this approach is often prohibitively high. For example, Lang [148] shows that computing the election winners – when each voter specifies its preferences using the language of prioritized goals and (a suitable generalized form of) the plurality rule is used – is coNP-hard, even when each voter only states a single goal. Similarly, deciding whether there exists a Condorcet winner is coNP-hard under the same conditions. For languages that can express richer preference structures, the complexity of winner determination will typically be beyond NP.

One useful property of preferences represented by a CP-net is that, if we hold the values of all but one issue fixed, then the CP-net specifies the agent’s preferences over that remaining issue. While it is not computationally efficient to do so, conceptually, we can consider, for every issue and for every possible setting of the other issues, all agents’ preferences. We can then choose winners based on these “local elections” [78, 153, 223]. For example, we can look for an alternative that defeats all of its neighboring alternatives (that is, the alternatives that differ on only one issue from it) in pairwise elections. Of course, there may be more than one such alternative, or none. The maximum likelihood approach mentioned earlier in this chapter has also been pursued in this context [225].

Developing practical algorithms for voting in combinatorial domains is one of the most pressing issues on the research agenda for computational social choice in the near future.

5 Fair Division

So far we have discussed social choice, in its most general and abstract form, as the problem of choosing one or several “alternatives,” or as the problem of ranking them. An alternative might be a candidate to be elected to political office or it might be a joint plan to be executed by a group of software agents. In principle, it might also be an allocation of resources to a group of agents. In this section, we specifically focus on this latter problem of *multiagent resource allocation*.

To underline our emphasis on fairness considerations we shall favor the term *fair division*. In the economics literature, the broad research area concerned with determining a fair and economically efficient allocation of resources in society is known as *welfare economics*. We will introduce some of the fundamental concepts from this literature, discuss them from an algorithmic point of view, and review

their relevance to multiagent systems.

Fair division differs from the other types of social choice problems discussed in this chapter in at least two important respects:

1. Fair division problems come with a rich *internal structure*: alternatives are allocations of goods to agents and an agent's preferences are usually assumed to only depend on their own bundle.
2. In the context of fair division problems, preferences are usually assumed to be *valuation functions*, mapping allocations to numerical values, rather than binary relations for ranking allocations.

Below (in Section 5.1) we will therefore begin by reviewing preference representation languages for compactly modeling such valuation functions.

First, however, we need to fix the type of goods to be allocated. The main line of differentiation is between *divisible* and *indivisible* goods. A classical example of the former scenario is the fair division of a cake. While there have been a number of contributions to the cake-cutting literature in theoretical computer science and more recently also in artificial intelligence, to date, most work in multiagent systems has concentrated on indivisible goods. We shall therefore only give one example here, which is illustrative of the simple and elegant solutions that have been obtained in the field of cake-cutting. Consider the *moving-knife procedure* due to Dubins and Spanier [91]:

A referee moves a knife across the cake, from left to right. Whenever an agent shouts "stop," he receives the piece to the left of the knife and leaves.

Under standard assumptions on the agents' valuation functions (namely, *continuity* and *additivity*), this procedure is *proportional*: it guarantees each agent at least $\frac{1}{n}$ of the full value of the cake, according to its own valuation, whatever the other agents may do (where n is the number of agents). To see this, observe that you are free to shout "stop" when the knife reaches a point where the piece to be cut would be exactly $\frac{1}{n}$ according to your valuation; and if another agent shouts "stop" earlier, then that only means that it will leave with a piece that you consider to be of less value than $\frac{1}{n}$. The books by Brams and Taylor [37] and by Robertson and Webb [182] both provide excellent expositions of the cake-cutting problem.

For the remainder of this section we will focus on the problem of fairly allocating a set of indivisible goods to a group of agents. After introducing several languages for representing preferences in this context, we define the most important criteria for measuring fairness and economic efficiency, and we review examples of work in computational social choice concerning the complexity of computing a fair allocation and designing protocols that can guarantee convergence to a fair

allocation. For a more extensive review of the variety of multiagent resource allocation problems and computational aspects of fairness than is possible here we refer to the survey article by Chevaleyre et al. [54].

5.1 Preference Representation

Let G be a finite set of indivisible *goods*, with $\ell = |G|$. Each agent may receive any subset of G . The preferences of agent $i \in N$ are given by means of a *valuation function* $v_i : 2^G \rightarrow \mathbb{R}$, mapping every bundle it might receive to the value it assigns to it. Valuation functions are often assumed to be *monotonic*, i.e., for any two sets of goods S and S' , it will be the case that $v_i(S) \leq v_i(S')$ whenever $S \subseteq S'$. This assumption is also known as *free disposal*. For many applications it makes sense to assume that valuation functions are monotonic, while for others we also need to be able to model undesirable goods.

An *explicit representation* of a valuation function v_i will often not be feasible in practice, as it requires us to specify a list of 2^ℓ numbers. However, if valuations are “well-behaved” in the sense of exhibiting some structural regularities, then a compact representation using a suitable preference modeling language will often be possible.

A powerful family of languages, closely related to the prioritized goal languages discussed in Section 4.1, is based on *weighted goals*. This language originates in the work on penalty logic by Pinkas [172] and variants of it have been used in many areas of artificial intelligence and elsewhere. Its relevance for preference representation in the context of social choice has first been recognized by Lafage and Lang [146]. The basic idea is again to have an agent express its goals in terms of formulas of propositional logic and to assign numbers, or weights, to these goals to indicate their importance. We can then aggregate the weights of the goals satisfied by a given alternative to compute the value of that alternative. The most widely used form of aggregation is to take the *sum* of the weights of the satisfied goals. For example, suppose G is a set of three goods, associated with the propositional variables p , q , and r . An agent providing the weighted goals $(p \vee q, 5)$ and $(q \wedge r, 3)$ expresses the following valuation function:

$$\begin{array}{llll}
 v(\emptyset) & = & 0 & v(\{p, q\}) & = & 5 \\
 v(\{p\}) & = & 5 & v(\{p, r\}) & = & 5 \\
 v(\{q\}) & = & 5 & v(\{q, r\}) & = & 8 \\
 v(\{r\}) & = & 0 & v(\{p, q, r\}) & = & 8
 \end{array}$$

That is, obtaining one of p and q (or both) has value 5 for the agent, and in the event it obtains the latter, obtaining r on top of it results in an additional value of 3.

By putting restrictions on the kinds of formulas we want to admit to describe goals, we can define different languages. For instance, we may only permit conjunctions of atomic formulas, or we may only allow formulas of length at most 3, and so forth. Different such languages have different properties, in view of their *expressive power*, in terms of their *succinctness* for certain classes of valuation functions, and regarding the *computational complexity* of basic reasoning tasks, such as computing the most preferred bundle for an agent with a given set of weighted goals. For full definitions and a detailed analysis of these properties, we refer to the work of Uckelman et al. [211].

Weighted goal languages are closely related to other languages to be found in the literature. For instance, *k-additive functions*, studied in measure theory [127], correspond to the weighted goal language we obtain when the only admissible logical connective is conjunction and when each formula may involve at most k propositional variables [211]. In cooperative game theory, *marginal contribution nets* [138], a language for modeling coalitional games, correspond to the language of conjunctions of literals of arbitrary length.²¹ Weighted goal languages have also been studied for other forms of aggregation than summing up the weights of the satisfied goals [146, 210].

Preference representation languages also play an important role in the literature on *combinatorial auctions* [82], where they are known as *bidding languages*. In an auction, each bidder has to describe its valuation of the goods on sale to the auctioneer, i.e., a bid amounts to the specification of a valuation function (whether or not the bidder does so *truthfully* is irrelevant for the representation problem at hand). The idea of using weighted goals has been used also in this domain [30]. The most widely used basic bidding languages, however, belong to the *OR/XOR family*. An *atomic bid* is a bundle of goods together with a price, e.g., $(\{p, q\}, 7)$. Each bidder can provide any number of atomic bids. Under the *OR-language*, the value of a bundle for the bidder is the maximum price that we can obtain by assigning each item in the bundle to (at most) one atomic bid and summing up the prices for those atomic bids that are covered completely by this assignment. For example, given the bid $(\{p, q\}, 7) \text{ OR } (\{p, r\}, 5) \text{ OR } (\{r\}, 3)$, the value of the bundle $\{p, q, r\}$ is $7 + 3 = 10$. Under the *XOR-language* the value of a bundle is the price of the most valued atomic bid it can cover. That is, the XOR-language is like the explicit representation mentioned earlier (together with an implicit monotonicity assumption). Combinations of OR and XOR have also been studied. Full definitions and results regarding the expressive power and succinctness of different languages are available in a review article by Nisan [169].

²¹In some expositions of marginal contribution nets the restriction to conjunctions of literals is not imposed, in which case we obtain the general language of weighted goals (see, e.g., the chapter on “Computational Coalition Formation” in this book).

Yet another option is to think of a valuation function as a *program* that takes bundles as inputs and returns values as output. In the context of fair division, this idea has been explored in the work of Dunne et al. [94].

While most work in fair division assumes that preferences are given in the form of valuation functions, the problem of fairly dividing goods over which agents have ordinal preferences is also interesting. CP-nets, the most important language for research on voting in combinatorial domains, is only of very limited interest here, because CP-nets cannot express most *monotonic* preferences. A possible alternative are so-called *conditional importance networks*, or CI-nets [34].

5.2 Fairness and Efficiency

What makes a *fair* allocation of resources? More generally, what makes a *good* allocation? Next we shall review several proposals for measuring the quality of an allocation. The first set of proposals is based on the idea of a *collective utility function*. Any given allocation yields some utility $u_i \in \mathbb{R}$ for agent i . This utility will usually be the result of applying agent i 's valuation function to the bundle it receives under the allocation in question. Now we can associate an allocation with a *utility vector* $(u_1, \dots, u_n) \in \mathbb{R}^n$.

Definition 6.5 A collective utility function (CUF) is a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

That is, a CUF returns a single collective utility value for any given utility vector (which in turn we can think of as being generated by an allocation). This collective utility is also referred to as the *social welfare* of the corresponding allocation. The following are the most important CUFs studied in the literature:

- Under the *utilitarian* CUF, $f_u(u_1, \dots, u_n) := \sum_{i \in N} u_i$, i.e., the social welfare of an allocation is the sum of the utilities of the individual agents. This is a natural way of measuring the quality of an allocation: the higher the *average* utility enjoyed by an agent, the higher the social welfare. On the other hand, this CUF hardly qualifies as *fair*: an extra unit of utility awarded to the agent currently best off cannot be distinguished from an extra unit of utility awarded to the agent currently worst off. Note that authors simply writing about “social welfare” are usually talking about utilitarian social welfare.
- Under the *egalitarian* CUF, $f_e(u_1, \dots, u_n) := \min\{u_i \mid i \in N\}$, i.e., the social welfare of an allocation is taken to be the utility of the agent worst off under that allocation. This CUF clearly does focus on fairness, but it is less attractive in view of economic efficiency considerations. In the special

case where we are only interested in allocations where each agent receives (at most) one item, the problem of maximizing egalitarian social welfare is also known as the *Santa Claus problem* [9].

- A possible compromise is the *Nash CUF*, under which $f_n(u_1, \dots, u_n) := \prod_{i \in N} u_i$. Like the utilitarian CUF, this form of measuring social welfare rewards increases in individual utility at all levels, but more so for the weaker agents. For instance, the vectors $(1, 6, 5)$ and $(4, 4, 4)$ have the same utilitarian social welfare, but the latter has a higher Nash product (and intuitively is the fairer solution of the two). For the special case of just two agents, the Nash product is discussed in more detail in the chapter on “Negotiation and Bargaining” in this book.

Any CUF gives rise to a *social welfare ordering* (SWO), a transitive and complete order on the space of utility vectors (in the same way as an individual utility function induces a preference relation). We can also define SWOs directly. The most important example in this respect is the *leximin ordering*. For the following definition, suppose that all utility vectors are ordered, i.e., $u_1 \leq u_2 \leq \dots \leq u_n$. Under the leximin ordering, (u_1, \dots, u_n) is socially preferred to (v_1, \dots, v_n) if and only if there exists a $k \leq n$ such that $u_i = v_i$ for all $i < k$ and $u_k > v_k$. This is a refinement of the idea underlying the egalitarian CUF. Under the leximin ordering, we first try to optimize the well-being of the worst-off agent. Once our options in this respect have been exhausted, we try to optimize the situation for the second worst-off agent, and so forth.

SWOs have been studied using the axiomatic method in a similar manner as SWFs and SCFs. Let us briefly review three examples of axioms considered in this area.

- An SWO \succsim is *zero independent* if $u \succsim v$ entails $(u + w) \succsim (v + w)$ for any $w \in \mathbb{R}^n$. That is, according to this axiom, social judgments should not change if some of the agents change their individual “zero point.” Zero independence is the central axiom in a characterization of the SWOs induced by the utilitarian CUF [83, 165].
- An SWO \succsim is *independent of the common utility pace* if $u \succsim v$ entails $(g(u_1), \dots, g(u_n)) \succsim (g(v_1), \dots, g(v_n))$ for any increasing bijection $g : \mathbb{R} \rightarrow \mathbb{R}$. You might think of g as a function that maps gross to net income. Then the axiom says that we want to be able to make social judgments independently from the details of g (modeling the taxation laws), as long as it never inverts the relative welfare of two individuals. The utilitarian SWO fails this axiom, but the egalitarian SWO does satisfy it.

- An SWO \succsim satisfies the *Pigou-Dalton principle* if $u \succsim v$ whenever u can be obtained from v by changing the individual utilities of only two agents in such a way that their mean stays the same and their difference reduces. The Pigou-Dalton principle plays a central role in the axiomatic characterization of the leximin ordering [165].

For an excellent introduction to the axiomatics of welfare economics, providing much more detail than what is possible here, we refer to the book of Moulin [165]. Broadly speaking, the additional information carried by a valuation function (on top of its ordinal content, i.e., on top of the kind of information used in voting theory), avoids some of the impossibilities encountered in the ordinal framework. For instance, if we enrich our framework with a monetary component and stipulate that each agent's utility can be expressed as the sum of that agent's money and its valuation for its goods (so-called *quasilinear preferences*), then we can define strategyproof mechanisms that are not dictatorial. Examples are the mechanism used in the well-known Vickrey auction and its generalizations (see the chapter on "Mechanism Design and Auctions" in this book).

Another important fairness criterion is *envy-freeness*. An allocation A of goods is envy-free if no agent would rather obtain the bundle allocated to one of the other agents: $v_i(A(i)) \geq v_i(A(j))$ for any two agents i and j , with $A(i)$ and $A(j)$ denoting the bundles of goods allocated to i and j , respectively. Note that this concept cannot be modeled in terms of a CUF or an SWO. If we insist on allocating all goods, then an envy-free allocation will not always exist. A simple example is the case of two agents and one item that is desired by both of them: in this case, neither of the two complete allocations will be envy-free. When no envy-free allocation is possible, then we might want to aim for an allocation that minimizes the *degree of envy*. A variety of definitions for the degree of envy of an allocation have been proposed in the literature, such as counting the number of agents experiencing some form of envy or counting the pairs of agents where the first agent envies the second [55, 154, 155].

A new application in multiagent systems may very well call for a new fairness or efficiency criterion. However, any new idea of this kind should always be clearly positioned with respect to the existing standard criteria, which are well motivated philosophically and deeply understood mathematically.

5.3 Computing Fair and Efficient Allocations

Once we have settled on a language for modeling the valuation functions of individual agents and on an efficiency or fairness criterion we want to apply, the question arises of how to compute an optimal allocation of goods. Algorithmic methods that have been used in this field include linear and integer programming,

heuristic-guided search, and constraint programming. Rather than discussing specific algorithms here, let us focus on the computational complexity of the combinatorial optimization problems such an algorithm would have to tackle.

First, suppose we want to compute an allocation with maximal utilitarian social welfare. In the combinatorial auction literature, this problem is generally known as the “winner determination problem” and it has received a large amount of attention there [82]. The goal is to maximize the sum of the valuations of the individual agents. As complexity theory is more neatly applied to decision problems, let us consider the corresponding decision problem:

We are given a profile of valuation functions (v_1, \dots, v_n) , one for each agent, and a number K and ask whether there exists an allocation $A : N \rightarrow 2^G$ mapping agents to bundles of goods, with $A(1) \cup \dots \cup A(n) = G$ and $A(i) \cap A(j) = \emptyset$ for any two agents i and j , such that $f_u(v_1(A(1)), \dots, v_n(A(n))) \geq K$, i.e., such that the utilitarian social welfare of A is at least K .

This problem turns out to be NP-complete for any of the preference representation languages discussed [54, 82]. To show that it is in NP is usually easy. The fact that it is NP-hard was first observed by Rothkopf et al. [187], in the context of what is now known as the OR-language. The proof proceeds via a reduction from a standard NP-hard problem known as SET PACKING [121]: given a collection of finite sets \mathcal{C} and a number K , is there a collection $\mathcal{C}' \subseteq \mathcal{C}$ of pairwise disjoint sets such that $|\mathcal{C}'| \geq K$? Now consider any preference representation language that allows us to specify for each agent i one bundle B_i such that $v_i(B) = 1$ if $B = B_i$ (or possibly $B \supseteq B_i$) and $v_i(B) = 0$ otherwise (weighted goals, the OR-language, and the XOR-language can all do this). Then we have a one-to-one correspondence between finding an allocation with a utilitarian social welfare of at least K and finding at least K non-overlapping bundles B_i . Hence, welfare optimization must be at least as hard as SET PACKING. (The problem has also been shown to be NP-hard to approximate [190, 233].)

Let us now briefly go over some related complexity results, but with less attention to detail. Rather than stating these results precisely, we focus on some of the crucial insights they represent and cite the original sources for further details.

- Computing allocations that are optimal under the egalitarian CUF or the Nash CUF is also NP-hard [32, 180]. In case all valuation functions are *additive*, i.e., if we can always compute the value of a bundle by adding the values of the individual items in that bundle, then computing an allocation with maximal utilitarian social welfare becomes easy (simply assign each item to the agent giving it the highest value), but the same domain restriction does not render the problem polynomial when we are interested in egalitarian social welfare or the Nash product.

- An allocation A is Pareto optimal if there is no other allocation A' that is strictly preferred by some agent and not worse for any of the others. Deciding whether a given allocation is Pareto optimal is typically coNP-complete [54, 57, 87, 94]. The crucial difference with respect to optimizing utilitarian social welfare is that we now have to check that there is *no* other allocation that is “better” (hence the switch in complexity class).
- Computing allocations that are envy-free can be significantly more difficult.²² Bouveret and Lang [33] show that deciding whether there exists an allocation that is both Pareto optimal and envy-free is Σ_2^P -complete when preferences are represented using weighted goals, even when each agent can only distinguish between “good” (valuation 1) and “bad” (valuation 0) bundles. When valuations are additive, then deciding whether there exists an envy-free allocation (that allocates all goods) is still NP-complete [155]. Lipton et al. [155] also discuss approximation schemes for envy-freeness.

5.4 Convergence to Fair and Efficient Allocations

The complexity results we have reviewed above apply in situations where we have complete information regarding the individual preferences of the agents and we want to compute a socially optimal allocation in a centralized manner. Of course, this is an idealized situation that we will rarely encounter in a multiagent system in practice. Besides the algorithmic challenges highlighted above, we also need to face the game-theoretical problem of ensuring that agents report their preferences truthfully (in case we consider truthfulness an important desideratum). We need to design a suitable elicitation protocol to obtain the relevant preference information from the agents.

An alternative approach, which we shall discuss next, is as follows: rather than centrally collecting all the agents’ preference information and determining an optimal allocation, we let agents locally find utility-improving deals that involve the exchange of some of the goods in their possession. We can then analyze the effects that sequences of such local trading activities have on the allocations emerging at the global level. If we give up control in this manner, we might not always be able to reach a socially optimal allocation. Instead, we now have to ask what quality guarantees we can still provide. This *distributed approach to multiagent resource allocation* requires us to fix a set of assumptions regarding the local behavior of agents. For instance, we could make the assumption that each agent is a utility-maximizer in the full game-theoretic sense. Often this will be unrealistic, given the high complexity of computing one’s optimal strategy under all circumstances

²²One allocation that is always envy-free is the one where nobody gets anything. To prevent such trivialities, normally some efficiency requirement is added as well.

in this context. Instead, let us assume that agents are *individually rational* and *myopic*. This means that we assume that an agent will agree to participate in a deal if and only if that deal increases the agent's utility. On the other hand, it will not try to optimize its payoff in every single deal and it does not plan ahead beyond the next deal to be implemented.

Formally, a *deal* is a pair of allocations $\delta = (A, A')$ with $A \neq A'$, describing the situation before and after the exchange of goods. Note that this definition permits exchanges involving any number of agents and goods at a time. Bilateral deals, involving only two agents, or simple deals, involving only one item, are special cases. For the result we want to present in some detail here, we assume that a deal may be combined with monetary *side payments* to compensate some of the agents for a loss in utility. This can be modeled via a function $p : N \rightarrow \mathbb{R}$, mapping each agent to the amount it has to pay (or receive, in case $p(i)$ is negative), satisfying $p(1) + \dots + p(n) = 0$, i.e., the sum of all payments made must equal the sum of all payments received. A deal $\delta = (A, A')$ is *individually rational* if there exists a payment function p such that $v_i(A'(i)) - v_i(A(i)) > p(i)$ for all agents $i \in N$, with the possible exception of $p(i) = 0$ in case $A(i) = A'(i)$. That is, a deal is individually rational if payments can be arranged in such a way that for each agent involved in the deal its gain in valuation exceeds the payment it has to make (or its loss in valuation is trumped by the money it receives). We shall assume that every deal made is individually rational in this sense. Note that we do not *force* agents to make deals under these conditions; we simply assume that any deal that is implemented is (at least) individually rational.

Now, by a rather surprising result due to Sandholm [189], any sequence of individually rational deals must converge to an allocation with maximal utilitarian social welfare. That is, provided all agents are individually rational and continue to make individually rational deals as long as such deals exist, we can be certain that the resulting sequence of deals must be finite and that the final allocation reached must be socially optimal in the sense of maximizing utilitarian social welfare. For a detailed discussion and a full proof of this result we refer to the work of Endriss et al. [104]. The crucial step in the proof is a lemma that shows that, in fact, a deal is individually rational if and only if it increases utilitarian social welfare. Convergence then follows from the fact that the space of possible allocations is finite.

How useful is this convergence result in practice? Of course, the complexity results discussed in Section 5.3 did not just go away: finding an allocation that maximizes utilitarian social welfare is still NP-hard. Indeed, to decide whether it is possible to implement yet another individually rational deal, our agents do have to solve an NP-hard problem (in practice, most of the time they might find it easy to identify a good deal, but in the worst case this can be very hard). Also the *struc-*

tural complexity of the deals required (in the worst case) is very high. Indeed, if our agents use a negotiation protocol that excludes deals involving a certain number of agents or goods, then convergence cannot be guaranteed any longer [104]. On the other hand, a simple positive result shows that, if all valuation functions are *additive*, then we can get away with a protocol that only allows agents to make deals regarding the reallocation of one item at a time. Unfortunately, this is the best result we can hope for along these lines: for no strictly larger class of valuation functions will a simple protocol of one-item-at-a-time deals still suffice to guarantee convergence [60].

Similar results are also available for other fairness and efficiency criteria, such as Pareto optimality [104], egalitarian social welfare [104], and envy-freeness [55]. Most of the work in the field is of a theoretical nature, but the convergence problem has also been studied using agent-based simulations [4, 50].

Some of the results in the literature are based on the same notion of myopic individual rationality used here and others rely on other *rationality assumptions*. In fact, there are two natural perspectives regarding this point. First, we might start by postulating reasonable assumptions regarding the rational behavior of individual agents and then explore what convergence results can be proven. Second, we might start with a convergence property we would like to be able to guarantee, and then *design* appropriate rationality assumptions that will allow us to prove the corresponding theorem. That is, we may think of a multiagent system as, first, a system of self-interested agents we cannot control (but about which we can make certain assumptions) or, second, as a system of agents the behavior of which we can design and program ourselves, as a tool for distributed problem solving. Which perspective is appropriate depends on the application at hand.

Finally, the distributed approach to multiagent resource allocation also gives rise to new questions regarding *computational complexity*. For instance, we might ask how hard it is to decide whether a given profile of valuation functions and a given initial allocation admit a path consisting only of individually rational deals involving the exchange of a single item each to a socially optimal allocation. Questions of this kind have been studied in depth by Dunne and colleagues [93, 94].

6 Conclusion

This chapter has been an introduction to classical social choice theory and an exposition of some of the most important research trends in computational social choice. We have argued in the beginning that social choice theory, the mathematical theory of collective decision making, has a natural role to play when we think about the foundations of multiagent systems. As we are concluding the

chapter, we would like to relativize this statement somewhat: it is true that many decision problems in multiagent systems are naturally modeled as problems of social choice, but it is also true that many of the problems that one is likely to encounter in practice will not fit the template provided by the classical formal frameworks introduced here exactly, or will have additional structure that can be exploited. More research is required to improve our understanding of best practices for adapting the elegant mathematical tools that computational social choice can provide to the problems encountered by practitioners developing real multiagent systems. We hope that readers of this chapter will feel well equipped to participate in this undertaking.

Let us conclude with a brief review of additional topics in computational social choice, which we have not been able to cover in depth here, as well as with a few pointers to further reading.

6.1 Additional Topics

In terms of social choice *settings*, we have covered preference aggregation, voting, and fair division. Another important area of social choice theory is *matching*, which addresses the problem of how to pair up the elements of two groups that have preferences over each other (e.g., men and women, doctors and hospitals, or kidney donors and patients). Matching theory is particularly notable for its many successful applications. An excellent introduction to the field is the seminal monograph by Roth and Sotomayor [184]. Matching can be seen as a special case of coalition formation where agents have preferences over the various possible partitions of the set of agents (see the chapter on “Computational Coalition Formation” in this book).

Preferences are not the only individual characteristics that the members of a group might want to aggregate. Other examples include beliefs and judgments. In both cases there exists a small but significant technical literature in which beliefs and judgments, respectively, are modeled as sets of formulas in propositional logic that need to be aggregated. The work of Konieczny and Pino Pérez [143] is a good starting point for reading about *belief merging*, and List and Puppe [156] survey the literature on *judgment aggregation*. While belief merging grew out of the literature on belief revision in artificial intelligence and computational logic and always had a computational flavor to it, judgment aggregation initially developed in the political philosophy and the philosophical logic literature, and computational aspects did not get investigated until very recently [105].

Throughout the chapter, we have occasionally alluded to connections to *mechanism design*, a topic at the interface of social choice and game theory (see also the chapter on “Mechanism Design and Auctions” in this book). On the mechanism

design side, there has been interest in designing voting rules that are *false-name-proof* [227], that is, robust to a single voter participating under multiple identities. While this is not an inherently computational topic, it is motivated by applications such as elections that take place on the Internet. The design of such rules has been studied both in general [62] and under single-peaked preferences [208]. Unfortunately, the results are rather negative here. To address this, other work has extended the model, for example by making it costly to obtain additional identities [212] or by using social network structure to identify “suspect” identities [77]. An overview of work on false-name-proofness is given by Conitzer and Yokoo [72]. Another exciting new direction in the intersection of computational social choice and mechanism design is that of *approximate mechanism design without money* [178], where the goal is to obtain formal approximation ratios under the constraint of strategyproofness, without using payments.

In terms of *techniques*, we have focused on the axiomatic method, on algorithms, and on computational complexity. We have also discussed the use of tools from knowledge representation (for the representation of preferences in combinatorial domains). A further important research trend in computational social choice has considered *communication requirements* in social choice. This includes topics such as the amount of information that voters have to supply before we can compute the winner of an election [69, 196], the most efficient form of storing an intermediate election result that will permit us to compute the winner once the remaining ballots have been received [59, 219], whether voters can jointly compute the outcome of a voting rule while preserving the privacy of their individual preferences [44], and the number of deals that agents have to forge before a socially optimal allocation of goods will be found [103].

Another technique we have not discussed concerns the use of tools developed in *automated reasoning* to verify properties of social choice mechanisms and to confirm or discover theorems within social choice theory. Examples in this line of work include the verification of proofs of classical theorems in social choice theory in higher-order theorem proofs [168], a fully automated proof of Arrow’s theorem for the special case of three alternatives [205], and the automated discovery of theorems pertaining to the problem of ranking sets of objects [123].

6.2 Further Reading

There are a number of excellent textbooks on classical social choice theory that are highly recommended for further reading. General texts include those by Austen-Smith and Banks [7] and by Gaertner [120]. Taylor [206] specifically focuses on the manipulation problem in voting. Moulin [165] covers not only preference aggregation and voting, but also the axiomatic foundations of welfare economics

(i.e., fair division) and cooperative game theory. Two highly recommended surveys are those of Plott [173] and Sen [200].

The area of computational social choice (or certain parts thereof) has been surveyed by several authors. Chevaleyre et al. [56] provide a broad overview of the field as a whole. The literature on using computational complexity as a barrier against manipulation in voting is reviewed by Faliszewski et al. [114] and Faliszewski and Procaccia [107]; Faliszewski et al. [110] also discuss the complexity of winner determination and control problems in depth. Chevaleyre et al. [58] give an introduction to social choice in combinatorial domains. The survey on multiagent resource allocation by Chevaleyre et al. [54] covers the basics of fair division and also discusses connections to other areas relevant to multiagent systems, particularly combinatorial auctions. Conitzer [64, 65] compares research directions across mechanism design, combinatorial auctions, and voting. Endriss [101] gives concise proofs of classical results such as Arrow’s theorem and the Gibbard-Satterthwaite theorem, and then discusses application of logic in social choice theory, e.g., in judgment aggregation and to model preferences in combinatorial domains. Rothe et al. [186] provide a book-length introduction to computational social choice (in German), covering topics in voting, judgment aggregation, and fair division, and focusing particularly on complexity questions. Finally, the short monograph of Rossi et al. [183] on preference handling includes extensive discussions of voting and matching from the point of view of computational social choice.

Acknowledgments

Felix Brandt gratefully acknowledges DFG grants BR 2312/6-1, BR 2312/7-1, BR 2312/9-1, and BR 2312/10-1, for support. Vincent Conitzer gratefully acknowledges NSF awards IIS-0812113, IIS-0953756, and CCF-1101659, as well as an Alfred P. Sloan fellowship, for support. The work of Ulle Endriss has been partially supported by an NWO Vidi grant (639.022.706). All three authors furthermore acknowledge the support of the ESF EUROCORES project “Computational Foundations of Social Choice.”

7 Exercises

1. **Level 1** A utility function $u : U \rightarrow \mathbb{R}$ is said to *represent* a preference relation on U if, for all $a, b \in U$, $u(a) \geq u(b)$ if and only if $a \succsim b$. Show that, when U is finite, a preference relation can be represented by a utility function if and only if it is transitive and complete.

2. An SWF f is *non-imposing* if for every preference relation \succsim there exists a profile $R = (\succsim_1, \dots, \succsim_n)$ such that $f(R) = \succsim$. The purpose of this exercise is to investigate what happens to Arrow's theorem when we replace the axiom of Pareto optimality by the axiom of non-imposition.
 - (a) **Level 1** Show that Pareto optimality is strictly stronger than non-imposition. That is, show that every Pareto optimal SWF is non-imposing and that there exists a non-imposing SWF that is not Pareto optimal.
 - (b) **Level 2** Show that Arrow's theorem ceases to hold when we replace Pareto optimality by non-imposition. That is, show that there exists a SWF that satisfies IIA and that is both non-imposing and non-dictatorial.
3. **Level 2** Prove that the four conditions in Theorem 6.2 are logically independent by providing, for each of the conditions, an SCF that violates this property but satisfies the other three.
4. **Level 2** Show that every Copeland winner lies in the uncovered set and hence reaches every other alternative on a majority rule path of length at most two (assuming an odd number of voters).
5. **Level 1** Consider the following preference profile for 100 voters (due to Michel Balinski).

	33	16	3	8	18	22
a	a	b	c	c	d	e
b	b	d	d	e	e	c
c	c	c	b	b	c	b
d	d	e	a	d	b	d
e	e	a	e	a	a	a

Determine the winners according to plurality, Borda's rule, Copeland's rule, STV, and plurality with runoff (which yields the winner of a pairwise comparison between the two alternatives with the highest plurality score).

6. **Level 2** Give a polynomial-time algorithm that, for a given preference profile, decides whether an alternative will win under *all* scoring rules.
7. **Level 3** A *Condorcet loser* is an alternative that loses against every other alternative in pairwise majority comparisons. Check which of the following voting rules may choose a Condorcet loser: Borda's rule, Nanson's rule, Young's rule, maximin. Prove your answers.

8. **Level 3** An SCF is *monotonic* if a winning alternative will still win after it has been raised in one or more of the individual preference orderings (leaving everything else unchanged). Check which of the SCFs and voting rules mentioned in this chapter satisfy monotonicity and which satisfy Pareto optimality. Prove your answers.
9. **Level 2** Assume there is an odd number of voters and consider a restricted domain of preferences that always admits a Condorcet winner. Show that the voting rule that always yields the Condorcet winner is strategyproof.
10. **Level 2** Assume there is an odd number of voters, and rank the alternatives by their Copeland scores. Prove that there are no cycles in the pairwise majority relation if and only if no two alternatives are tied in this Copeland ranking.
11. **Level 2** Recall the definition of single-peakedness. Similarly, a preference profile R is *single-caved* if for every $x, y, z \in U$, it holds that if $(x < y < z)$ or $(z < y < x)$, then $y \succ_i x$ implies $z \succ_i y$ for every $i \in N$. Prove or disprove the following statements.
 - (a) Every preference profile for two voters and three alternatives is single-peaked.
 - (b) Every preference profile for two voters and more than three alternatives is single-peaked.
 - (c) Every single-peaked preference profile is single-peaked with respect to the linear order given by the preferences of one of the voters.
 - (d) Plurality and Condorcet winners coincide for single-peaked preferences.
 - (e) Plurality and Condorcet winners coincide for single-caved preferences.
 - (f) Borda and Condorcet winners coincide for single-peaked preferences.
12. **Level 4** We have seen that any non-dictatorial voting rule can be manipulated when we want that rule to operate on all possible preference profiles. We have also seen that this problem can be avoided when we restrict the domain of possible profiles appropriately, e.g., to single-peaked profiles. What we have not discussed is the *frequency* of manipulability: how often will we encounter a profile in which a voter has an incentive to manipulate? One way of studying this problem is by means of simulations: generate a large number of profiles and check for which proportion of them the problem under consideration (here, manipulability) occurs. The standard

method for generating profiles is to make the *impartial culture assumption*, under which every logically possible preference order has the same probability of occurring. For instance, if there are 3 alternatives, then there are $3! = 6$ possible (strict) preference orders, so each preference order should have probability $\frac{1}{6}$ to be a given voter's preference.

- (a) Write a program to analyze the frequency of manipulability of some of the voting rules introduced in this chapter under the impartial culture assumption.
- (b) While it is considered a useful base line, the impartial culture assumption has also been severely criticized for being too simplistic. Indeed, real electorates, be it in politics or multiagent systems, are unlikely to be impartial cultures. Can you think of better methods for generating data to test the frequency of interesting phenomena in social choice theory?

A good starting point for further reading on generating data for studying the frequency of social choice phenomena is the book of Regenwetter et al. [181]. There has also been a significant amount of theoretical work on the frequency of manipulability [recent contributions include, e.g., 8, 177, 203, 218].

13. **Level 2** Show that for each of the following voting rules the manipulation problem (with a single manipulator) can be solved in polynomial time by providing a suitable algorithm: the plurality rule, Borda's rule, Copeland's rule. Argue why your algorithms are correct and analyze their run-time in terms of the number of voters and alternatives.
14. For some voting rules, it is possible to significantly reduce the amount of information that the voters need to communicate by having the communication take place in rounds. A natural example is the STV rule (also known as instant runoff voting). Instead of having each agent communicate an entire ranking of all the alternatives at the outset, we can simply have the agents communicate their first-ranked alternatives; based on that, we can determine which alternative gets eliminated first; then, the agents who had ranked that alternative first communicate their next-most-preferred alternative; etc. In effect, this is removing the "instant" from "instant runoff voting"!
 - (a) **Level 2** When there are n voters and m alternatives, how many bits of communication does this protocol require in the worst case? Hints:
 - If there are i alternatives left, how many bits does an agent need to communicate to indicate its most-preferred one among them?

- If there are i alternatives left and we remove the one with the fewest votes, what is an upper bound on how many agents need to indicate a new most-preferred alternative among the $i - 1$ remaining ones?
- (b) **Level 4** Using tools from communication complexity [144], a lower bound of $\Omega(n \log m)$ bits of information in the worst case has been shown to hold for *any* communication protocol for the STV rule [69]. This leaves a gap with the result from (a). Can you close the gap, either by giving a better protocol or a better lower bound?
15. **Level 2** A *weak Condorcet winner* is an alternative that wins or draws against any other alternative in pairwise contests. Just like a (normal) Condorcet winner, a weak Condorcet winner need not exist for all preference profiles. Unlike a Condorcet winner, when it does exist, a weak Condorcet winner need not be unique. In the context of voting in combinatorial domains, show that when voters model their preferences using the language of prioritized goals and each voter only specifies a single goal, then there must always be a weak Condorcet winner.
16. **Level 1** In the context of measuring the fairness and efficiency of allocations of goods, check which of the following statements are true. Give either a proof (in the affirmative case) or a counterexample (otherwise).
- (a) Any allocation with maximal utilitarian social welfare is Pareto optimal.
 - (b) No allocation can maximize both utilitarian and egalitarian social welfare.
 - (c) Any allocation that is optimal with respect to the leximin ordering is both Pareto optimal and maximizes egalitarian social welfare.
 - (d) The Nash SWO is zero independent.
 - (e) The Nash SWO is independent of the common utility pace.
 - (f) The egalitarian SWO respects the Pigou-Dalton transfer principle.
17. **Level 2** The *elitist* CUF is defined via $f_{el}(u_1, \dots, u_n) := \max\{u_i \mid i \in N\}$, i.e., social welfare is equated with the individual utility of the agent that is currently best off. This CUF clearly contradicts our intuitions about fairness, but it might be just the right efficiency measure for some applications, e.g., in a multiagent system where we only care about at least one agent achieving its goal. What is the computational complexity of (the decision variant of) the problem of finding an allocation of indivisible goods (without money) that maximizes elitist social welfare?

- (a) First state your answer (and your proof) with respect to the explicit form of representing valuation functions (where the size of the representation of a function is proportional to the number of bundles to which it assigns a non-zero value).
- (b) Then repeat the same exercise, this time assuming that valuation functions are expressed using the language of weighted goals (without restrictions to the types of formulas used). Hint: You might expect that the complexity will increase, because now the input will be represented more compactly (on the other hand, as discussed in Section 5.3, there was no such increase in complexity for the utilitarian CUF).

Note that both of these languages can express valuation functions that need not be monotonic (that is, simply giving all the items to one agent will usually not yield an allocation with maximal elitist social welfare).

18. **Level 4** Consider a fair division problem with an odd number of agents. Under the *median-rank dictator CUF* the social welfare of an allocation is equal to the utility of the middle-most agent: $f_{mrd}(u_1, \dots, u_n) := u_{i^*}$, where i^* is defined as the (not necessarily unique) agent for which $|\{i \in N \mid u_i \leq u_{i^*}\}| = |\{i \in N \mid u_i \geq u_{i^*}\}|$. This is an attractive form of measuring social welfare: it associates social welfare with the individual utility of a representative agent, while being less influenced by extreme outliers than, for instance, the utilitarian CUF. At the time of writing, most of the problems discussed in the section on fair division have not yet been investigated for the median-rank dictator CUF.
 - (a) What is the computational complexity of computing an allocation that is optimal under the median-rank dictator CUF? Consider this question for different forms of representing individual valuation functions, such as the explicit form, weighted propositional formulas, or the OR/XOR family of bidding languages used in combinatorial auctions.
 - (b) Design and implement an algorithm for computing an optimal allocation under the median-rank dictator CUF for a preference representation language of your choice. You may find it useful to consult the literature on efficient algorithms for the winner determination problem in combinatorial auctions to get ideas on how to approach this task.
 - (c) Can you devise a notion of rationality (replacing myopic individual rationality as defined in this chapter) so that distributed negotiation will guarantee convergence to an optimal allocation under the median-rank dictator CUF? Are there suitable domain restrictions (limiting the diversity of valuation functions that agents may hold) that will ensure

convergence even when negotiation is limited to structurally simple deals (such as deals involving at most two agents at a time)?

References

- [1] A. Ali and M. Meila. Experiments with Kemeny ranking: What works when? *Mathematical Social Sciences*, 2012. Forthcoming.
- [2] N. Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1): 137–142, 2006.
- [3] A. Altman and M. Tennenholtz. Axiomatic foundations for ranking systems. *Journal of Artificial Intelligence Research*, 31:473–495, 2008.
- [4] M. Andersson and T. Sandholm. Contract type sequencing for reallocative negotiation. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS-2000)*, pages 154–160. IEEE Computer Society Press, 2000.
- [5] K. J. Arrow. *Social Choice and Individual Values*. New Haven: Cowles Foundation, 1st edition, 1951. 2nd edition 1963.
- [6] K. J. Arrow. Rational choice functions and orderings. *Economica*, 26:121–127, 1959.
- [7] D. Austen-Smith and J. S. Banks. *Positive Political Theory I: Collective Preference*. University of Michigan Press, 2000.
- [8] E. Baharad and Z. Neeman. The asymptotic strategyproofness of scoring and Condorcet consistent rules. *Review of Economic Design*, 4:331–340, 2002.
- [9] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40. ACM Press, 2006.
- [10] S. Barberà. The manipulation of social choice mechanisms that do not leave “too much” to chance. *Econometrica*, 45(7):1573–1588, 1977.
- [11] S. Barberà. A note on group strategy-proof decision schemes. *Econometrica*, 47(3):637–640, 1979.
- [12] S. Barberà. Majority and positional voting in a probabilistic framework. *Review of Economic Studies*, 46(2):379–389, 1979.
- [13] S. Barberà. Strategy-proof social choice. In K. J. Arrow, A. K. Sen, and K. Suzumura, editors, *Handbook of Social Choice and Welfare*, volume 2, chapter 25, pages 731–832. Elsevier, 2010.

- [14] S. Barberà, A. Bogomolnaia, and H. van der Stel. Strategy-proof probabilistic rules for expected utility maximizers. *Mathematical Social Sciences*, 35(2):89–103, 1998.
- [15] J. P. Barthélemy and B. Monjardet. The median procedure in cluster analysis and social choice theory. *Mathematical Social Sciences*, 1:235–267, 1981.
- [16] J. Bartholdi, III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modelling*, 16(8-9):27–40, 1992.
- [17] J. Bartholdi, III and J. B. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.
- [18] J. Bartholdi, III and M. Trick. Stable matching with preferences derived from a psychological model. *Operations Research Letters*, 5(4):165–169, 1986.
- [19] J. Bartholdi, III, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.
- [20] J. Bartholdi, III, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(3):157–165, 1989.
- [21] D. Baumeister and J. Rothe. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Information Processing Letters*, 2012. Forthcoming.
- [22] N. Betzler and B. Dorn. Towards a dichotomy for the possible winner problem in elections based on scoring rules. *Journal of Computer and System Sciences*, 76(8):812–836, 2010.
- [23] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny rankings. *Theoretical Computer Science*, 410(45):4554–4570, 2009.
- [24] N. Betzler, J. Guo, and R. Niedermeier. Parameterized complexity of Dodgson and Young elections. *Information and Computation*, 208(2):165–177, 2010.
- [25] N. Betzler, R. Niedermeier, and G. J. Woeginger. Unweighted coalitional manipulation under the Borda rule is NP-hard. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 55–60. AAAI Press, 2011.
- [26] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 2012. Forthcoming.

- [27] D. Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.
- [28] G. Bordes. Consistency, rationality and collective choice. *Review of Economic Studies*, 43(3):451–457, 1976.
- [29] G. Bordes and N. Tideman. Independence of irrelevant alternatives in the theory of voting. *Theory and Decision*, 30(2):163–186, 1991.
- [30] C. Boutilier and H. H. Hoos. Bidding languages for combinatorial auctions. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1211–1217. Morgan Kaufmann, 2001.
- [31] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Pool. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.
- [32] S. Bouveret. *Allocation et partage équitables de ressources indivisibles: modélisation, complexité et algorithmique*. PhD thesis, Supaéro/University of Toulouse, 2007.
- [33] S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. *Journal of Artificial Intelligence Research*, 32(1):525–564, 2008.
- [34] S. Bouveret, U. Endriss, and J. Lang. Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009)*, pages 67–72. AAAI Press, 2009.
- [35] S. Brams, D. Kilgour, and W. Zwicker. The paradox of multiple elections. *Social Choice and Welfare*, 15(2):211–236, 1998.
- [36] S. J. Brams and P. C. Fishburn. *Approval Voting*. Springer-Verlag, 2nd edition, 2007.
- [37] S. J. Brams and A. D. Taylor. *Fair Division: From Cake-cutting to Dispute Resolution*. Cambridge University Press, 1996.
- [38] F. Brandt. Some remarks on Dodgson’s voting rule. *Mathematical Logic Quarterly*, 55(4):460–463, 2009.
- [39] F. Brandt. Group-strategyproof irresolute social choice functions. In T. Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 79–84. AAAI Press, 2011.

- [40] F. Brandt. Minimal stable sets in tournaments. *Journal of Economic Theory*, 146(4):1481–1499, 2011.
- [41] F. Brandt and M. Brill. Necessary and sufficient conditions for the strategyproofness of irresolute social choice functions. In K. Apt, editor, *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 136–142. ACM Press, 2011.
- [42] F. Brandt and F. Fischer. Computing the minimal covering set. *Mathematical Social Sciences*, 56(2):254–268, 2008.
- [43] F. Brandt and P. Harrenstein. Set-rationalizable choice and self-stability. *Journal of Economic Theory*, 146(4):1721–1731, 2011.
- [44] F. Brandt and T. Sandholm. Unconditional privacy in social choice. In R. van der Meyden, editor, *Proceedings of the 10th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 207–218. ACM Press, 2005.
- [45] F. Brandt, F. Fischer, and P. Harrenstein. The computational complexity of choice sets. *Mathematical Logic Quarterly*, 55(4):444–459, 2009.
- [46] F. Brandt, M. Brill, E. Hemaspaandra, and L. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. In M. Fox and D. Poole, editors, *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 715–722. AAAI Press, 2010.
- [47] F. Brandt, F. Fischer, P. Harrenstein, and M. Mair. A computational analysis of the tournament equilibrium set. *Social Choice and Welfare*, 34(4):597–609, 2010.
- [48] F. Brandt, M. Brill, and H. G. Seedig. On the fixed-parameter tractability of composition-consistent tournament solutions. In T. Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 85–90. AAAI Press, 2011.
- [49] F. Brandt, M. Chudnovsky, I. Kim, G. Liu, S. Norin, A. Scott, P. Seymour, and S. Thomassé. A counterexample to a conjecture of Schwartz. *Social Choice and Welfare*, 2012. Forthcoming.
- [50] H. Buisman, G. Kruitbosch, N. Peek, and U. Endriss. Simulation of negotiation policies in distributed multiagent resource allocation. In *Engineering Societies in the Agents World VIII*, volume 4995 of *LNAI*, pages 224–239. Springer-Verlag, 2008.
- [51] I. Caragiannis, J. A. Covey, M. Feldman, C. M. Homan, C. Kaklamanis, N. Karanikolas, A. D. Procaccia, and J. S. Rosenschein. On the approximability of Dodgson and Young elections. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1058–1067. ACM Press, 2009.

- [52] I. Caragiannis, C. Kaklamanis, N. Karanikolas, and A. D. Procaccia. Socially desirable approximations for Dodgson's voting rule. *ACM Transactions on Algorithms*, 2012. Forthcoming.
- [53] I. Charon and O. Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR*, 5(1):5–60, 2007.
- [54] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.
- [55] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Reaching envy-free states in distributed negotiation settings. In M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, pages 1239–1244. AAAI Press, 2007.
- [56] Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. A short introduction to computational social choice. In *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 4362 of *Lecture Notes in Computer Science (LNCS)*, pages 51–69. Springer-Verlag, 2007.
- [57] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Multiagent resource allocation in k -additive domains: Preference representation and complexity. *Annals of Operations Research*, 163(1):49–62, 2008.
- [58] Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Preference handling in combinatorial domains: From AI to social choice. *AI Magazine*, 29(4):37–46, 2008.
- [59] Y. Chevaleyre, J. Lang, N. Maudet, and G. Ravilly-Abadie. Compiling the votes of a subelectorate. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 97–102. AAAI Press, 2009.
- [60] Y. Chevaleyre, U. Endriss, and N. Maudet. Simple negotiation schemes for agents with simple preferences: Sufficiency, necessity and maximality. *Journal of Autonomous Agents and Multiagent Systems*, 20(2):234–259, 2010.
- [61] V. Conitzer. Computing Slater rankings using similarities among candidates. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 613–619. AAAI Press, 2006.
- [62] V. Conitzer. Anonymity-proof voting rules. In *Proceedings of the Fourth Workshop on Internet and Network Economics (WINE)*, volume 5385 of *Lecture Notes in Computer Science (LNCS)*, pages 295–306. Springer-Verlag, 2008.
- [63] V. Conitzer. Eliciting single-peaked preferences using comparison queries. *Journal of Artificial Intelligence Research*, 35:161–191, 2009.

- [64] V. Conitzer. Making decisions based on the preferences of multiple agents. *Communications of the ACM*, 53(3):84–94, 2010.
- [65] V. Conitzer. Comparing multiagent systems research in combinatorial auctions and voting. *Annals of Mathematics and Artificial Intelligence*, 58(3):239–259, 2010.
- [66] V. Conitzer. Should social network structure be taken into account in elections? *Mathematical Social Sciences*, 2012. Forthcoming.
- [67] V. Conitzer and T. Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 392–397. AAAI Press, 2002.
- [68] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 781–788, 2003.
- [69] V. Conitzer and T. Sandholm. Communication complexity of common voting rules. In *Proceedings of the 6th ACM Conference on Electronic Commerce (ACM-EC)*, pages 78–87. ACM Press, 2005.
- [70] V. Conitzer and T. Sandholm. Common voting rules as maximum likelihood estimators. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 145–152, 2005.
- [71] V. Conitzer and T. Sandholm. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 627–634. AAAI Press, 2006.
- [72] V. Conitzer and M. Yokoo. Using mechanism design to prevent false-name manipulations. *AI Magazine*, 31(4):65–77, 2010. Special Issue on Algorithmic Game Theory.
- [73] V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 620–627. AAAI Press, 2006.
- [74] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3), 2007.
- [75] V. Conitzer, J. Lang, and L. Xia. How hard is it to control sequential elections via the agenda? In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 103–108. AAAI Press, 2009.
- [76] V. Conitzer, M. Rognlie, and L. Xia. Preference functions that score rankings and maximum likelihood estimation. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 109–115. AAAI Press, 2009.

- [77] V. Conitzer, N. Immorlica, J. Letchford, K. Munagala, and L. Wagman. False-name-proofness in social networks. In *Proceedings of the Sixth Workshop on Internet and Network Economics (WINE)*, volume 6484 of *Lecture Notes in Computer Science (LNCS)*, pages 209–221. Springer-Verlag, 2010.
- [78] V. Conitzer, J. Lang, and L. Xia. Hypercubewise preference aggregation in multi-issue domains. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 158–163. AAAI Press, 2011.
- [79] V. Conitzer, T. Walsh, and L. Xia. Dominating manipulations in voting with partial information. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 638–643. AAAI Press, 2011.
- [80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.
- [81] S. Coste-Marquis, J. Lang, P. Liberatore, and P. Marquis. Expressive power and succinctness of propositional languages for preference representation. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR-2004)*, pages 203–212. AAAI Press, 2004.
- [82] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2006.
- [83] C. d’Aspremont and L. Gevers. Equity and the informational basis of collective choice. *Review of Economic Studies*, 44(2):199–209, 1977.
- [84] A. Davenport and J. Kalagnanam. A computational study of the Kemeny rule for preference aggregation. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, pages 697–702. AAAI Press, 2004.
- [85] J. Davies, G. Katsirelos, N. Narodytska, and T. Walsh. Complexity of and algorithms for Borda manipulation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 657–662. AAAI Press, 2011.
- [86] M. de Condorcet. *Essai sur l’application de l’analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie Royale, 1785. Facsimile published in 1972 by Chelsea Publishing Company, New York.
- [87] B. de Keijzer, S. Bouveret, T. Klos, and Y. Zhang. On the complexity of efficiency and envy-freeness in fair division of indivisible goods with additive preferences. In *Proceedings of the 1st International Conference on Algorithmic Decision Theory (ADT)*, pages 98–110. Springer-Verlag, 2009.
- [88] R. Deb. On Schwartz’s rule. *Journal of Economic Theory*, 16:103–110, 1977.

- [89] Y. Desmedt and E. Elkind. Equilibria of plurality voting with abstentions. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 347–356. ACM Press, 2010.
- [90] M. Drissi-Bakhkhat and M. Truchon. Maximum likelihood approach to vote aggregation with variable probabilities. *Social Choice and Welfare*, 23:161–185, 2004.
- [91] L. E. Dubins and E. H. Spanier. How to cut a cake fairly. *American Mathematical Monthly*, 68(1):1–17, 1961.
- [92] J. Duggan and T. Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard-Satterthwaite generalized. *Social Choice and Welfare*, 17(1):85–93, 2000.
- [93] P. E. Dunne and Y. Chevaleyre. The complexity of deciding reachability properties of distributed negotiation schemes. *Theoretical Computer Science*, 396(1–3):113–144, 2008.
- [94] P. E. Dunne, M. Wooldridge, and M. Laurence. The complexity of contract negotiation. *Artificial Intelligence*, 164(1–2):23–46, 2005.
- [95] B. Dutta. Covering sets and a new Condorcet choice correspondence. *Journal of Economic Theory*, 44:63–80, 1988.
- [96] B. Dutta, H. Peters, and A. Sen. Strategy-proof cardinal decision schemes. *Social Choice and Welfare*, 28(1):163–179, 2007.
- [97] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, pages 613–622. ACM Press, 2001.
- [98] E. Elkind and H. Lipmaa. Hybrid voting protocols and hardness of manipulation. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science (LNCS)*, pages 206–215. Springer-Verlag, 2005.
- [99] E. Elkind, P. Faliszewski, and A. Slinko. On the role of distances in defining voting rules. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 375–382. IFAAMAS, 2010.
- [100] E. Elkind, P. Faliszewski, and A. Slinko. Rationalizations of Condorcet-consistent rules via distances of hamming type. *Social Choice and Welfare*, 2012. Forthcoming.
- [101] U. Endriss. Logic and social choice theory. In A. Gupta and J. van Benthem, editors, *Logic and Philosophy Today*, volume 2, pages 333–377. College Publications, 2011.

- [102] U. Endriss and J. Lang, editors. *Proceedings of the 1st International Workshop on Computational Social Choice (COMSOC-2006)*, 2006. ILLC, University of Amsterdam.
- [103] U. Endriss and N. Maudet. On the communication complexity of multilateral trading: Extended report. *Journal of Autonomous Agents and Multiagent Systems*, 11 (1):91–107, 2005.
- [104] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Negotiating socially optimal allocations of resources. *Journal of Artificial Intelligence Research*, 25:315–348, 2006.
- [105] U. Endriss, U. Grandi, and D. Porello. Complexity of judgment aggregation: Safety of the agenda. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*. IFAAMAS, 2010.
- [106] B. Escoffier, J. Lang, and M. Öztürk. Single-peaked consistency and its complexity. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pages 366–370. IOS Press, 2008.
- [107] P. Faliszewski and A. D. Procaccia. AI’s war on manipulation: Are we winning? *AI Magazine*, 31(4):53–64, 2010.
- [108] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. The complexity of bribery in elections. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 641–646. AAAI Press, 2006.
- [109] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: Ties matter. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 983–990. IFAAMAS, 2008.
- [110] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. In S. Ravi and S. Shukla, editors, *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*. Springer-Verlag, 2009.
- [111] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. In *Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 118–127. ACM Press, 2009.
- [112] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.
- [113] P. Faliszewski, E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research*, 35:275–341, 2009.

- [114] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Using complexity to protect elections. *Communications of the ACM*, 53(11):74–82, 2010.
- [115] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Manipulation of Copeland elections. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 367–374. IFAAMAS, 2010.
- [116] J. Farfel and V. Conitzer. Aggregating value ranges: Preference elicitation and truthfulness. *Journal of Autonomous Agents and Multiagent Systems*, 22(1):127–150, 2011. Special Issue on Computational Social Choice.
- [117] M. Fey. Choosing from a large tournament. *Social Choice and Welfare*, 31(2): 301–309, 2008.
- [118] P. C. Fishburn. *The Theory of Social Choice*. Princeton University Press, 1973.
- [119] E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *Proceedings of the 49th Symposium on Foundations of Computer Science (FOCS)*, pages 243–249. IEEE Computer Society Press, 2008.
- [120] W. Gaertner. *A Primer in Social Choice Theory*. LSE Perspectives in Economic Analysis. Oxford University Press, 2006.
- [121] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [122] S. Gaspers and M. Mnich. Feedback vertex sets in tournaments. In *Proceedings of the 18th European Symposium on Algorithms (ESA)*, volume 6346 of *Lecture Notes in Computer Science (LNCS)*, pages 267–277. Springer-Verlag, 2010.
- [123] C. Geist and U. Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research*, 40:143–174, 2011.
- [124] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.
- [125] A. Gibbard. Manipulation of schemes that mix voting with chance. *Econometrica*, 45(3):665–681, 1977.
- [126] A. Gibbard. Straightforwardness of game forms with lotteries as outcomes. *Econometrica*, 46(3):595–614, 1978.
- [127] M. Grabisch. k -order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92:167–189, 1997.
- [128] R. J. Gretlein. Dominance elimination procedures on finite alternative games. *International Journal of Game Theory*, 12(2):107–113, 1983.

- [129] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. *Journal of Computer and System Sciences*, 73(1):73–83, 2007.
- [130] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll’s 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.
- [131] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349:382–391, 2005.
- [132] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *AI Journal*, 171(5–6):255–285, 2007.
- [133] O. Hudry. A note on “Banks winners in tournaments are difficult to recognize” by G. J. Woeginger. *Social Choice and Welfare*, 23:113–114, 2004.
- [134] O. Hudry. A survey on the complexity of tournament solutions. *Mathematical Social Sciences*, 57(3):292–303, 2009.
- [135] O. Hudry. On the complexity of Slater’s problems. *European Journal of Operational Research*, 203(1):216–221, 2010.
- [136] O. Hudry. On the computation of median linear orders, of median complete pre-orders and of median weak orders. *Mathematical Social Sciences*, 2012. Forthcoming.
- [137] A. Hylland. Strategyproofness of voting procedures with lotteries as outcomes and infinite sets of strategies. Mimeo, 1980.
- [138] S. Ieong and Y. Shoham. Marginal contribution nets: A compact representation scheme for coalitional games. In *Proceedings of the 6th ACM Conference on Electronic Commerce (ACM-EC)*, pages 193–202. ACM Press, 2005.
- [139] M. Isaksson, G. Kindler, and E. Mossel. The geometry of manipulation: A quantitative proof of the Gibbard-Satterthwaite theorem. In *Proceedings of the 51st Symposium on Foundations of Computer Science (FOCS)*, pages 319–328. IEEE Computer Society Press, 2010.
- [140] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:577–591, 1959.
- [141] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors: A PTAS for weighted feedback arc set on tournaments. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 95–103. ACM Press, 2007.
- [142] K. Konczak and J. Lang. Voting procedures with incomplete preferences. In *Proceedings of the 1st Multidisciplinary Workshop on Advances in Preference Handling*, 2005.

- [143] S. Konieczny and S. Pino Pérez. Logic based merging. *Journal of Philosophical Logic*, 40(2):239–270, 2011.
- [144] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- [145] D. Lacy and E. M. S. Niou. A problem with referendums. *Journal of Theoretical Politics*, 12(1):5–31, 2000.
- [146] C. Lafage and J. Lang. Logical representation of preferences for group decision making. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR-2000)*, pages 457–468. Morgan Kaufmann Publishers, 2000.
- [147] G. Laffond, J.-F. Laslier, and M. Le Breton. The bipartisan set of a tournament game. *Games and Economic Behavior*, 5:182–201, 1993.
- [148] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):37–71, 2004.
- [149] J. Lang and L. Xia. Sequential composition of voting rules in multi-issue domains. *Mathematical Social Sciences*, 57(3):304–324, 2009.
- [150] J. Lang, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Winner determination in sequential majority voting. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1372–1377. AAAI Press, 2007.
- [151] J.-F. Laslier. *Tournament Solutions and Majority Voting*. Springer-Verlag, 1997.
- [152] J.-F. Laslier and M. R. Sanver, editors. *Handbook on Approval Voting*. Studies in Choice and Welfare. Springer-Verlag, 2010.
- [153] M. Li, Q. B. Vo, and R. Kowalczyk. Majority-rule-based preference aggregation on multi-attribute domains with CP-nets. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 659–666. IFAAMAS, 2011.
- [154] C. Lindner and J. Rothe. Degrees of guaranteed envy-freeness in finite bounded cake-cutting protocols. In *Proceedings of the 5th International Workshop on Internet and Network Economics (WINE)*, volume 5929 of *Lecture Notes in Computer Science (LNCS)*, pages 149–159. Springer-Verlag, 2009.
- [155] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce (ACM-EC)*, pages 125–131. ACM Press, 2004.

- [156] C. List and C. Puppe. Judgment aggregation: A survey. In *Handbook of Rational and Social Choice*. Oxford University Press, 2009.
- [157] K. May. A set of independent, necessary and sufficient conditions for simple majority decisions. *Econometrica*, 20:680–684, 1952.
- [158] J. McCabe-Dansted, G. Pritchard, and A. Slinko. Approximability of Dodgson’s rule. *Social Choice and Welfare*, 31(2):311–330, 2008.
- [159] R. D. McKelvey and R. G. Niemi. A multistage game representation of sophisticated voting for binary procedures. *Journal of Economic Theory*, 18(1):1–22, 1978.
- [160] I. McLean and A. B. Urken, editors. *Classics of Social Choice*. University of Michigan Press, Ann Arbor, 1995.
- [161] E. Meskanen and H. Nurmi. Closeness counts in social choice. In M. Braham and F. Steffen, editors, *Power, Freedom, and Voting*. Springer-Verlag, 2008.
- [162] H. Moulin. Dominance solvable voting schemes. *Econometrica*, 47(6):1137–1151, 1979.
- [163] H. Moulin. *The Strategy of Social Choice*. North-Holland, 1983.
- [164] H. Moulin. Choosing from a tournament. *Social Choice and Welfare*, 3:271–291, 1986.
- [165] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.
- [166] N. Narodytska, T. Walsh, and L. Xia. Manipulation of Nanson’s and Baldwin’s rule. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 713–718. AAAI Press, 2011.
- [167] E. M. S. Niou. A note on Nanson’s rule. *Public Choice*, 54:191–193, 1987.
- [168] T. Nipkow. Social choice theory in HOL: Arrow and Gibbard-Satterthwaite. *Journal of Automated Reasoning*, 43(3):289–304, 2009.
- [169] N. Nisan. Bidding languages for combinatorial auctions. In *Combinatorial Auctions*, chapter 9, pages 215–232. MIT Press, 2006.
- [170] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical Report 1999–66, Stanford University, 1999.

- [171] D. M. Pennock, E. Horvitz, and C. L. Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 729–734. AAAI Press, 2000.
- [172] G. Pinkas. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. *Artificial Intelligence*, 77(2):203–247, 1995.
- [173] C. R. Plott. Axiomatic social choice theory: An overview and interpretation. *American Journal of Political Science*, 20(3):511–596, 1976.
- [174] D. Porello and U. Endriss. Ontology merging as social choice. In *Proceedings of the 12th International Workshop on Computational Logic in Multiagent Systems (CLIMA-2011)*, volume 6814 of *LNAI*, pages 157–170. Springer-Verlag, 2011.
- [175] A. Procaccia. Can approximation circumvent Gibbard-Satterthwaite? In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 836–841. AAAI Press, 2010.
- [176] A. D. Procaccia and J. S. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research*, 28:157–181, 2007.
- [177] A. D. Procaccia and J. S. Rosenschein. Average-case tractability of manipulation in voting via the fraction of manipulators. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 718–720. AAAI Press, 2007.
- [178] A. D. Procaccia and M. Tennenholtz. Approximate mechanism design without money. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 177–186, Stanford, CA, USA, 2009.
- [179] A. D. Procaccia, J. S. Rosenschein, and A. Zohar. Multi-winner elections: Complexity of manipulation, control and winner-determination. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1476–1481. AAAI Press, 2007.
- [180] S. Ramezani and U. Endriss. Nash social welfare in multiagent resource allocation. In *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, volume 59 of *Lecture Notes in Business Information Processing*, pages 117–131. Springer-Verlag, 2010.
- [181] M. Regenwetter, B. Grofman, A. A. J. Marley, and I. M. Tsetlin. *Behavioral Social Choice: Probabilistic Models, Statistical Inference, and Applications*. Cambridge University Press, 2006.
- [182] J. Robertson and W. Webb. *Cake-Cutting Algorithms*. A. K. Peters, 1998.

- [183] F. Rossi, K. B. Venable, and T. Walsh. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan & Claypool Publishers, 2011.
- [184] A. Roth and M. A. O. Sotomayor. *Two-Sided Matching: A Study in Game Theoretic Modelling and Analysis*. Cambridge University Press, 1990.
- [185] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems*, 36(4):375–386, 2003.
- [186] J. Rothe, D. Baumeister, C. Lindner, and I. Rothe. *Einführung in Computational Social Choice*. Spektrum Akademischer Verlag, 2011.
- [187] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
- [188] P. A. Samuelson. Arrow’s mathematical politics. In S. Hook, editor, *Human Values and Economic Policy*. New York University Press, 1967.
- [189] T. Sandholm. Contract types for satisficing task allocation: I Theoretical results. In *Proceedings of the AAAI Spring Symposium on Satisficing Models*, 1998.
- [190] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
- [191] M. A. Satterthwaite. Strategy-proofness and Arrow’s conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.
- [192] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011.
- [193] T. Schwartz. *The Logic of Collective Choice*. Columbia University Press, 1986.
- [194] T. Schwartz. Cyclic tournaments and cooperative majority voting: A solution. *Social Choice and Welfare*, 7:19–29, 1990.
- [195] A. Scott and M. Fey. The minimal covering set in large tournaments. *Social Choice and Welfare*, 38(1):1–9, 2012.
- [196] I. Segal. The communication requirements of social choice rules and supporting budget sets. *Journal of Economic Theory*, 136:341–378, 2007.
- [197] A. K. Sen. A possibility theorem on majority decisions. *Econometrica*, 34(2):491–499, 1966.

- [198] A. K. Sen. Quasi-transitivity, rational choice and collective decision. *Review of Economic Studies*, 36(3):381–393, 1969.
- [199] A. K. Sen. Social choice theory: A re-examination. *Econometrica*, 45(1):53–89, 1977.
- [200] A. K. Sen. Social choice theory. In K. J. Arrow and M. D. Intriligator, editors, *Handbook of Mathematical Economics*, volume 3, chapter 22, pages 1073–1181. Elsevier, 1986.
- [201] A. K. Sen and P. K. Pattanaik. Necessary and sufficient conditions for rational choice under majority decision. *Journal of Economic Theory*, 1:178–202, 1969.
- [202] K. A. Shepsle and B. R. Weingast. Uncovered sets and sophisticated outcomes with implications for agenda institutions. *American Journal of Political Science*, 28(1):49–74, 1984.
- [203] A. Slinko. How large should a coalition be to manipulate an election? *Mathematical Social Sciences*, 47(3):289–293, 2004.
- [204] J. H. Smith. Aggregation of preferences with variable electorate. *Econometrica*, 41(6):1027–1041, 1973.
- [205] P. Tang and F. Lin. Computer-aided proofs of Arrow’s and other impossibility theorems. *Artificial Intelligence*, 173(11):1041–1053, 2009.
- [206] A. D. Taylor. *Social Choice and the Mathematics of Manipulation*. Cambridge University Press, 2005.
- [207] T. N. Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.
- [208] T. Todo, A. Iwasaki, and M. Yokoo. False-name-proof mechanism design without money. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 651–658. IFAAMAS, 2011.
- [209] M. Truchon. Borda and the maximum likelihood approach to vote aggregation. *Mathematical Social Sciences*, 55(1):96–102, 2008.
- [210] J. Uckelman and U. Endriss. Compactly representing utility functions using weighted goals and the max aggregator. *Artificial Intelligence*, 174(15):1222–1246, 2010.
- [211] J. Uckelman, Y. Chevaleyre, U. Endriss, and J. Lang. Representing utility functions via weighted goals. *Mathematical Logic Quarterly*, 55(4):341–361, 2009.

- [212] L. Wagman and V. Conitzer. Optimal false-name-proof voting rules with costly voting. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 190–195. AAAI Press, 2008.
- [213] T. Walsh. Uncertainty in preference elicitation and aggregation. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 3–8. AAAI Press, 2007.
- [214] T. Walsh. Where are the really hard manipulation problems? The phase transition in manipulating the veto rule. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 324–329. AAAI Press, 2009.
- [215] R. B. Wilson. Social choice theory without the Pareto principle. *Journal of Economic Theory*, 5:478–486, 1972.
- [216] G. J. Woeginger. Banks winners in tournaments are difficult to recognize. *Social Choice and Welfare*, 20:523–528, 2003.
- [217] L. Xia and V. Conitzer. A sufficient condition for voting rules to be frequently manipulable. In *Proceedings of the 9th ACM Conference on Electronic Commerce (ACM-EC)*, pages 99–108. ACM Press, 2008.
- [218] L. Xia and V. Conitzer. Generalized scoring rules and the frequency of coalitional manipulability. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 109–118. ACM Press, 2008.
- [219] L. Xia and V. Conitzer. Compilation complexity of common voting rules. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 915–920. AAAI Press, 2010.
- [220] L. Xia and V. Conitzer. Stackelberg voting games: Computational aspects and paradoxes. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 921–926. AAAI Press, 2010.
- [221] L. Xia and V. Conitzer. Determining possible and necessary winners under common voting rules given partial orders. *Journal of Artificial Intelligence Research*, 41:25–67, 2011.
- [222] L. Xia and V. Conitzer. A maximum likelihood approach towards aggregating partial orders. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 446–451. AAAI Press, 2011.
- [223] L. Xia, V. Conitzer, and J. Lang. Voting on multiattribute domains with cyclic preferential dependencies. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 202–207. AAAI Press, 2008.

- [224] L. Xia, M. Zuckerman, A. D. Procaccia, V. Conitzer, and J. S. Rosenschein. Complexity of unweighted coalitional manipulation under some common voting rules. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 348–353. AAAI Press, 2009.
- [225] L. Xia, V. Conitzer, and J. Lang. Aggregating preferences in multi-issue domains by using maximum likelihood estimators. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 399–406. AAAI Press, 2010.
- [226] L. Xia, V. Conitzer, and J. Lang. Strategic sequential voting in multi-issue domains and multiple-election paradoxes. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 179–188. ACM Press, 2011.
- [227] M. Yokoo, Y. Sakurai, and S. Matsubara. The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. *Games and Economic Behavior*, 46(1):174–188, 2004.
- [228] H. P. Young. An axiomatization of Borda’s rule. *Journal of Economic Theory*, 9: 43–52, 1974.
- [229] H. P. Young. Social choice scoring functions. *SIAM Journal on Applied Mathematics*, 28(4):824–838, 1975.
- [230] H. P. Young. Extending Condorcet’s rule. *Journal of Economic Theory*, 16:335–353, 1977.
- [231] H. P. Young. Condorcet’s theory of voting. *The American Political Science Review*, 82(4):1231–1244, 1988.
- [232] H. P. Young. Optimal voting rules. *Journal of Economic Perspectives*, 9(1):51–64, 1995.
- [233] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Chapter 7

Mechanism Design and Auctions¹

Kevin Leyton-Brown and Yoav Shoham

1 Introduction

Mechanism design is a strategic version of social choice theory, which adds the assumption that agents will behave so as to maximize their individual payoffs. For example, in an election agents may not vote their true preference. Like social choice theory, however, the scope of mechanism design is broader than voting. The most famous application of mechanism design is *auction theory*, to which we devote the second part of this chapter. However, mechanism design has many other applications.

Consider the transportation network described in Figure 7.1. The number next to a given edge is the cost of transporting along that edge, but these costs are the private information of the various shippers that own each edge. The task here is to find the shortest (least-cost) path from S to T ; this is hard because the shippers may lie about their costs. Your one advantage is that you know that they are interested in maximizing their revenue. How can you use that knowledge to extract from them the information needed to compute the desired path?

This is where *mechanism design*, or *implementation theory*, comes in. Mechanism design is sometimes colloquially called “inverse game theory.” The problem most conventionally addressed by game theory can be framed as follows: given

¹This chapter is distilled from Chapters 10 and 11 of *Multiagent Systems: Algorithmic, Game-Theoretic and Logical Foundations*, published by Cambridge University Press [29]. This material is reprinted with the permission of its original publisher.

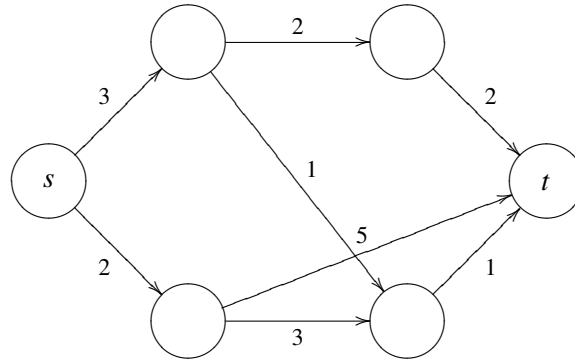


Figure 7.1: Transportation network with selfish agents.

an interaction among a set of agents, how do we predict or prescribe the course of action of the various agents participating in the interaction? In mechanism design, rather than investigate a given strategic interaction, we start with certain desired behaviors on the part of agents and ask what strategic interaction among these agents might give rise to these behaviors. Roughly speaking, from the technical point of view this will translate to the following. We will assume unknown individual preferences, and ask whether we can design a game such that, no matter what the secret preferences of the agents actually are, the equilibrium of the game is guaranteed to have a certain desired property or set of properties. Mechanism design is perhaps the most “computer scientific” part of game theory, since it concerns itself with designing effective protocols for distributed systems. The key difference from the traditional work in distributed systems is that in the current setting the distributed elements are not necessarily cooperative, and must be motivated to play their part. For this reason one can think of mechanism design as an exercise in “incentive engineering.”

2 Mechanism Design with Unrestricted Preferences

We begin by introducing some of the broad principles of mechanism design, placing no restriction on the preferences agents can have. (We will consider such restrictions in later sections.) Because mechanism design is most often studied in settings where agents’ preferences are unknown, we start by defining a Bayesian game setting.

Definition 7.1 (Bayesian game setting) A Bayesian game setting is a tuple (N, O, Θ, p, u) , where

- N is a finite set of n agents;

- O is a set of outcomes;
- $\Theta = \Theta_1 \times \cdots \times \Theta_n$ is a set of possible joint type vectors;
- p is a (common-prior) probability distribution on Θ ; and
- $u = (u_1, \dots, u_n)$, where $u_i : O \times \Theta \mapsto \mathbb{R}$ is the utility function for each player i .

Given a Bayesian game setting, we can define a mechanism.

Definition 7.2 (Mechanism) A mechanism (for a Bayesian game setting (N, O, Θ, p, u)) is a pair (A, M) , where

- $A = A_1 \times \cdots \times A_n$, where A_i is the set of actions available to agent $i \in N$; and
- $M : A \mapsto \Pi(O)$ maps each action profile to a distribution over outcomes.

A mechanism is *deterministic* if for every $a \in A$, there exists $o \in O$ such that $M(a)(o) = 1$; in this case we write simply $M(a) = o$.

2.1 Implementation

Together, a Bayesian game setting and a mechanism define a Bayesian game. The aim of mechanism design is to select a mechanism, given a particular Bayesian game setting, whose equilibria have desirable properties. We now define the most fundamental such property: that the outcomes that arise when the game is played are consistent with a given social choice function.

Definition 7.3 (Implementation in dominant strategies) Given a Bayesian game setting (N, O, Θ, p, u) , a mechanism (A, M) is an implementation in dominant strategies of a social choice function C (over N and O) if for any vector of utility functions u , the game has an equilibrium in dominant strategies, and in any such equilibrium a^* we have $M(a^*) = C(u)$.

A mechanism that gives rise to dominant strategies is sometimes called *strategyproof*, because there is no need for agents to reason about each others' actions in order to maximize their utility. This suggests that the above definition can be relaxed, and can appeal to solution concepts that are weaker than dominant-strategy equilibrium. For example, one can appeal to the Bayes–Nash equilibrium.

Definition 7.4 (Implementation in Bayes–Nash equilibrium) *Given a Bayesian game setting (N, O, Θ, p, u) , a mechanism (A, M) is an implementation in Bayes–Nash equilibrium of a social choice function C (over N and O) if there exists a Bayes–Nash equilibrium of the game of incomplete information (N, A, Θ, p, u) such that for every $\theta \in \Theta$ and every action profile $a \in A$ that can arise given type profile θ in this equilibrium, we have that $M(a) = C(u(\cdot, \theta))$.*

A classical example of Bayesian mechanism design is auction design. While we defer a lengthier discussion of auctions to Section 4.4, the basic idea is as follows. The designer wishes, for example, to ensure that the bidder with the highest valuation for a given item will win the auction, but the valuations of the agents are all private. The outcomes consist of allocating the item (in the case of a simple, single-item auction) to one of the agents, and having the agents make or receive some payments. The auction rules define the actions available to the agents (the “bidding rules”), and the mapping from action vectors to outcomes (“allocation rules” and “payment rules”: who wins and who pays what as a function of the bidding). If we assume that the valuations are drawn from some known distribution, each particular auction design and particular set of agents define a Bayesian game, in which the signal of each agent is its own valuation.

Finally, there exist implementation concepts that are satisfied by a larger set of strategy profiles than implementation in dominant strategies, but that are not guaranteed to be achievable for any given social choice function and set of preferences, unlike Bayes–Nash implementation. For example, we could consider only symmetric Bayes–Nash equilibria, on the principle that strategies that depend on agent identities would be less likely to arise in practice. It turns out that symmetric Bayes–Nash equilibria always exist in symmetric Bayesian games. A second implementation notion that deserves mention is *ex post* implementation. An *ex post* equilibrium has the property that no agent can ever gain by changing its strategy even if it observes the other agents’ types, as long as all the other agents follow the equilibrium strategies. Thus, unlike a Bayes–Nash equilibrium, an *ex post* equilibrium does not depend on the type distribution. Regardless of the implementation concept, we can require that the desired social choice function is implemented in the only equilibrium, in every equilibrium, or in at least one equilibrium of the underlying game.

2.2 The Revelation Principle

One property that is often desired of mechanisms is called *truthfulness*. This property holds when agents truthfully disclose their preferences to the mechanism in equilibrium. It turns out that this property can always be achieved regardless of the social choice function implemented and of the agents’ preferences. More for-

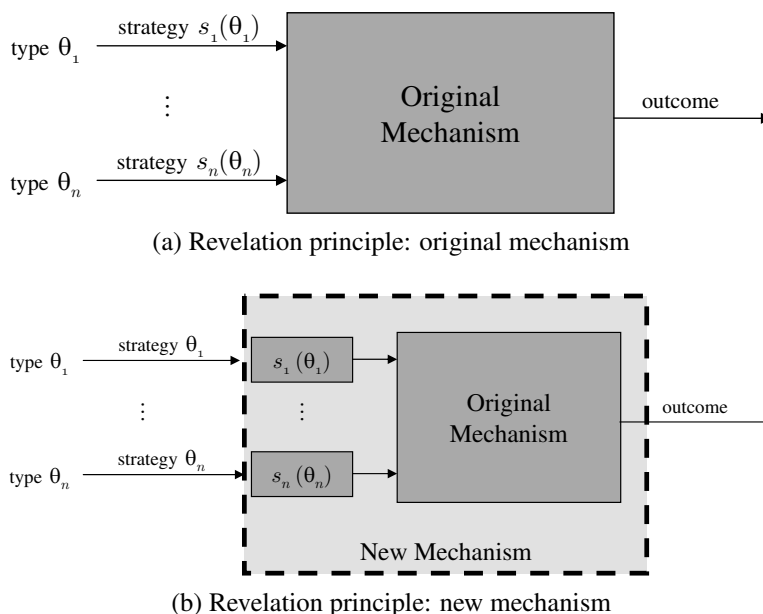


Figure 7.2: The revelation principle: how to construct a new mechanism with a truthful equilibrium, given an original mechanism with equilibrium (s_1, \dots, s_n) .

mally, a *direct mechanism* is one in which the only action available to each agent is to announce its private information. Since in a Bayesian game an agent's private information is its type, direct mechanisms have $A_i = \Theta_i$. When an agent's set of actions is the set of all its possible types, it may lie and announce a type $\hat{\theta}_i$ that is different from its true type θ_i . A direct mechanism is said to be *truthful* (or *incentive compatible*) if, for any type vector θ , in equilibrium of the game defined by the mechanism, every agent i 's strategy is to announce its true type, so that $\hat{\theta}_i = \theta_i$. We can thus speak about *incentive compatibility in dominant strategies* and *Bayes–Nash incentive compatibility*. Our claim that truthfulness can always be achieved implies, for example, that the social choice functions implementable by dominant-strategy truthful mechanisms are precisely those implementable by strategyproof direct mechanisms. This means that we can, without loss of coverage, limit ourselves to a small sliver of the space of all mechanisms.

Theorem 7.1 (Revelation principle) *If there exists any mechanism that implements a social choice function C in dominant strategies, then there exists a direct mechanism that implements C in dominant strategies and is truthful.*

Proof. Consider an arbitrary mechanism for n agents that implements a social choice function C in dominant strategies. This mechanism is illustrated in Figure 7.2a. Let s_1, \dots, s_n denote the dominant strategies for agents $1, \dots, n$. We will

construct a new mechanism that *truthfully* implements C . Our new mechanism will ask the agents for their utility functions, use them to determine s_1, \dots, s_n , the agents' dominant strategies under the original mechanism, and then choose the outcome that would have been chosen by the original mechanism for agents following the strategies s_1, \dots, s_n . This new mechanism is illustrated in Figure 7.2b.

Assume that some agent i would be better off declaring a utility function u_i^* to the new mechanism rather than its true utility function u_i . This implies that i would have preferred to follow some different strategy s_i^* in the original mechanism rather than s_i , contradicting our assumption that s_i is a dominant strategy for i . (Intuitively, if i could gain by lying to the new mechanism, it could likewise gain by “lying to itself” in the original mechanism.) Thus the new mechanism is dominant-strategy truthful. ■

In other words, any solution to a mechanism design problem can be converted into one in which agents always reveal their true preferences, if the new mechanism “lies for the agents” in just the way they would have chosen to lie to the original mechanism. The revelation principle is arguably the most basic result in mechanism design. It means that, while one might have thought *a priori* that a particular mechanism design problem calls for an arbitrarily complex strategy space, in fact one can restrict one's attention to truthful, direct mechanisms.

As we asserted earlier, the revelation principle does not apply only to implementation in dominant strategies; we have stated the theorem in this way only to keep things simple. Following exactly the same argument we can argue that, for example, a mechanism that implements a social choice function in a Bayes–Nash equilibrium can be converted into a direct, Bayes–Nash incentive-compatible mechanism.

The argument we used to justify the revelation principle also applies to original mechanisms that are indirect (e.g., ascending auctions). The new, direct mechanism can take the agents' utility functions, construct their strategies for the indirect mechanism, and then simulate the indirect mechanism to determine which outcome to select. One caveat is that, even if the original indirect mechanism had a unique equilibrium, there is no guarantee that the new revelation mechanism will not have additional equilibria.

Before moving on, we finally offer some computational caveats to the revelation principle. Observe that the general effect of constructing a revelation mechanism is to push an additional computational burden onto the mechanism, as is implicit in Figure 7.2b. There are many settings in which agents' equilibrium strategies are computationally difficult to determine. When this is the case, the additional burden absorbed by the mechanism may be considerable. Furthermore, the revelation mechanism forces the agents to reveal their types completely. There may be settings in which agents are not willing to compromise their privacy to this

degree. (Observe that the original mechanism may require them to reveal much less information.) Finally, even if not objectionable on privacy grounds, this full revelation can sometimes place an unreasonable burden on the communication channel. For all these reasons, in practical settings one must apply the revelation principle with caution.

2.3 Impossibility of General, Dominant-Strategy Implementation

We now ask what social choice functions can be implemented in dominant strategies. Given the revelation principle, we can restrict our attention to truthful mechanisms. The first answer is disappointing.

Theorem 7.2 (Gibbard–Satterthwaite) *Consider any social choice function C of N and O . If:*

1. $|O| \geq 3$ (there are at least three outcomes);
2. C is onto; that is, for every $o \in O$ there is a preference profile $[\succ]$ such that $C([\succ]) = o$ (this property is sometimes also called citizen sovereignty); and
3. C is dominant-strategy truthful,

then C is dictatorial.

Note that this negative result is specific to dominant-strategy implementation. It does not hold for the weaker concepts of Nash or Bayes–Nash equilibrium implementation.

3 Quasilinear Preferences

If we are to design a dominant-strategy truthful mechanism that is not dictatorial, we are going to have to relax some of the conditions of the Gibbard–Satterthwaite theorem. First, we relax the requirement that agents be able to express any preferences and replace it with the requirement that agents be able to express any preferences in a limited set. Second, we relax the condition that the mechanism be onto. We now introduce our limited set of preferences.

Definition 7.5 (Quasilinear utility function) *Agents have quasilinear utility functions (or quasilinear preferences) in an n -player Bayesian game when the set of outcomes is $O = X \times \mathbb{R}^n$ for a finite set X , and the utility of an agent i given joint type θ is given by $u_i(o, \theta) = u_i(x, \theta) - p_i$, where $o = (x, p)$ is an element of O , $u_i : X \times \Theta \mapsto \mathbb{R}$ is an arbitrary function.*

Intuitively, we split outcomes into two pieces that are linearly related. First, X represents a finite set of non-monetary outcomes, such as the allocation of an object to one of the bidders in an auction or the selection of a candidate in an election. Second, p_i is the (possibly negative) payment made by agent i to the mechanism, such as a payment to the auctioneer.

What does it mean to assume that agents' preferences are quasilinear? First, it means that we are in a setting in which the mechanism can choose to charge or reward the agents by an arbitrary monetary amount. Second, and more restrictive, it means that an agent's degree of preference for the selection of any choice $x \in X$ is independent of its degree of preference for having to pay the mechanism some amount $p_i \in \mathbb{R}$. Thus an agent's utility for a choice cannot depend on the total amount of money that it has (e.g., an agent cannot value having a yacht more if it is rich than if it is poor). Finally, it means that agents care only about the choice selected and about their own payments: in particular, they do not care about the monetary payments made or received by other agents.

Strictly speaking, we have defined quasilinear preferences in a way that fixes the set of agents. However, we generally consider families of quasilinear problems, for any set of agents. In the following we assume that a quasilinear utility function is still defined when any one agent is taken away. In this case the set of non-monetary outcomes must be updated (e.g., in an auction setting the missing agent cannot be the winner), and is denoted by O_{-i} . Similarly, the utility functions u_i and the choice function C must be updated accordingly.

We have also made another restrictive assumption about quasilinear preferences, albeit one commonly made: we assume that the agent values money in the same units as it values utility. This assumption is called *transferable utility*, because it means that utility can be shifted from one agent to another through monetary transfers. It implies a second assumption, called *risk neutrality*, which means that the agent's value for a unit of currency is independent of the total amount of money the agent has. For a discussion of what happens when these assumptions are violated, please see [29].

3.1 Mechanism Design in the Quasilinear Setting

Now that we have defined the quasilinear preference model, we can talk about the design of mechanisms for agents with these preferences. We concentrate on Bayesian games because most mechanism design is performed in such domains.

First, we point out that since quasilinear preferences split the outcome space into two parts, we can modify our formal definition of a mechanism accordingly.

Definition 7.6 (Quasilinear mechanism) A mechanism in the quasilinear setting (for a Bayesian game setting $(N, O = X \times \mathbb{R}^n, \Theta, p, u)$) is a triple (A, χ, \wp) , where

- $A = A_1 \times \cdots \times A_n$, where A_i is the set of actions available to agent $i \in N$,
- $\chi : A \mapsto \Pi(X)$ maps each action profile to a distribution over choices, and
- $\wp : A \mapsto \mathbb{R}^n$ maps each action profile to a payment for each agent.

In effect, we have split the function M into two functions χ and \wp , where χ is the *choice rule* and \wp is the *payment rule*. We will use the notation \wp_i to denote the payment function for agent i .

A direct revelation mechanism in the quasilinear setting is one in which each agent is asked to state its type.

Definition 7.7 (Direct quasilinear mechanism) A direct quasilinear mechanism (for a Bayesian game setting $(N, O = X \times \mathbb{R}^n, \Theta, p, u)$) is a pair (χ, \wp) . It defines a standard mechanism in the quasilinear setting, where for each i , $A_i = \Theta_i$.

In many quasilinear mechanism design settings it is helpful to make the assumption that agents' utilities depend only on their own types, a property that we call *conditional utility independence*.²

Definition 7.8 (Conditional utility independence) A Bayesian game exhibits conditional utility independence if for all agents $i \in N$, for all outcomes $o \in O$ and for all pairs of joint types θ and $\theta' \in \Theta$ for which $\theta_i = \theta'_i$, it holds that $u_i(o, \theta) = u_i(o, \theta')$.

We will assume conditional utility independence for the rest of this chapter. Thus, we can write an agent i 's utility function as $u_i(o, \theta_i)$, since it does not depend on the other agents' types. We can also refer to an agent's *valuation* for choice $x \in X$, written $v_i(x) = u_i(x, \theta)$. v_i should be thought of as the maximum amount of money that i would be willing to pay to get the mechanism designer to implement choice x – in fact, having to pay this much would exactly make i indifferent about whether it was offered this deal or not.³ Note that an agent's valuation depends on its type, even though we do not explicitly refer to θ_i . In the future when we discuss direct quasilinear mechanisms, we will usually mean mechanisms that ask agents to declare their valuations for each choice; of course, this alternate definition is equivalent to Definition 7.7. Let V_i denote the set of all possible valuations for

²This assumption is sometimes referred to as *privacy*. We avoid that terminology here because the assumption does not imply that agents cannot learn about others' utility functions by observing their own types.

³Observe that here we rely upon the assumption of risk neutrality discussed earlier. Furthermore, observe that it is also meaningful to extend the concept of valuation beyond settings in which conditional utility independence holds; in such cases, we say that agents do not know their own valuations. We describe such settings in [29].

agent i . We will use the notation $\hat{v}_i \in V_i$ to denote the valuation that agent i declares to such a direct mechanism, which may be different from its true valuation v_i . We denote the vector of all agents' declared valuations as \hat{v} and the set of all possible valuation vectors as V . Finally, we denote the vector of declared valuations from all agents other than i as \hat{v}_{-i} .

Now we can state some properties that it is common to require of quasilinear mechanisms.

Definition 7.9 (Truthfulness) *A quasilinear mechanism is truthful if it is direct and $\forall i \forall v_i$, agent i 's equilibrium strategy is to adopt the strategy $\hat{v}_i = v_i$.*

Of course, this is equivalent to the definition of truthfulness that we gave in Section 2.2; we have simply updated the notation for the quasilinear utility setting.

Definition 7.10 (Efficiency) *A quasilinear mechanism is strictly Pareto efficient, or just efficient, if in equilibrium it selects a choice x such that $\forall v \forall x', \sum_i v_i(x) \geq \sum_i v_i(x')$.*

That is, an efficient mechanism selects the choice that maximizes the sum of agents' utilities, disregarding the monetary payments that agents are required to make. We describe this property as *economic efficiency* when there is a danger that it will be confused with other (e.g., computational) notions of efficiency. Observe that efficiency is defined in terms of agents' true valuations, not their declared valuations. This condition is also known as *social welfare maximization*.

Definition 7.11 (Budget balance) *A quasilinear mechanism is budget balanced when $\forall v, \sum_i \phi_i(s(v)) = 0$, where s is the equilibrium strategy profile.*

In other words, regardless of the agents' types, the mechanism collects and disburses the same amount of money from and to the agents, meaning that it makes neither a profit nor a loss. Sometimes we relax this condition and require only *weak budget balance*, meaning that $\forall v, \sum_i \phi_i(s(v)) \geq 0$ (i.e., the mechanism never takes a loss, but it may make a profit). Finally, we can require that either strict or weak budget balance hold *ex ante*, which means that $\mathbb{E}_v [\sum_i \phi_i(s(v))]$ is either equal to or greater than zero. (That is, the mechanism is required to break even or make a profit only on expectation.)

Definition 7.12 (Ex interim individual rationality) *A quasilinear mechanism is ex interim individually rational when*

$$\forall i \forall v_i, \mathbb{E}_{v_{-i}|v_i} [v_i(\chi(s_i(v_i), s_{-i}(v_{-i}))) - \phi_i(s_i(v_i), s_{-i}(v_{-i}))] \geq 0,$$

where s is the equilibrium strategy profile.

This condition requires that no agent loses by participating in the mechanism. We call it *ex interim* because it holds for *every* possible valuation for agent i , but averages over the possible valuations of the other agents. This approach makes sense because it requires that, based on the information that an agent has when it chooses to participate in a mechanism, no agent would be better off choosing not to participate. Of course, we can also strengthen the condition to say that no agent *ever* loses by participation.

Definition 7.13 (Ex post individual rationality) A quasilinear mechanism is *ex post individually rational* when $\forall i \forall v, v_i(\chi(s(v))) - \wp_i(s(v)) \geq 0$, where s is the equilibrium strategy profile.

We can also restrict mechanisms based on their computational requirements rather than their economic properties.

Definition 7.14 (Tractability) A quasilinear mechanism is *tractable* when $\forall a \in A$, $\chi(a)$ and $\wp(a)$ can be computed in polynomial time.

Finally, in some domains there will be many possible mechanisms that satisfy the constraints we choose, meaning that we need to have some way of choosing among them. (And as we will see later, for other combinations of constraints no mechanisms exist at all.) The usual approach is to define an optimization problem that identifies the optimal outcome in the feasible set. For example, although we have defined efficiency as a constraint, it is also possible to soften the constraint and require the mechanism to achieve as much social welfare as possible. Here we define some other quantities that a mechanism designer can seek to optimize.

First, the mechanism designer can take a selfish perspective. Interestingly, this goal turns out to be quite different from the goal of maximizing social welfare. (We give an example of the differences between these approaches when we consider single-good auctions in Section 5.)

Definition 7.15 (Revenue maximization) A quasilinear mechanism is *revenue maximizing* when, among the set of functions χ and \wp that satisfy the other constraints, the mechanism selects the χ and \wp that maximize $\mathbb{E}_v [\sum_i \wp_i(s(v))]$, where $s(v)$ denotes the agents' equilibrium strategy profile.

The mechanism designer might be concerned with selecting a *fair* outcome. However, the notion of fairness can be tricky to formalize. For example, an outcome that fines all agents \$100 and makes a choice that all agents hate equally is in some sense fair, but it does not seem desirable. Here we define so-called *maxmin fairness*, which says that the fairest outcome is the one that makes the

least-happy agent the happiest. We also take an expected value over different valuation vectors, but we could instead have required a mechanism that does the best in the worst case.

Definition 7.16 (Maxmin fairness) *A quasilinear mechanism is maxmin fair when, among the set of functions χ and \wp that satisfy the other constraints, the mechanism selects the χ and \wp that maximize $\mathbb{E}_v[\min_{i \in N} v_i(\chi(s(v))) - \wp_i(s(v))]$, where $s(v)$ denotes the agents' equilibrium strategy profile.*

Finally, the mechanism designer might not be able to implement a social-welfare-maximizing mechanism (e.g., in order to satisfy a tractability constraint) but may want to get as close as possible. Thus, the goal could be minimizing the *price of anarchy*, the worst-case ratio between optimal social welfare and the social welfare achieved by the given mechanism. Here we also consider the worst case across agent valuations.

Definition 7.17 (Price-of-anarchy minimization) *A quasilinear mechanism minimizes the price of anarchy when, among the set of functions χ and \wp that satisfy the other constraints, the mechanism selects the χ and \wp that minimize*

$$\max_{v \in V} \frac{\max_{x \in X} \sum_{i \in N} v_i(x)}{\sum_{i \in N} v_i(\chi(s(v)))},$$

where $s(v)$ denotes the agents' equilibrium strategy profile in the worst equilibrium of the mechanism – that is, the one in which $\sum_{i \in N} v_i(\chi(s(v)))$ is the smallest.

4 Efficient Mechanisms

Efficiency (Definition 7.10) is often considered to be one of the most important properties for a mechanism to satisfy in the quasilinear setting. One reason is that, whenever an inefficient choice is selected, it is possible to find a set of side payments among the agents with the property that all agents would prefer the efficient choice in combination with the side payments to the inefficient choice. (Intuitively, the sum of agents' valuations for the efficient choice is greater than for the inefficient choice. Thus, the agents who prefer the efficient choice would still strictly prefer it even if they had to make side payments to the other agents so that each of them also strictly preferred the efficient choice.) A great deal of research has considered the design of mechanisms that are guaranteed to select efficient choices when agents follow dominant or equilibrium strategies. In this section we survey these mechanisms.

4.1 Groves Mechanisms

The most important family of efficient mechanisms are the Groves mechanisms.

Definition 7.18 (Groves mechanisms) Groves mechanisms are direct quasilinear mechanisms (χ, \wp) , for which

$$\begin{aligned}\chi(\hat{v}) &= \arg \max_x \sum_i \hat{v}_i(x), \\ \wp_i(\hat{v}) &= h_i(\hat{v}_{-i}) - \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})).\end{aligned}$$

In other words, Groves mechanisms are direct mechanisms in which agents can declare any valuation function \hat{v} (and thus any quasilinear utility function \hat{u}). The mechanism then optimizes its choice assuming that the agents disclosed their true utility function. An agent is made to pay an arbitrary amount $h_i(\hat{v}_{-i})$ that does not depend on its own declaration, and is paid the sum of every other agent's declared valuation for the mechanism's choice. The fact that the mechanism designer has the freedom to choose the h_i functions explains why we refer to the *family* of Groves mechanisms rather than to a single mechanism.

The remarkable property of Groves mechanisms is that they provide a dominant-strategy truthful implementation of a social-welfare-maximizing social choice function. It is easy to see that if a Groves mechanism is dominant-strategy truthful, then it must be social welfare maximizing: the function χ in Definition 7.18 performs exactly the maximization called for by Definition 7.10 when $\hat{v} = v$. Thus, it suffices to show the following.

Theorem 7.3 *Truth-telling is a dominant strategy under any Groves mechanism.*

Proof. Consider a situation where every agent j other than i follows some arbitrary strategy \hat{v}_j . Consider agent i 's problem of choosing the best strategy \hat{v}_i . As a shorthand, we write $\hat{v} = (\hat{v}_{-i}, \hat{v}_i)$. The best strategy for i is one that solves

$$\max_{\hat{v}_i} (v_i(\chi(\hat{v})) - \wp_i(\hat{v})).$$

Substituting in the payment function from the Groves mechanism, we have

$$\max_{\hat{v}_i} \left(v_i(\chi(\hat{v})) - h_i(\hat{v}_{-i}) + \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})) \right).$$

Since $h_i(\hat{v}_{-i})$ does not depend on \hat{v}_i , it is sufficient to solve

$$\max_{\hat{v}_i} \left(v_i(\chi(\hat{v})) + \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})) \right).$$

The only way in which the declaration \hat{v}_i influences the maximization above is through the term $v_i(\chi(\hat{v}))$. If possible, i would like to pick a declaration \hat{v}_i that will lead the mechanism to pick an $x \in X$ which solves

$$\max_x \left(v_i(x) + \sum_{j \neq i} \hat{v}_j(x) \right). \quad (7.1)$$

The Groves mechanism chooses an $x \in X$ as

$$\chi(\hat{v}) = \arg \max_x \left(\sum_i \hat{v}_i(x) \right) = \arg \max_x \left(\hat{v}_i(x) + \sum_{j \neq i} \hat{v}_j(x) \right).$$

Thus, agent i leads the mechanism to select the choice that it most prefers by declaring $\hat{v}_i = v_i$. Because this argument does not depend in any way on the declarations of the other agents, truth-telling is a dominant strategy for agent i . ■

Intuitively, the reason that Groves mechanisms are dominant-strategy truthful is that agents' externalities are internalized. Imagine a mechanism in which agents declared their valuations for the different choices $x \in X$ and the mechanism selected the efficient choice, but in which the mechanism did not impose any payments on agents. Clearly, agents would be able to change the mechanism's choice to another that they preferred by overstating their valuation. Under Groves mechanisms, however, an agent's utility does not depend only on the selected choice, because payments *are* imposed. Since agents are paid the (reported) utility of all the other agents under the chosen allocation, each agent becomes just as interested in maximizing the other agents' utilities as in maximizing its own. Thus, once payments are taken into account, all agents have the same interests.

Groves mechanisms illustrate a property that is generally true of dominant-strategy truthful mechanisms: an agent's payment does not depend on the amount of its own declaration. Although other dominant-strategy truthful mechanisms exist in the quasilinear setting, the next theorem shows that Groves mechanisms are the *only* mechanisms that implement an efficient allocation in dominant strategies among agents with arbitrary quasilinear utilities.

Theorem 7.4 (Green–Laffont) *An efficient social choice function $C : \mathbb{R}^{Xn} \mapsto X \times \mathbb{R}^n$ can be implemented in dominant strategies for agents with unrestricted quasilinear utilities only if $\phi_i(v) = h(v_{-i}) - \sum_{j \neq i} v_j(\chi(v))$.*

We do not give the proof here; it appears in [29]. It has also been shown that Groves mechanisms are unique among Bayes–Nash incentive-compatible efficient mechanisms, in a weaker sense. Specifically, any Bayes–Nash incentive-compatible efficient mechanism corresponds to a Groves mechanism in the sense that each agent makes the same *ex interim* expected payments and hence has the same *ex interim* expected utility under both mechanisms.

4.2 The VCG Mechanism

So far, we have said nothing about how to set the function h_i in a Groves mechanism's payment function. Here we will discuss the most popular answer, which is called the Clarke tax. In the subsequent sections we will discuss some of its properties, but first we define it.

Definition 7.19 (Clarke tax) *The Clarke tax sets the h_i term in a Groves mechanism as*

$$h_i(\hat{v}_{-i}) = \sum_{j \neq i} \hat{v}_j(\chi(\hat{v}_{-i})),$$

where χ is the Groves mechanism allocation function.

The resulting Groves mechanism goes by many names. We will see in Section 5 that the Vickrey auction (invented in 1961) is a special case; thus, in resource allocation settings the mechanism is sometimes known as the *generalized Vickrey auction*. Second, it is also called the *pivot mechanism*; we will explain the rationale behind this name in a moment. From now on, though, we will refer to it as the *Vickrey–Clarke–Groves mechanism* (VCG), naming its contributors in chronological order of their contributions. We restate the full mechanism here.

Definition 7.20 (Vickrey–Clarke–Groves (VCG) mechanism)

The VCG mechanism is a direct quasilinear mechanism (χ, ϕ) , where

$$\begin{aligned} \chi(\hat{v}) &= \arg \max_x \sum_i \hat{v}_i(x), \\ \phi_i(\hat{v}) &= \sum_{j \neq i} \hat{v}_j(\chi(\hat{v}_{-i})) - \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})). \end{aligned}$$

First, note that because the Clarke tax does not depend on an agent i 's own declaration \hat{v}_i , our previous arguments that Groves mechanisms are dominant-strategy truthful and efficient carry over immediately to the VCG mechanism. Now, we try to provide some intuition about the VCG payment rule. Assume that all agents follow their dominant strategies and declare their valuations truthfully. The second sum in the VCG payment rule pays each agent i the sum of every other agent $j \neq i$'s utility for the mechanism's choice. The first sum charges each agent i the sum of every other agent's utility for the choice that *would have been made* had i not participated in the mechanism. Thus, each agent is made to pay its *social cost* – the aggregate impact that its participation has on other agents' utilities.

What can we say about the amounts of different agents' payments to the mechanism? If some agent i does not change the mechanism's choice by its participation – that is, if $\chi(v) = \chi(v_{-i})$ – then the two sums in the VCG payment function will cancel out. The social cost of i 's participation is zero, and so it has to pay

nothing. In order for an agent i to be made to pay a non-zero amount, it must be *pivotal* in the sense that the mechanism's choice $\chi(v)$ is different from its choice without i , $\chi(v_{-i})$. This is why VCG is sometimes called the pivot mechanism – only pivotal agents are made to pay. Of course, it is possible that some agents will *improve* other agents' utilities by participating; such agents will be made to pay a negative amount, or, in other words, will be paid by the mechanism.

Let us consider an example. Recall that we previously discussed the problem of buying a shortest path in a transportation network. We will now determine what route and what payments the VCG mechanism would select. For convenience, we reproduce Figure 7.1 as Figure 7.3, now labeling the nodes so that we have names to refer to the agents (the edges).

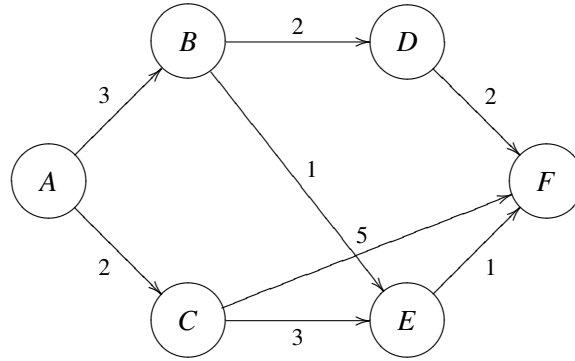


Figure 7.3: Transportation network with selfish agents.

Note that in this example, the numbers labeling the edges in the graph denote agents' costs rather than utilities; thus, an agent's utility is $-c$ if a route involving its edge (having cost c) is selected, and zero otherwise. The $\arg\max$ in χ will amount to cost minimization. Thus, $\chi(v)$ will return the shortest path in the graph, which is $ABEF$. How much will agents have to pay? First, let us consider the agent AC . The shortest path taking its declaration into account has a length of 5 and imposes a cost of -5 on agents other than it (because it is not involved). Likewise, the shortest path without AC 's declaration also has a length of 5. Thus, its payment is $p_{AC} = (-5) - (-5) = 0$. This is what we expect, since AC is not pivotal. Clearly, by the same argument BD , CE , CF , and DF will all be made to pay zero. Now let us consider the pivotal agents. The shortest path taking AB 's declaration into account has a length of 5, and imposes a cost of 2 on other agents. The shortest path without AB is $ACEF$, which has a cost of 6. Thus $p_{AB} = (-6) - (-2) = -4$: AB is paid 4 for its participation. Arguing similarly, you can verify that $p_{BE} = (-6) - (-4) = -2$, and $p_{EF} = (-7) - (-4) = -3$.

Note that although EF had the same cost as BE , they are paid different amounts for the use of their edges. This occurs because EF has more *market power*: for the other agents, the situation without EF is worse than the situation without BE .

4.3 Properties of VCG

4.3.1 VCG and Individual Rationality

We have seen that Groves mechanisms are dominant-strategy truthful and efficient. We have also seen that no other mechanism has both of these properties in general quasilinear settings. Thus, we might be a bit worried that we have not been able to guarantee either individual rationality or budget balance, two properties that are quite important in practice. (Recall that individual rationality means that no agent would prefer not to participate in the mechanism; budget balance means that the mechanism does not lose money.) We will consider budget balance in Section 4.4; here we investigate individual rationality.

As it turns out, our worry is well founded: even with the freedom to set h_i , we cannot find a mechanism that guarantees us individual rationality in an unrestricted quasilinear setting. However, we are often able to guarantee the strongest variety of individual rationality when the setting satisfies certain mild restrictions.

Definition 7.21 (Choice-set monotonicity) *An environment exhibits choice-set monotonicity if $\forall i, X_{-i} \subseteq X$ (removing any agent weakly decreases – that is, never increases – the mechanism’s set of possible choices X).*

Definition 7.22 (No negative externalities) *An environment exhibits no negative externalities if $\forall i \forall x \in X_{-i}, v_i(x) \geq 0$ (every agent has zero or positive utility for any choice that can be made without its participation).*

Theorem 7.5 *The VCG mechanism is ex post individually rational when the choice-set monotonicity and no negative externalities properties hold.*

Consider a market consisting of a set of agents interested in buying a single unit of a good such as a share of stock and another set of agents interested in selling a single unit of this good. The choices in this environment are sets of buyer–seller pairings. (Prices are imposed through the payment function.) If a new agent is introduced into the market, no previously existing pairings become infeasible, but new ones become possible; thus choice-set monotonicity is satisfied. Because agents have zero utility both for choices that involve trades between other agents and no trades at all, there are no negative externalities.

4.3.2 VCG and Weak Budget Balance

What about weak budget balance, the requirement that the mechanism will not lose money? Our two previous conditions, choice-set monotonicity and no negative externalities, are not sufficient to guarantee weak budget balance: for example, the “buying the shortest path” example given earlier satisfied these two conditions, but we saw that the VCG mechanism paid out money and did not collect any. Thus, we will have to explore further restrictions to the quasilinear setting.

Definition 7.23 (No single-agent effect) *An environment exhibits no single-agent effect if $\forall i, \forall v_{-i}, \forall x \in \arg \max_y \sum_j v_j(y)$ there exists a choice x' that is feasible without i and that has $\sum_{j \neq i} v_j(x') \geq \sum_{j \neq i} v_j(x)$.*

In other words, removing any agent does not worsen the total value of the best solution to the others, regardless of their valuations. For example, this property is satisfied in a single-sided auction – dropping an agent just reduces the amount of competition in the auction, making the others better off.

Theorem 7.6 *The VCG mechanism is weakly budget balanced when the no single-agent effect property holds.*

Indeed, we can say something more about VCG’s revenue properties: restricting ourselves to settings in which VCG is *ex post* individually rational as discussed earlier, and comparing to all other efficient and *ex interim* individually rational mechanisms, VCG turns out to collect the maximal amount of revenue from the agents. This is somewhat surprising, since this result does not require dominant strategies, and hence compares VCG to all Bayes–Nash mechanisms. A useful corollary of this result is that VCG is as budget balanced as any efficient mechanism can be: it satisfies weak budget balance in every case where *any* dominant strategy, efficient, and *ex interim* individually rational mechanism would be able to do so.

4.3.3 Drawbacks of VCG

The VCG mechanism is one of the most powerful positive results in mechanism design: it gives us a general way of constructing dominant-strategy truthful mechanisms to implement social-welfare-maximizing social choice functions in quasilinear settings. We have seen that no fundamentally different mechanism could do the same job. And VCG gives us even more: under the right conditions it further guarantees *ex post* individual rationality and weak budget balance. Thus, it is not surprising that this mechanism has been enormously influential and continues to be widely studied. However, despite these attractive properties, VCG also has

some undesirable characteristics. We discuss these at length in [29], but briefly list them here.

1. Agents must fully disclose private information.
2. VCG is susceptible to collusion.
3. VCG is not “frugal”: prices can be many times higher than the true value of the best allocation involving no winning agents.
4. Excluding bidders can (unboundedly) increase revenue.
5. It is impossible to return all of VCG’s revenue to the agents without distorting incentives.
6. The problem of identifying the $\arg \max$ can be computationally intractable (e.g., see Section 7).

Having listed these problems, however, we offer a caveat: although there exist mechanisms that circumvent each of the drawbacks we discuss, none of the drawbacks are *unique* to VCG, or even to Groves mechanisms. Indeed, in some cases the problems are known to crop up in extremely broad classes of mechanisms.

4.4 Budget Balance and Efficiency

In Section 4.3.2 we identified a realistic case in which the VCG mechanism is weakly budget balanced. However, we also noted that there exist other important and practical settings in which the no single-agent effect property does not hold. For example, define a *simple exchange* as an environment consisting of buyers and sellers with quasilinear utility functions, all interested in trading a single identical unit of some good. The no single-agent effect property is not satisfied in a simple exchange because dropping a seller could make some buyer worse off and vice versa. Can we find some other argument to show that VCG will remain budget balanced in this important setting?

It turns out that neither VCG nor any other Groves mechanism is budget balanced in the simple exchange setting. (Recall Theorem 7.4: only Groves mechanisms are both dominant-strategy incentive compatible and efficient.)

Theorem 7.7 (Green–Laffont; Hurwicz) *No dominant-strategy incentive-compatible mechanism is always both efficient and weakly budget balanced, even if agents are restricted to the simple exchange setting.*

Furthermore, another seminal result showed that a similar problem arises in the broader class of Bayes–Nash incentive-compatible mechanisms (which, recall, includes the class of dominant-strategy incentive-compatible mechanisms) if we also require *ex interim* individual rationality and allow general quasilinear utility functions.

Theorem 7.8 (Myerson–Satterthwaite) *No Bayes–Nash incentive-compatible mechanism is always simultaneously efficient, weakly budget balanced, and ex interim individually rational, even if agents are restricted to quasilinear utility functions.*

5 Single-Good Auctions

We now consider the problem of allocating (discrete) resources among selfish agents in a multiagent system. Auctions – an interesting and important application of mechanism design – turn out to provide a general solution to this problem. We describe various different flavors of auctions, including single-good and combinatorial auctions. In each case, we survey some of the key theoretical, practical, and computational insights from the literature.

The auction setting is important for two reasons. First, auctions are widely used in real life, in consumer, corporate, as well as government settings. Millions of people use auctions daily on Internet consumer web sites to trade goods. More complex types of auctions have been used by governments around the world to sell important public resources such as access to electromagnetic spectrum. Indeed, all financial markets constitute a type of auction (one of the family of so-called *double auctions*). Auctions are also often used in computational settings to efficiently allocate bandwidth and processing power to applications and users.

The second – and more fundamental – reason to care about auctions is that they provide a general theoretical framework for understanding resource allocation problems among self-interested agents. Formally speaking, an auction is any protocol that allows agents to indicate their interest in one or more resources and that uses these indications of interest to determine both an allocation of resources and a set of payments by the agents. Thus, auctions are important for a wide range of computational settings (e.g., the sharing of computational power in a grid computer on a network) that would not normally be thought of as auctions and that might not even use money as the basis of payments.

It is important to realize that the most familiar type of auction – the ascending-bid, English auction – is a drop in the ocean of auction types. Indeed, since auctions are simply mechanisms for allocating goods, there is an infinite number of auction types. In the most familiar types of auctions there is one good for

sale, one seller, and multiple potential buyers. Each buyer has its own valuation for the good, and each wishes to purchase it at the lowest possible price. These auctions are called *single-sided*, because there are multiple agents on only one side of the market. Our task is to design a protocol for this auction that satisfies certain desirable global criteria. For example, we might want an auction protocol that maximizes the expected revenue of the seller. Or, we might want an auction that is economically efficient; that is, one that guarantees that the potential buyer with the highest valuation ends up with the good.

Given the popularity of auctions, on the one hand, and the diversity of auction mechanisms, on the other, it is not surprising that the literature on the topic is vast. In this section we provide a taste for this literature, beginning by concentrating on auctions for selling a single good. We explore richer settings later in the chapter.

5.1 Canonical Auction Families

To give a feel for the broad space of single-good auctions, we start by describing some of the most famous families: English, Japanese, Dutch, and sealed-bid auctions.

5.1.1 English Auctions

The *English auction* is perhaps the best-known family of auctions, since in one form or another such auctions are used in the venerable, old-guard auction houses, as well as most of the online consumer auction sites. The auctioneer sets a starting price for the good, and agents then have the option to announce successive bids, each of which must be higher than the previous bid (usually by some minimum increment set by the auctioneer). The rules for when the auction closes vary; in some instances the auction ends at a fixed time, in others it ends after a fixed period during which no new bids are made, in others at the latest of the two, and in still other instances at the earliest of the two. The final bidder, who by definition is the agent with the highest bid, must purchase the good for the amount of its final bid.

5.1.2 Japanese Auctions

The *Japanese auction*⁴ is similar to the English auction in that it is an ascending-bid auction but is different otherwise. Here the auctioneer sets a starting price for the good, and each agent must choose whether or not to be “in,” that is, whether it is willing to purchase the good at that price. The auctioneer then calls out

⁴Unlike the terms *English* and *Dutch*, the term *Japanese* is not used universally; however, it is commonly used, and there is no competing name for this family of auctions.

successively increasing prices in a regular fashion,⁵ and after each call each agent must announce whether it is still in. When an agent drops out it is irrevocable, and it cannot re-enter the auction. The auction ends when there is exactly one agent left in; the agent must then purchase the good for the current price.

5.1.3 Dutch Auctions

In a *Dutch auction* the auctioneer begins by announcing a high price and then proceeds to announce successively lower prices in a regular fashion. In practice, the descending prices are indicated by a clock that all of the agents can see. The auction ends when the first agent signals the auctioneer by pressing a buzzer and stopping the clock; the signaling agent must then purchase the good for the displayed price. This auction gets its name from the fact that it is used in the Amsterdam flower market; in practice, it is most often used in settings where goods must be sold quickly.

5.1.4 Sealed-Bid Auctions

All the auctions discussed so far are considered *open-outcry* auctions, in that all the bidding is done by calling out the bids in public (however, as we will discuss shortly, in the case of the Dutch auction this is something of an optical illusion). The family of *sealed-bid auctions*, probably the best known after English auctions, is different. In this case, each agent submits to the auctioneer a secret, “sealed” bid for the good that is not accessible to any of the other agents. The agent with the highest bid must purchase the good, but the price at which it does so depends on the type of sealed-bid auction. In a first-price sealed-bid auction (or simply *first-price auction*) the winning agent pays an amount equal to its own bid. In a *second-price auction* it pays an amount equal to the next highest bid (i.e., the highest rejected bid). The second-price auction is also called the *Vickrey auction*. In general, in a *kth-price auction* the winning agent purchases the good for a price equal to the *k*th highest bid.

5.2 Auctions as Bayesian Mechanisms

We now move to a more formal investigation of single-good auctions. Our starting point is the observation that choosing an auction that has various desired properties is a mechanism design problem. Ordinarily we assume that agents’ utility functions in an auction setting are quasilinear. To define an auction as a quasilinear mechanism (see Definition 7.6) we must identify the following elements:

⁵In the theoretical analyses of this auction, the assumption is usually that the prices rise continuously.

- set of agents N ,
- set of outcomes $O = X \times \mathbb{R}^n$,
- set of actions A_i available to each agent $i \in N$,
- choice function χ that selects one of the outcomes given the agents' actions, and
- payment function \wp that determines what each agent must pay given all agents' actions.

In an auction, the possible outcomes O consist of all possible ways to allocate the good – the set of choices X – and all possible ways of charging the agents. The agents' actions will vary in different auction types. In a sealed-bid auction, each set A_i is an interval from \mathbb{R} (i.e., an agent's action is the declaration of a bid amount between some minimum and maximum value). A Japanese auction is an imperfect-information extensive-form game with chance nodes, and so in this case the action space is the space of all policies the agent could follow (i.e., all different ways of acting conditioned on different observed histories). As in all mechanism design problems, the choice and payment functions χ and \wp depend on the objective of the auction, such as achieving an efficient allocation or maximizing revenue.

A Bayesian game with quasilinear preferences includes two more ingredients that we need to specify: the common prior and the agents' utility functions. We will say more about the common prior – the distribution from which the agents' types are drawn – later; here, just note that the definition of an auction as a Bayesian game is incomplete without it. Considering the agents' utility functions, note that the quasilinearity assumption (see Definition 7.5) allows us to write $u_i(o, \theta_i) = u_i(x, \theta_i) - p_i$.

We are left with the task of describing the agents' valuations: their utilities for different allocations of the goods $x \in X$. Auction theory distinguishes between a number of different settings here. One of the best-known and most extensively studied is the *independent private value* (IPV) setting. In this setting all agents' valuations are drawn independently from the same (commonly known) distribution, and an agent's type (or “signal”) consists only of its own valuation, giving it no information about the valuations of the others. An example in which the IPV setting is appropriate is in auctions consisting of bidders with personal tastes who aim to buy a piece of art purely for their own enjoyment. We will assume that agents have independent private values; in [29] we also explore an alternative, the common-value assumption.

5.3 Second-Price, Japanese, and English Auctions

Let us now consider whether the second-price sealed-bid auction, which is a direct mechanism, is truthful (i.e., whether it provides incentive for the agents to bid their true values). The following very conceptually straightforward proof shows that in the IPV case it is truthful.

Theorem 7.9 *In a second-price auction where bidders have independent private values, truth-telling is a dominant strategy.*

The second-price auction is a special case of the VCG mechanism, and hence of the Groves mechanism. Thus, Theorem 7.9 follows directly from Theorem 7.3. However, a proof of this narrower claim is considerably more intuitive than the general argument.

Proof. Assume that all bidders other than i bid in some arbitrary way, and consider i 's best response. First, consider the case where i 's valuation is larger than the highest of the other bidders' bids. In this case i would win and would pay the next-highest bid amount, as illustrated in Figure 7.4a. Could i be better off by bidding dishonestly in this case? If it bid higher, it would still win and would still pay the same amount, as illustrated in Figure 7.4b. If it bid lower, it would either still win and still pay the same amount (Figure 7.4c) or lose and pay zero (Figure 7.4d).⁶ Since i gets non-negative utility for receiving the good at a price less than or equal to its valuation, i cannot gain, and would sometimes lose by bidding dishonestly in this case. Now consider the other case, where i 's valuation is less than at least one other bidder's bid. In this case i would lose and pay zero (Figure 7.4e). If it bid less, it would still lose and pay zero (Figure 7.4f). If it bid more, either it would still lose and pay zero (Figure 7.4g) or it would win and pay more than its valuation (Figure 7.4h), achieving negative utility. Thus again, i cannot gain, and would sometimes lose by bidding dishonestly in this case. ■

In the IPV case, we can identify strong relationships between the second-price auction and Japanese and English auctions. Consider first the comparison between second-price and Japanese auctions. In both cases the bidder must select a number (in the sealed-bid case the number is the one written down, and in the Japanese case it is the price at which the agent will drop out); the bidder with highest amount wins, and pays the amount selected by the second-highest bidder. The difference between the auctions is that information about other agents' bid amounts is disclosed in the Japanese auction. In the sealed-bid auction an agent's bid amount must be selected without knowing anything about the amounts

⁶Figure 7.4d is oversimplified: the winner will not always pay i 's bid in this case. (Do you see why?)

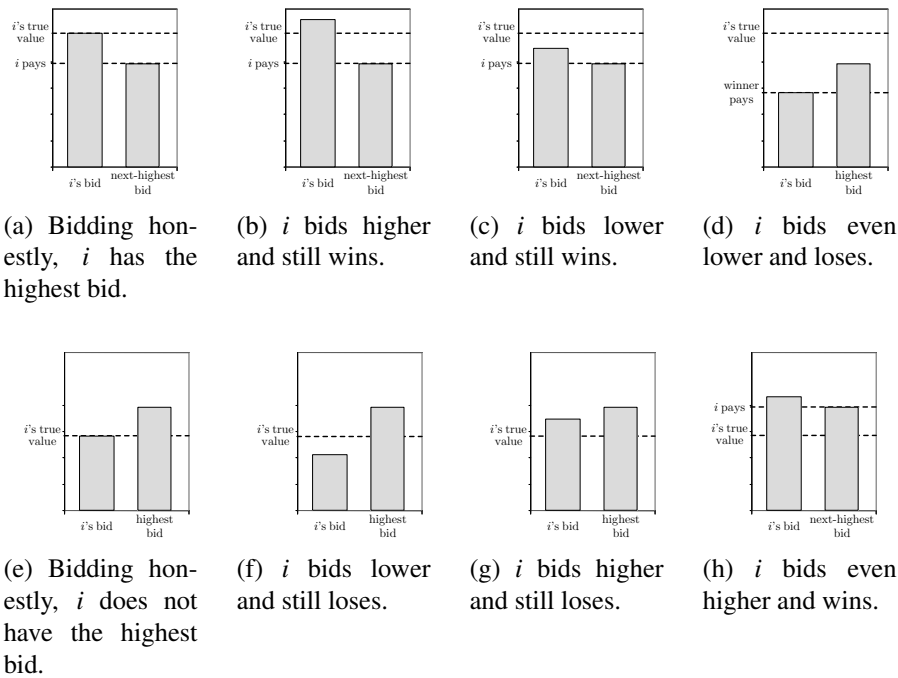


Figure 7.4: A case analysis to show that honest bidding is a dominant strategy in a second-price auction with independent private values.

selected by others, whereas in the Japanese auction the amount can be updated based on the prices at which lower bidders are observed to drop out. In general, this difference can be important; however, it makes no difference in the IPV case. Thus, Japanese auctions are also dominant-strategy truthful when agents have independent private values.

Obviously, the Japanese and English auctions are closely related. Thus, it is not surprising to find that second-price and English auctions are also similar. One connection can be seen through *proxy bidding*, a service offered on some online auction sites such as eBay. Under proxy bidding, a bidder tells the system the maximum amount it is willing to pay. The user can then leave the site, and the system bids as the bidder's proxy: every time the bidder is outbid, the system will respond with a bid one increment higher, until the bidder's maximum is reached. It is easy to see that if all bidders use the proxy service and update it only once, what occurs will be identical to a second-price auction (excepting that the winner's payment may be one bid increment higher).

The main complication with English auctions is that bidders can place so-called *jump bids*: bids that are greater than the previous high bid by more than the minimum increment. Although it seems relatively innocuous, this feature compli-

cates analysis of such auctions. Indeed, when an ascending auction is analyzed it is usually the Japanese variant, not the English.

5.4 First-Price and Dutch Auctions

Let us now consider first-price auctions. The first observation we can make is that the Dutch auction and the first-price auction, while quite different in appearance, are actually the same auction (in the technical jargon, they are *strategically equivalent*). In both auctions each agent must select an amount without knowing about the other agents' selections; the agent with the highest amount wins the auction, and must purchase the good for that amount. Strategic equivalence is a very strong property: it says the auctions are exactly the same no matter what risk attitudes the agents have, and no matter what valuation model describes their utility functions. This being the case, it is interesting to ask why both auction types are held in practice. One answer is that they make a trade-off between time complexity and communication complexity. First-price auctions require each bidder to send a message to the auctioneer, which could be unwieldy with a large number of bidders. Dutch auctions require only a single bit of information to be communicated to the auctioneer, but requires the auctioneer to broadcast prices.

Of course, all this talk of equivalence does not help us to understand anything about how an agent should actually *bid* in a first-price or Dutch auction. Unfortunately, unlike the case of second-price auctions, here we do not have the luxury of dominant strategies, and must thus resort to Bayes–Nash equilibrium analysis. Let us assume that agents have independent private valuations, that agents are risk neutral, and that their valuations are drawn uniformly from some interval, say $[0, 1]$. Let s_i denote the bid of player i , and v_i denote its true valuation. Thus if player i wins, its payoff is $u_i = v_i - s_i$; if it loses, it is $u_i = 0$. Then it can be shown that there is an equilibrium in which each player bids a fraction of its true valuation that depends on the number of participants.

Theorem 7.10 *In a first-price sealed-bid auction with n risk-neutral agents whose valuations are independently drawn from a uniform distribution on the same bounded interval of the real numbers, the unique symmetric equilibrium is given by the strategy profile $(\frac{n-1}{n}v_1, \dots, \frac{n-1}{n}v_n)$.*

In other words, the unique equilibrium of the auction occurs when each player bids $\frac{n-1}{n}$ of its valuation. This theorem can be proved using calculus, but the proof is long and tedious. Furthermore, this proof only shows how to *verify* an equilibrium strategy. How do we identify one in the first place? Although it is also possible to do this from first principles (at least for straightforward auctions such as first-price), we will explain a simpler technique below.

5.5 Revenue Equivalence

Of the large (in fact, infinite) space of auctions, which one should an auctioneer choose? To a certain degree, the choice does not matter, a result formalized by the following theorem.

Theorem 7.11 (Revenue equivalence theorem) *Assume that each of n risk-neutral agents has an independent private valuation for a single good at auction, drawn from a common cumulative distribution $F(v)$ that is strictly increasing and atomless on $[\underline{v}, \bar{v}]$. Then any efficient⁷ auction mechanism in which any agent with valuation \underline{v} has an expected utility of zero yields the same expected revenue, and hence results in any bidder with valuation v_i making the same expected payment.*

We omit this proof, as it is fairly technical; we refer interested readers to [29]. This theorem tells us that when bidders are risk neutral and have independent private valuations, all the auctions we have spoken about so far – English, Japanese, Dutch, and all sealed-bid auction protocols – are revenue equivalent. The revenue equivalence theorem is useful beyond telling the auctioneer that it does not much matter which auction she holds, however. It is also a powerful analytic tool. In particular, we can make use of this theorem to identify equilibrium bidding strategies for auctions that meet the theorem’s conditions.

For example, let us consider again the n -bidder first-price auction discussed in Theorem 7.10. Does this auction satisfy the conditions of the revenue equivalence theorem? The second condition is easy to verify; the first is harder, because it speaks about the outcomes of the auction under the equilibrium bidding strategies. For now, let us assume that the first condition is satisfied as well.

The revenue equivalence theorem only helps us, of course, if we use it to compare the revenue from a first-price auction with that of another auction that we already understand. The second-price auction serves nicely in this latter role: we already know its equilibrium strategy, and it meets the conditions of the theorem. We know from the proof that a bidder of the same type will make the same expected payment in both auctions. In both of the auctions we are considering, a bidder’s payment is zero unless it wins. Thus a bidder’s expected payment conditional on being the winner of a first-price auction must be the same as its expected payment conditional on being the winner of a second-price auction. Since the first-price auction is efficient, we can observe that under the symmetric equilibrium agents will bid this amount all the time: if the agent is the high bidder then it will make the right expected payment, and if it is not, its bid amount will not matter.

⁷Here we make use of the definition of economic efficiency given in Definition 7.10. Equivalently, we could require that the auction has a symmetric and increasing equilibrium and always allocates the good to an agent who placed the highest bid.

We must now find an expression for the expected value of the second-highest valuation, given that bidder i has the highest valuation. It is helpful to know the formula for the k th *order statistic*, in this case of draws from the uniform distribution. The k th order statistic of a distribution is a formula for the expected value of the k th-largest of n draws. For n independent and identically distributed draws from $[0, v_{\max}]$, the k th order statistic is

$$\frac{n+1-k}{n+1}v_{\max}. \quad (7.2)$$

If bidder i 's valuation v_i is the highest, then there are $n-1$ other valuations drawn from the uniform distribution on $[0, v_i]$. Thus, the expected value of the second-highest valuation is the first-order statistic of $n-1$ draws from $[0, v_i]$. Substituting into Equation (7.2), we have $\frac{(n-1)+1-(1)}{(n-1)+1}(v_i) = \frac{n-1}{n}v_i$. This confirms the equilibrium strategy from Theorem 7.10. It also gives us a suspicion (that turns out to be correct) about the equilibrium strategy for first-price auctions under valuation distributions other than uniform: each bidder bids the expectation of the second-highest valuation, conditioned on the assumption that its own valuation is the highest.

A caveat must be given about the revenue equivalence theorem: this result makes an “if” statement, not an “if and only if” statement. That is, while it is true that all auctions satisfying the theorem’s conditions must yield the same expected revenue, it is *not* true that all strategies yielding that expected revenue constitute equilibria. Thus, after using the revenue equivalence theorem to identify a strategy profile that one believes to be an equilibrium, one must then prove that this strategy profile is indeed an equilibrium. This should be done in the standard way, by assuming that all but one of the agents play according to the equilibrium and show that the equilibrium strategy is a best response for the remaining agent.

Finally, recall that we assumed above that the first-price auction allocates the good to the bidder with the highest valuation. The reason it was reasonable to do this (although we could instead have proved that the auction has a symmetric, increasing equilibrium) is that we have to check the strategy profile derived using the revenue equivalence theorem anyway. Given the equilibrium strategy, it is easy to confirm that the bidder with the highest valuation will indeed win the good.

6 Position Auctions

Search engines make most of their money – many billions of dollars annually – by selling advertisements through what are called *position auctions*. In these auctions, multiple different goods (keyword-specific “slots,” usually a list on the right-hand side of a page of search results) are simultaneously offered for sale to

interested advertisers. Slots are considered to be more valuable the closer they are to the top of the page, because this affects their likelihood of being clicked by a user. Advertisers place bids on keywords of interest, and every time a user searches for a keyword on which advertisers have bid, an auction is held. The outcome of this auction is a decision about which ads will appear on the search results page and in which order. Advertisers are required to pay only if a user clicks on their ad.

We now give a formal model. As before, let N be the set of bidders (advertisers), and let v_i be i 's (commonly known) valuation for getting a click. Let $b_i \in \mathbb{R}_+$ denote i 's bid, and let $b_{(j)}$ denote the j th-highest bid, or 0 if there are fewer than j bids. Let $G = \{1, \dots, m\}$ denote the set of goods (slots), and let α_j denote the expected number of clicks (the *click-through rate*) that an ad will receive if it is listed in the i th slot. Observe that we assume that α does not depend on the bidder's identity. Observe that our model treats the auction as unrepeated, and assumes that agents know each other's valuations. The single-shot assumption is motivated by the fact that advertisers tend to value clicks additively (i.e., the value derived from a given user clicking on an ad is independent of how many other users clicked earlier), at least when advertisers do not face budget constraints. The perfect-information assumption makes sense because search engines allow bidders either to observe other bids or to figure them out by probing the mechanism.

The generalized first-price auction was the first position auction to be used by search engine companies.

Definition 7.24 (Generalized first-price auction) *The generalized first-price auction (GFP) awards the bidder with the j th-highest bid the j th slot. If bidder i 's ad receives a click, it pays the auctioneer b_i .*

Unfortunately, these auctions do not always have pure-strategy equilibria, even in the unrepeated, perfect-information case. For example, consider three bidders 1, 2, and 3 who value clicks at \$10, \$4, and \$2, respectively, participating in an auction for two slots, where the probability of a click for the two slots is $\alpha_1 = 0.5$ and $\alpha_2 = 0.25$, respectively. Bidder 2 needs to bid at least \$2 to get a slot; suppose it bids \$2.01. Then bidder 1 can win the top slot for a bid of \$2.02. But bidder 2 could get the top slot for \$2.03, increasing its expected utility. If the agents bid by best responding to each other – as has indeed been observed in practice – their bids will increase all the way up to bidder 2's valuation, at which point bidder 2 will drop out, bidder 1 will reduce its bid to bidder 3's valuation, and the cycle will begin again.

The instability of bidding under the GFP led to the introduction of the generalized second-price auction, which is now the dominant mechanism in practice.

Definition 7.25 (Generalized second-price auction) *The generalized second-price auction (GSP) awards the bidder with the j th-highest bid the j th slot. If bidder i 's ad is ranked in slot j and receives a click, it pays the auctioneer $b_{(j+1)}$.*

The GSP is more stable than the GFP. Continuing the example from above, if all bidders bid truthfully, then bidder 1 would pay \$4 per click for the first slot, bidder 2 would pay \$2 per click for the second slot, and bidder 3 would lose. Bidder 1's expected utility would be $0.5(\$10 - \$4) = \$3$; if it bid less than \$4 but more than \$2 it would pay \$2 per click for the second slot and achieve expected utility of $0.25(\$10 - \$2) = \$2$, and if it bid even less then its expected utility would be zero. Thus bidder 1 prefers to bid truthfully in this example. If bidder 2 bid more than \$10 then it would win the top slot for \$10, and would achieve negative utility; thus in this example bidder 2 also prefers honest bidding.

This example suggests a connection between the GSP and the VCG mechanisms. However, these two mechanisms are actually quite different, as becomes clear when we apply the VCG formula to the position auction setting.

Definition 7.26 (VCG) *In the position auction setting, the VCG mechanism awards the bidder with the j th-highest bid the j th slot. If bidder i 's ad is ranked in slot j and receives a click, it pays the auctioneer $\frac{1}{\alpha_j} \sum_{k=j+1}^{m+1} b_{(k)}(\alpha_{k-1} - \alpha_k)$.*

Intuitively, the key difference between the GSP and VCG is that the former does not charge an agent its social cost, which depends on the differences between click-through rates that other agents would receive with and without its presence. Indeed, truthful bidding is not always a good idea under the GSP. Consider the same bidders as in our running example, but change the click-through rate of slot 2 to $\alpha_2 = 0.4$. When all bidders bid truthfully we have already shown that bidder 1 would achieve expected utility of \$3 (this argument did not depend on α_2). However, if bidder 1 changed its bid to \$3, it would be awarded the second slot and would achieve expected utility of $0.4(\$10 - \$2) = \$3.2$. Thus the GSP is not even truthful in equilibrium, let alone in dominant strategies.

What *can* be said about the equilibria of the GSP? Briefly, it can be shown that in the perfect-information setting the GSP has many equilibria. The dynamic nature of the setting suggests that the most stable configurations will be *locally envy free*: no bidder will wish that it could switch places with the bidder who won the slot directly above its own. There exists a locally envy-free equilibrium of the GSP that achieves exactly the VCG allocations and payments. Furthermore, all other locally envy-free equilibria lead to higher revenues for the seller, and hence are worse for the bidders.

What about relaxing the perfect-information assumption? Here, it is possible to construct a generalized *English* auction that corresponds to the GSP, and to show that this English auction has a unique equilibrium with various desirable

properties. In particular, the payoffs under this equilibrium are again the same as the VCG payoffs, and the equilibrium is *ex post*, meaning that it is independent of the underlying valuation distribution.

7 Combinatorial Auctions

We now consider a broader auction setting, in which a whole variety of different goods are available in the same market. Switching to such an auction model is important when bidders' valuations depend strongly on which subset of the goods they receive. Some widely studied practical examples include governmental auctions for the electromagnetic spectrum, energy auctions, corporate procurement auctions, and auctions for paths (e.g., shipping rights, bandwidth) in a network.

More formally, let us consider a setting with a set of bidders $N = \{1, \dots, n\}$ (as before) and a set of goods $G = \{1, \dots, m\}$. Let $v = (v_1, \dots, v_n)$ denote the true *valuation functions* of the different bidders, where for each $i \in N$, $v_i : 2^G \mapsto \mathbb{R}$. We will usually be interested in settings where bidders have *non-additive valuation functions*, for example valuing bundles of goods more than the sum of the values for single goods. We identify two important kinds of non-additivity. First, when two items are *partial substitutes* for each other (e.g., a Sony TV and a Toshiba TV, or, more partially, a CD player and an MP3 player), their combined value is less than the sum of their individual values. Strengthening this condition, when two items are *strict substitutes* their combined value is the same as the value for either one of the goods. For example, consider two non-transferable tickets for seats on the same plane.

Definition 7.27 (Substitutability) *Bidder i 's valuation v_i exhibits substitutability if there exist two sets of goods $G_1, G_2 \subseteq G$, such that $G_1 \cap G_2 = \emptyset$ and $v(G_1 \cup G_2) < v(G_1) + v(G_2)$. When this condition holds, we say that the valuation function v_i is subadditive.*

The second form of non-additivity we will consider is *complementarity*. This condition is effectively the opposite of substitutability: the combined value of goods is greater than the sum of their individual values. For example, consider a left shoe and a right shoe, or two adjacent pieces of real estate.

Definition 7.28 (Complementarity) *Bidder i 's valuation v_i exhibits complementarity if there exist two sets of goods $G_1, G_2 \subseteq G$, such that $G_1 \cap G_2 = \emptyset$ and $v(G_1 \cup G_2) > v(G_1) + v(G_2)$. When this condition holds, we say that the valuation function v_i is superadditive.*

How should an auctioneer sell goods when faced with such bidders? One approach is simply to sell the goods individually, ignoring the bidders' valuations.

This is easy for the seller, but it makes things difficult for the bidders. In particular, it presents them with what is called the *exposure problem*: a bidder might bid aggressively for a set of goods in the hopes of winning a bundle, but succeed in winning only a subset of the goods and therefore pay too much. This problem is especially likely to arise in settings where bidders' valuations exhibit strong complementarities, because in these cases bidders might be willing to pay substantially more for bundles of goods than they would pay if the goods were sold separately.

The next-simplest method is to run essentially separate auctions for the different goods, but to connect them in certain ways. For example, one could hold a multiround (e.g., Japanese) auction, but synchronize the rounds in the different auctions so that as a bidder bids in one auction it has a reasonably good indication of what is transpiring in the other auctions of interest. This approach can be made more effective through the establishment of constraints on bidding that span all the auctions (so-called *activity rules*). For example, bidders might be allowed to increase their aggregate bid amount by only a certain percentage from one round to the next, thus providing a disincentive for bidders to fail to participate in early rounds of the auction and thus improving the information transfer between auctions. Bidders might also be subject to other constraints: for example, a budget constraint could require that a bidder not exceed a certain total commitment across all auctions. Both of these ideas can be seen in some government auctions for electromagnetic spectrum (where the so-called *simultaneous ascending auction* was used) as well as in some energy auctions. Despite some successes in practice, however, this approach has the drawback that it only mitigates the exposure problem rather than eliminating it entirely.

A third approach ties goods together in a more straightforward way: the auctioneer sells all goods in a single auction, and allows bidders to bid directly on bundles of goods. Such mechanisms are called *combinatorial auctions*. This approach eliminates the exposure problem because bidders are guaranteed that their bids will be satisfied “all or nothing.” For example a bidder may be permitted to offer \$100 for the pair (TV, DVD player), or to make a disjunctive offer “either \$100 for TV1 or \$90 for TV2, but not both.” However, we will see that while combinatorial auctions resolve the exposure problem, they raise many other questions. Indeed, these auctions have been the subject of considerable recent study in both economics and computer science.

VCG has some attractive properties when applied to combinatorial auctions. Specifically, it is dominant-strategy truthful, efficient, *ex post* individual rational, and weakly budget balanced (the latter by Theorems 7.5 and 7.6). The VCG combinatorial auction mechanism is not without shortcomings, however, as we already discussed in Section 4.3.3. For example, a bidder who declares its valua-

tion truthfully has two main reasons to worry – one is that the seller will examine its bid before the auction clears and submit a fake bid just below, thus increasing the amount that the agent would have to pay if it wins. (This is called a *shill bid*.) Another possibility is that both its competitors and the seller will learn its true valuation and will be able to exploit this information in a future transaction. Indeed, these two reasons are often cited as reasons why VCG auctions are rarely seen in practice. Other issues include the fact that VCG is vulnerable to collusion among bidders, and, conversely, to one bidder masquerading as several different ones (so-called *pseudonymous bidding* or *false-name bidding*). Perhaps the biggest potential hurdle, however, is computational, and it is not specific to VCG.

Any efficient combinatorial auction protocol must solve a core problem: given the agents' individual declarations \hat{v} , it must determine the allocation of goods to agents that maximizes social welfare. That is, we must compute $\max_{x \in X} \sum_{i \in N} \hat{v}_i(x)$. In single-good auctions this was simple – we just had to satisfy the agent with the highest valuation. In combinatorial auctions, determining the winners is a more challenging computational problem.

Definition 7.29 (Winner determination problem (WDP)) *The winner determination problem (WDP) for a combinatorial auction, given the agents' declared valuations \hat{v} , is to find the social-welfare-maximizing allocation of goods to agents. This problem can be expressed as the following integer program.*

$$\text{maximize } \sum_{i \in N} \sum_{S \subseteq G} \hat{v}_i(S) x_{S,i} \quad (7.3)$$

$$\text{subject to } \sum_{S \ni j} \sum_{i \in N} x_{S,i} \leq 1 \quad \forall j \in G \quad (7.4)$$

$$\sum_{S \subseteq G} x_{S,i} \leq 1 \quad \forall i \in N \quad (7.5)$$

$$x_{S,i} = \{0, 1\} \quad \forall S \subseteq G, i \in N \quad (7.6)$$

In this integer programming formulation, the valuations $\hat{v}_i(S)$ are constants and the variables are $x_{S,i}$. These variables are Boolean, indicating whether bundle S is allocated to agent i . The objective function (7.3) states that we want to maximize the sum of the agents' declared valuations for the goods they are allocated. Constraint (7.4) ensures that no overlapping bundles of goods are allocated, and constraint (7.5) ensures that no agent receives more than one bundle. (This makes sense since bidders explicitly assign a valuation to *every* subset of the goods.) Finally, constraint (7.6) is what makes this an *integer program* rather than a linear program: no subset can be partially assigned to an agent.

The fact that the WDP is an integer program rather than a linear program is bad news, since only the latter are known to admit a polynomial-time solution. Indeed,

a reader familiar with algorithms and complexity may recognize the combinatorial auction allocation problem as a *set packing problem* (SPP). Unfortunately, it is well known that the SPP is NP-complete. This means that it is not likely that a polynomial-time algorithm exists for the problem. Worse, it so happens that this problem cannot even be approximated uniformly, meaning that there does not exist a polynomial-time algorithm and a fixed constant $k > 0$ such that for all inputs the algorithm returns a solution that is at least $\frac{1}{k}s^*$, where s^* is the value of the optimal solution for the given input.

There are two primary approaches to getting around the computational problem. First, we can restrict ourselves to a special class of problems for which there is guaranteed to exist a polynomial-time solution. Second, we can resort to heuristic methods that give up the guarantee of polynomial running time, optimality of solution, or both. In both cases, *relaxation methods* are a common approach. One instance of the first approach is to relax the integrality constraint, thereby transforming the problem into a linear program, which is solvable in polynomial time. In general the solution results in “fractional” allocations, in which fractions of goods are allocated to different bidders. If we are lucky, however, our solution to the LP will just happen to be integral; the broadest case when such luck is assured arises when the integer program’s constraint matrix is *totally unimodular*.

8 Conclusions

Mechanism design studies the design of protocols that achieve desired objectives even in the presence of self-interested agents. We say that a social choice function is implementable if it can be achieved in the equilibrium of some mechanism. The revelation principle demonstrates that we can restrict our attention to direct and truthful mechanisms without changing the set of implementable social choice functions. However, few social choice functions are implementable when agents are allowed general preferences. We thus consider the case of quasilinear preferences, in which agents’ preferences for money are additively separable from their preferences for the choice made by a mechanism. VCG is a particularly important mechanism for the quasilinear setting. It guarantees efficiency and dominant strategy truthfulness, and under additional assumptions also achieves weak budget balance and individual rationality.

Auctions are mechanisms for the allocation of scarce resources among a set of selfish agents. Various canonical auctions exist for the single-good setting. Second-price auctions offer dominant strategies (and, indeed, are a special case of VCG for this setting), while first-price auctions offer only Bayes–Nash equilibria. However, both auction types achieve the same revenue for the seller in equilibrium, under standard assumptions about agents’ valuations. Auctions can also be

used in more complex settings. Two important examples are position auctions, which are used to sell advertisements alongside search results on the Internet, and combinatorial auctions, which sell multiple, heterogeneous goods in the same auction, and allow bidders to specify valuations for arbitrary bundles of goods.

Mechanism design and auctions are covered to varying degrees in modern game theory textbooks, but even better are the microeconomic textbook of [16] and the excellent formal introduction to auction theory by [14]. More technical overviews from a computer science perspective are given in the introductory chapters of [25], in [24], and in our own textbook [29], on which this chapter is based. [12] is a large edited collection of many of the most important papers on the theory of auctions, preceded by a thorough survey by the editor; this survey is reproduced in [11]. Earlier surveys include [1], [33], and [17]. These texts cover most of the canonical single-good auction types we discuss in the chapter. Specific publications that underlie some of the results covered in this chapter are as follows.

The foundational idea of mechanisms as communication systems that select outcomes based on messages from agents is due to [8], who also elaborated the theory to include the idea that mechanisms should be “incentive compatible” [9]. The revelation principle was first articulated by [5] and was developed in the greatest generality by Myerson [19, 20, 21]. In 2007, Hurwicz and Myerson shared a Nobel Prize (along with Maskin, whose work we do not discuss in this chapter), “for having laid the foundations of mechanism design theory.” Theorem 7.2 is due to both Satterthwaite and Gibbard, in two separate publications [5, 28]. The VCG mechanism was anticipated by [31], who outlined an extension of the second-price auction to multiple identical goods. [7] explicitly considered the general family of truthful mechanisms applying to multiple distinct goods (though the result had appeared already in his 1969 Ph.D. dissertation). [2] proposed his tax for use with public goods (i.e., goods such as roads and national defense, which are paid for by all regardless of personal use). Theorem 7.4 is due to [6]; Theorem 7.7 is due to that paper as well as to the earlier [10]. The fact that Groves mechanisms are payoff equivalent to all other Bayes–Nash incentive-compatible efficient mechanisms was shown by [13] and [32]; the former reference [13] also gave the results that VCG is *ex interim* individually rational and that VCG collects the maximal amount of revenue among all *ex interim* individually rational Groves mechanisms. The Myerson–Satterthwaite theorem (7.8) appears in [22].

Vickrey’s seminal contribution [31] is still recommended reading for anyone interested in auctions. In it Vickrey introduced the second-price auction and argued that bidders in such an auction do best when they bid sincerely. He also provided the analysis of the first-price auction under the independent private value model with the uniform distribution described in this chapter. He even proved

an early version of the revenue-equivalence theorem (Theorem 7.11), namely that in the independent private value case, the English, Dutch, first-price, and second-price auctions all produce the same expected revenue for the seller. For his work, Vickrey received a Nobel Prize in 1996. The more general form of the revenue-equivalence theorem, Theorem 7.11, is due to [23] and [26], who also investigated optimal (i.e., revenue-maximizing) auctions. Our discussion of position auctions generally follows [4]; see also [30]. Combinatorial auctions are covered in depth in the edited collection [3], which probably provides the best single-source overview of the area. The computational complexity of the WDP is discussed in a chapter by [15]. Algorithms for the WDP have an involved history, and are reprised in chapters by [18] and [27].

9 Exercises

1. **Level 2** Consider a potentially infinite outcome space $\mathcal{O} \subset [0, 1]$, and a finite set N of n agents. Denote the utility of an agent with type θ_i for outcome o as $u_i(o, \theta_i)$. Constrain the utility functions so that every agent has some unique, most-preferred outcome $b(\theta_i) \in \mathcal{O}$, and so that $|o' - b(\theta_i)| < |o'' - b(\theta_i)|$ implies that $u_i(o', \theta_i) > u_i(o'', \theta_i)$. Consider a direct mechanism that asks every agent to declare its most-preferred outcome and then selects the median outcome. (If there are an even number of agents, the mechanism chooses the larger of the two middle outcomes.)
 - (a) Prove that truth-telling is a dominant strategy.
 - (b) Prove that the mechanism selects a Pareto optimal outcome.
 - (c) Prove that if the mechanism designer submits $n - 1$ “dummy preferences” with any values he or she likes, and then runs the same mechanism on the $2n - 1$ preferences, the dominant strategy is preserved.
 - (d) As described so far, the mechanism selects the $\lceil \frac{n}{2} \rceil^{\text{th}}$ -order statistic of the declared preferences. Explain how to select dummy preferences in such a way that the mechanism selects the k^{th} -order statistic of the agents’ declared preferences for any $k \in \{1, \dots, n\}$. Of course, the dummy preferences must be set in a way that does not depend on the specific declarations made by the agents.
2. **Level 1** Consider the following problem: a mechanism designer wants to know how likely it is that a given unfair coin will come up heads when it is next tossed. There is a psychic (agent 1) who knows the true probability p

of the coin coming up heads. The designer could just offer to pay the psychic a flat fee, but then the psychic's utility would be the same, regardless of the probability that the psychic reports, \hat{p} . In order to induce the psychic to reveal the true p , the mechanism designer commits to the following mechanism: if the coin comes up heads, then the psychic will be paid $c + \log_2 \hat{p}$, and if the coin comes up tails, then the psychic will be paid $c + \log_2(1 - \hat{p})$.

- (a) Prove that the psychic's utility is maximized by revealing the true p .
- (b) Now consider a scenario where p , again known by the psychic, represents the probability that a buyer (agent 2) has a high valuation (200 rather than 100). Because p represents the psychic's private information, we can understand it as the psychic's type. The seller knows that the joint type distribution is as follows:

p	v_2	Probability of this type profile
1/6	200	1/10
1/6	100	5/10
3/4	200	3/10
3/4	100	1/10

Design a deterministic mechanism with the following properties:

- i. truthful in Bayes–Nash equilibrium;
 - ii. the psychic's payment is consistent with the $\log_2 p$ rule above;
 - iii. in equilibrium the psychic and the buyer are guaranteed *ex post* utilities of at least one and zero respectively; and
 - iv. maximizes revenue, subject to satisfying (i–iii).
- (c) What mechanism would the seller choose to maximize revenue if the psychic wasn't available? Does having access to the psychic increase the seller's expected (total) revenue? If so, by how much?
 - (d) Given this setting, but relaxing requirements ii–iv, is it possible to create a mechanism that is truthful in dominant strategies (where the dominance can be weak, but not very weak)? Explain why or why not.
3. **Level 1** The VCG mechanism does not violate the Myerson-Satterthwaite theorem because it is not budget balanced for general quasilinear preferences. But this seems like an easy enough problem to solve – we can just evenly redistribute any money that was collected by the mechanism (or, tax all agents equally if the net payment to the agents was positive). Below is a proposed, budget-balanced version of the VCG mechanism.

The *budget-balanced VCG mechanism* is a direct mechanism $M(\hat{v}) = (x(\hat{v}), p_1(\hat{v}), \dots, p_n(\hat{v}))$, where

$$\begin{aligned} x(\hat{v}) &= \arg \max_{x \in X} \sum_{i \in N} \hat{v}_i(x), \text{ and} \\ t_i(\hat{v}) &= \max_{o \in O_{-i}} \sum_{j \neq i} \hat{v}_j(o) - \sum_{j \neq i} \hat{v}_j(x) \\ p_i(\hat{v}) &= t_i - \frac{1}{n} \sum_i t_i(\hat{v}). \end{aligned}$$

- (a) Show that this mechanism is not incentive compatible.
- (b) Although our first mechanism failed, we can use a similar idea to make VCG budget balanced *ex-ante*. Assume that bidders valuations v_i are randomly drawn from some joint commonly-known distribution.

Ex-ante budget-balanced VCG mechanism is a direct mechanism $M(\hat{v}) = (x(\hat{v}), p_1(\hat{v}), \dots, p_n(\hat{v}))$, where

$$\begin{aligned} x(\hat{v}) &= \arg \max_{x \in X} \sum_{i \in N} \hat{v}_i(x), \text{ and} \\ t_i(\hat{v}) &= \max_{o \in O_{-i}} \sum_{j \neq i} \hat{v}_j(o) - \sum_{j \neq i} \hat{v}_j(x(\hat{v})) \\ p_i(\hat{v}) &= t_i(\hat{v}) - \frac{1}{n} \sum_j \mathbb{E}_v[t_j(v)]. \end{aligned}$$

Prove that truth-telling is a dominant strategy in this new mechanism.

- (c) Show that this mechanism is *ex-ante* budget balanced.

4. **Level 2** Suppose you have some object that each of n agents desires, but which you do not value. Assume that each agent i values it at v_i , with v_i 's drawn independently and uniformly from some positive real line interval, say $[0, 10^{100}]$. Although you do not desire the object and also do not care about the actual values of the v_i 's, you need to compute $\sqrt{v_i}$ for each i .

Unfortunately, you face two problems. First, agents are not inclined to just reveal to you anything about their v_i 's. Second, your computer is costly to operate. It costs you 1 unit to determine the greater of two values, 2 units to perform any basic arithmetic operation (+, −, ×, /), and anything more complicated (such as \sqrt{x}) costs 20 units. The (accurate) current time of day can be observed without cost.

- (a) How much would it cost to compute $\sqrt{v_i}$ for each i using a straightforward VCG mechanism? (When computing cost, ignore the revenue that the auction will generate.) Hint: this part is very easy.
- (b) Your answer above gives an upper bound on the cost of computing the square roots of the valuations. Design an incentive-compatible, dominant-strategy (“strategyproof”) direct mechanism that will allow you to compute all $\sqrt{v_i}$ at *minimal* cost. Assume that agents can do computations for free.

- (c) In the previous part you were restricted to direct mechanisms. Show that an *indirect* mechanism can achieve even lower cost.
5. **Level 1** Consider a first-price auction with two bidders. Assume that they have IPV valuations drawn uniformly from the interval $[0, 10]$, and that they are risk-neutral. We saw that $s_1(v_1) = \frac{1}{2}v_1$ and $s_2(v_2) = \frac{1}{2}v_2$ together form a Bayes–Nash equilibrium for this game.
- (a) Assuming that bidder 2 is instead using the bidding strategy $s_2(v_2) = v_2$ (i.e., bidder 2 bids bidder 1's valuation), what is the best response bidding strategy $s_1(v_1)$ for bidder 1?
 - (b) Now consider instead a *second-price* auction. However, suppose the mechanism has a buggy implementation of max: most of the time the mechanism works correctly, but with some probability p that is strictly less than $1/3$, it awards the object to the second-highest bidder (instead of the highest bidder). In all cases it correctly calculates price as the second highest bid. Assuming that bidder 2 is still bidding truthfully, compute the best-response strategy for bidder 1. Is it still truthful?
6. **Level 1** Consider the following lobbying problem. There are n different companies, each of which wants the government to pass legislation that will benefit that company and will have no direct effect on the other companies. If the legislation that favors company i is passed, i 's profit will be v_i ; otherwise it will be 0. In order to try to influence government policy, each company i considers making a donation of some amount d_i to the government. Let's consider the case where all v_i are independent random variables distributed uniformly on $[0, 1]$. Somewhat cynically, the government will pass the single piece of legislation that benefits its biggest donor; of course, it will keep all the donations it receives.
- (a) Model this problem as an auction. State all the relevant assumptions that you make in building this model, and explain why they are reasonable.
 - (b) Find a symmetric Bayes–Nash equilibrium of this game. You may assume that for this game there exists a symmetric, pure-strategy equilibrium for which the bid amount is a monotonically increasing function of the agent's valuation.
7. **Level 2** We say that a good is *multiunit* if multiple, identical copies of the good are available, and these copies are valued identically by the agents. Say that a bidder i in the multiunit setting has *simple multiunit preferences*

if there are numbers v_i and d_i such that the bidder values d_i or more units of the good at v_i , and any smaller number of units at 0. Say that i has *known simple multiunit preferences* if d_i is common knowledge. Consider mechanisms running in the multiunit setting where each agent has quasilinear, known simple multiunit preferences. In this setting, a direct mechanism is one where each agent i declares a valuation \hat{v}_i for its bundle size, and the mechanism chooses an allocation of its k units among the agents and payments that each agent must make.

An allocation rule is *monotone* if for any agent i and any profile \hat{v}_{-i} of declarations of the other agents, there exists a *critical value* $\kappa_i(\hat{v}_{-i}) \in (\mathbb{R} \cup \{\infty\})$ such that

- (1) any declaration $\hat{v}_i > \kappa_i(\hat{v}_{-i})$ is a winning declaration (i.e., causes i to be allocated its desired number of units), and
- (2) any declaration $\hat{v}_i < \kappa_i(\hat{v}_{-i})$ is a losing declaration (i.e., causes i not to be allocated its desired bundle).

An auction is *normalized* if $p_i \geq 0$ for every losing agent i .

Prove the following statement: Let $M = (\chi, p)$ be a direct, normalized, individually rational auction in this setting, where χ is the allocation rule and p is the payment rule. Then M is truthful if and only if

- (1) χ is monotone, and
- (2) each winning bidder pays its critical value, and each losing agent pays 0. That is, $p_i(\hat{v}) = \begin{cases} \kappa_i(\hat{v}_{-i}) & \text{if } i \text{ is allocated } d_i \text{ units,} \\ 0 & \text{otherwise.} \end{cases}$

8. **Level 2** Consider the following combinatorial auction scenario: a professor wants to supplement his or her income by auctioning off advertising space on his or her lecture slides. The professor decides to use VCG as an auction mechanism and to offer two goods: a top banner space and a sidebar space. Every advertiser's type has two parts: v_i , which specifies its utility for an allocation that satisfies it; and $f_i \in \{t, s, b\}$, which specifies what must happen for it to be satisfied. An agent with $f_i = t$ will be satisfied if it wins the top banner space, regardless of what happens to the sidebar space. An agent with $f_i = s$ will be satisfied if it wins the sidebar space, regardless of what happens to the top banner space. An agent with $f_i = b$ will only be satisfied if it wins both spaces. (It might only use one space, but it insists on not having any other ad shown alongside its ad.)

- (a) Let v_{1s} denote the first-highest bid for slot s . (Similarly define v_{1t} , v_{1b} , v_{2s} , v_{2t} and v_{2b}). Express the VCG revenue as a function of these values. Assume that the values are zero if there is no corresponding bidder. Assume that the auctioneer breaks ties in favor of bidders who want both slots.
 - (b) Demonstrate (with a specific example) that VCG's revenue can decrease as these quantities increase.
 - (c) Consider a setting where one of the agents is capable of creating a second identity (at some very small cost α), and submitting bids using both identities. (If it does so it gets every good won by either identity, and must pay for both.) Demonstrate that VCG is no longer dominant-strategy truthful in this case.
 - (d) Consider the following auction mechanism: allocate both goods to whichever advertiser has submitted the highest bid, and charge him or her the amount of the second highest bid (ignoring the f_i element of their types). Prove that this mechanism has the following properties:
 - i. the agents have a dominant strategy of reporting truthfully, using only their true identities, and
 - ii. in equilibrium, the mechanism generates at least half as much revenue as VCG would if the agents submitted truthful reports.
9. **Level 3** Using data from an auction website (e.g., eBay), estimate bidders' valuation distributions, using (e.g.) kernel density estimation. Do this for several dissimilar kinds of goods. How different are the maximum likelihood estimates of the valuation distributions across these auctions? How much social welfare would be lost if bidders played the equilibrium strategy from one auction setting in another?
10. **Level 4** Mechanism design and auction theory are based on a perfect-rationality model of agent behavior. Investigate how the theory is impacted by a more realistic model of behavior. (E.g., revenue equivalence holds only under a set of given assumptions. Consider which auction design a seller should choose if agents are not perfectly rational.)

References

- [1] R. Cassady. *Auctions and Auctioneering*. University of California Press, 1967.
- [2] E.H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

- [3] P.C. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, Cambridge, MA, 2006.
- [4] B. Edelman, M. Schwarz, and M. Ostrovsky. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, March 2007.
- [5] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41:587–601, 1973.
- [6] J. Green and J.J. Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45(2):427–438, 1977.
- [7] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–31, July 1973.
- [8] L. Hurwicz. Optimality and informational efficiency in resource allocation processes. In K.J. Arrow, S. Karlin, and P. Suppes, editors, *Mathematical Methods in the Social Sciences*, pages 27–46. Stanford University Press, Stanford, CA, 1960.
- [9] L. Hurwicz. On informationally decentralized systems. In C.B. McGuire and R. Radner, editors, *Decision and Organization*, pages 297–336. North-Holland, London, 1972.
- [10] L. Hurwicz. On the existence of allocation systems whose manipulative Nash equilibria are Pareto optimal. Unpublished, 1975.
- [11] Paul Klemperer. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–286, July 1999.
- [12] Paul Klemperer, editor. *The Economic Theory of Auctions*. Edward Elgar, 1999.
- [13] V. Krishna and M. Perry. Efficient Mechanism Design. Technical report, Pennsylvania State University, 1998.
- [14] Vijay Krishna. *Auction Theory*. Elsevier Science, New York, 2002.
- [15] D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In Cramton et al. [3], chapter 12, pages 297–318.
- [16] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, Oxford, 1995.
- [17] R. McAfee and J. MacMillan. Auctions and bidding. *Journal of Economic Literature*, 25(3):699–738, 1987.
- [18] R. Müller. Tractable cases of the winner determination problem. In Cramton et al. [3], chapter 13, pages 319–336.

- [19] R. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47(1):61–73, 1979.
- [20] R. Myerson. Optimal coordination mechanisms in generalized principal-agent models. *Journal of Mathematical Economics*, 11:67–81, 1982.
- [21] R. Myerson. Multistage games with communication. *Econometrica*, 54(2):323–358, 1986.
- [22] R. Myerson and M. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29(2):265–281, 1983.
- [23] R.B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.
- [24] N. Nisan. Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 9, pages 209–242. Cambridge University Press, Cambridge, UK, 2007.
- [25] D. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, May 2001.
- [26] J.G. Riley and W.F. Samuelson. Optimal auctions. *The American Economic Review*, 71(3):381–392, 1981.
- [27] T. Sandholm. Optimal winner determination algorithms. In Cramton et al. [3], chapter 14, pages 337–368.
- [28] M.A. Satterthwaite. Strategy-proofness and Arrow’s conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.
- [29] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.
- [30] H.R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, 2007.
- [31] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.
- [32] S.R. Williams. A characterization of efficient, Bayesian incentive compatible mechanisms. *Economic Theory*, 14(1):155–180, 1999.
- [33] Robert Wilson. Auction theory. In J. Eatwell, M. Milgate, and P. Newman, editors, *The New Palgrave Dictionary of Economics*, volume I. Macmillan, London, 1987.

Chapter 8

Computational Coalition Formation

Edith Elkind, Talal Rahwan,* and Nicholas R. Jennings*

1 Introduction

In many multiagent systems, agents can improve their performance by forming *coalitions*, i.e., pooling their efforts and resources so as to achieve the tasks at hand in a more efficient way. This holds both for *cooperative* agents, i.e., agents who share a common set of goals, and for *selfish* agents who only care about their own payoffs. For cooperative agents, to find the optimal collaboration pattern, it suffices to identify the best way of splitting agents into teams. In contrast, when the agents are selfish, we also have to specify how to distribute the gains from cooperation, since each agent needs to be incentivized to participate in the proposed solution.

In this chapter, we discuss coalition formation in multiagent systems for both selfish and cooperative agents. To deal with selfish agents, we introduce classic solution concepts of coalitional game theory that capture the notions of stability and fairness in coalition formation settings. We then give an overview of existing representation formalisms for coalitional games. For each such formalism, we discuss the complexity of computing the solution concepts defined earlier in the chapter, focusing on algorithms whose running time is polynomial in the number of agents n . In the second half of the chapter, we focus on practical approaches for finding an optimal partition of agents into teams. We present the state-of-the-art algorithms for this problem, and compare their relative strengths and weaknesses.

*The first two authors have contributed equally to the chapter.

1.1 Coalitional Games: A Bird's Eye View

The goal of the coalition formation process is to split the set of agents – or *players* – into disjoint teams, or *coalitions*: a partition of the set of agents into coalitions is called a *coalition structure*.¹ Once a coalition structure forms, each coalition chooses its action in a way that results in payoffs to its members. Coalitional games provide a formal model of coalition formation scenarios. They are usually classified according to two orthogonal dimensions: (1) whether agents can make payments to each other and (2) whether the payoff that a coalition obtains by choosing a particular action depends on the actions of other coalitions. We will now discuss this classification in more detail.

In some settings modeled by coalitional games, all agents have comparable utilities and can commit to monetary transfers among the members of a coalition. Whenever this is the case, we can simply assume that the coalitional action generates a single payoff, which is subsequently shared among the members of the coalition; this payoff is referred to as the *value* of this coalition. Such games are known as *transferable utility games*, or *TU games*. However, sometimes agents cannot make side payments to each other, either because their payoffs from the coalitional action are non-monetary in nature, or because there is no suitable infrastructure to transfer the money.

Example 8.1 *If several researchers from different universities write a joint paper, each researcher receives a payoff from its own university: the paper can count toward promotion or tenure, receive an internal prize, or, sometimes, be rewarded with a monetary bonus. However, these payoffs are allocated to individual researchers, and, with the exception of a bonus payment, cannot be transferred from one researcher to another.*

The settings similar to the one in Example 8.1 are modeled by assuming that each coalitional action corresponds to a vector of payoffs – one for each member of the coalition. Games represented in this manner are known as *games with non-transferable utility*, or *NTU games*.

It is important to note that in NTU settings two coalitional actions may be incomparable. For instance, consider the 2-player coalition $\{a_1, a_2\}$ that chooses between actions x and y . Suppose that whenever the players choose x , player a_1 gets a payoff of 5, whereas player a_2 gets a payoff of 1; on the other hand, if players choose y , player a_1 gets 2 and player a_2 gets 7. Obviously, player a_1 prefers x to y , even though action y has a higher total utility, whereas player a_2 prefers y to x . In contrast, in TU games, all players prefer the action(s) that result(s) in the highest sum of payoffs, as they can distribute the total payoff so

¹Recently, games with overlapping coalitions have also been considered; see, e.g. [12].

that everyone is better off. This intracoalitional competition makes NTU games more difficult to analyze, which may explain why TU games received much more attention in the multiagent literature. We will follow this trend, and for the rest of the chapter focus on TU games only.

Now, in each of the examples considered so far, the payoffs that each coalition could attain were determined by the identities and actions of the coalition members. However, there are cases where a coalition's productivity also depends on the coalition structure that it is a part of, i.e., it may be influenced by the actions of non-members. This is the case, for instance, in market-like environments, where each coalition provides a service, and the payment it can charge for its service depends on the competition it faces. While this phenomenon can be observed both in TU and in NTU settings, traditionally, it has been studied in the transferable utility model only. Transferable utility games where the value of each coalition may depend on the coalition structure it appears in are known as *partition function games* [37]. On the other hand, games where the value of each coalition is the same in every coalition structure are known as *characteristic function games*. Clearly, characteristic function games form a proper subclass of partition function games, and tend to be much easier to work with. Thus, from now on, we will further restrict our attention to characteristic function games.

2 Definitions

In this section, we will formally define characteristic function games as well as several important subclasses of these games.

Definition 8.1 A characteristic function game G is given by a pair (A, v) , where $A = \{a_1, \dots, a_n\}$ is a finite set of players, or agents, and $v : 2^A \rightarrow \mathbb{R}$ is a characteristic function, which maps each subset, or coalition, of agents C to a real number $v(C)$. This number is referred to as the value of the coalition C .

We remark that we can represent a characteristic function game by explicitly listing all coalitions together with their values; the size of this naive representation is exponential in n . However, in practice we are usually interested in games that admit a succinct representation and can be analyzed in time polynomial in n . A number of such representations have been considered in the literature; we will discuss some of them in Section 4.

We will now present two examples of characteristic function games.

Example 8.2 Charlie (C), Marcie (M), and Pattie (P) want to pool their savings to buy ice cream. Charlie has c dollars, Marcie has m dollars, Pattie has p dollars, and the ice cream packs come in three different sizes: (1) 500g which costs \$7,

(2) 750g which costs \$9, and (3) 1000g which costs \$11. The children value ice cream, and assign no utility to money. Thus, the value of each coalition is determined by how much ice cream it can buy.

This situation corresponds to a characteristic function game with the set of players $A = \{C, M, P\}$. For $c = 3$, $m = 4$, $p = 5$, its characteristic function v is given by $v(\emptyset) = 0$, $v(\{C\}) = v(\{M\}) = v(\{P\}) = 0$, $v(\{C, M\}) = v(\{C, P\}) = 500$, $v(\{M, P\}) = 750$, $v(\{C, M, P\}) = 1000$. For $c = 8$, $m = 8$, $p = 1$, its characteristic function v is given by $v(\emptyset) = 0$, $v(\{C\}) = v(\{M\}) = 500$, $v(\{P\}) = 0$, $v(\{C, P\}) = v(\{M, P\}) = 750$, $v(\{C, M\}) = 1250$, $v(\{C, M, P\}) = 1250$.

Example 8.3 A fictional country X has a 101-member parliament, where each representative belongs to one of the four parties: Liberal (L), Moderate (M), Conservative (C), or Green (G). The Liberal party has 40 representatives, the Moderate party has 22 representatives, the Conservative party has 30 representatives, and the Green party has 9 representatives. The parliament needs to decide how to allocate \$1 billion of discretionary spending, and each party has its own preferred way of using this money. The decision is made by a simple majority vote, and we assume that all representatives vote along the party lines. Parties can form coalitions; a coalition has value \$1 billion if it can win the budget vote no matter what the other parties do, and value 0 otherwise.

This situation can be modeled as a 4-player characteristic function game, where the set of players in $A = \{L, M, C, G\}$ and the characteristic function v is given by

$$v(C) = \begin{cases} 0 & \text{if } |C| \leq 1, \text{ or } |C| = 2 \text{ and } G \in C \\ 10^9 & \text{otherwise.} \end{cases}$$

It is usually assumed that the value of the empty coalition \emptyset is 0, i.e., $v(\emptyset) = 0$. Moreover, it is often the case that the value of each coalition is non-negative (i.e., agents form coalitions to make a profit), or else that the value of each coalition is non-positive (i.e., agents form coalitions to share costs). Throughout this chapter, we will mostly focus on the former scenario, i.e., we assume that $v(C) \geq 0$ for all $C \subseteq A$. However, all our definitions and results can be easily adapted to the latter scenario.

2.1 Outcomes

An outcome of a characteristic function game consists of two parts: (1) a partition of players into coalitions, and (2) a payoff vector, which distributes the value of each coalition among its members.

Formally, a *coalition structure* over A is a collection of non-empty coalitions $CS = \{C_1, \dots, C_{|CS|}\}$ such that

- $\bigcup_{j=1}^{|CS|} C_j = A$, and
- $C_i \cap C_j = \emptyset$ for any $i, j = 1, \dots, |CS|$ such that $i \neq j$.

We will denote the space of all coalition structures over A by \mathcal{P}^A . Also, given a coalition structure $CS = \{C_1, \dots, C_{|CS|}\} \in \mathcal{P}^A$, we will say that a vector $\mathbf{x} = (x_1, \dots, x_n)$ is a *payoff vector* for CS , where x_i specifies the payoff of a_i in CS , if

- $x_i \geq 0$ for all $i = 1, \dots, n$, and
- $\sum_{i: a_i \in C_j} x_i = v(C_j)$ for any $j = 1, \dots, |CS|$.

Definition 8.2 Given a characteristic function game $G = (A, v)$, an outcome of G is a pair (CS, \mathbf{x}) , where $CS \in \mathcal{P}^A$ and \mathbf{x} is a payoff vector for CS .

A payoff vector \mathbf{x} for a coalition structure $CS \in \mathcal{P}^A$ is said to be an *imputation* if it satisfies the *individual rationality* condition, i.e., $x_i \geq v(\{a_i\})$ for each $a_i \in A$. If a payoff vector is an imputation, each player weakly prefers being in the coalition structure to being on its own. Now, of course, players may still find it profitable to deviate *as a group*; we will discuss the issue of stability against group deviations in Section 3. However, before we do that, let us consider a few important classes of characteristic function games, and discuss the relationship among them.

2.2 Subclasses of Characteristic Function Games

We will now define four important subclasses of coalitional games: monotone games, superadditive games, convex games, and simple games.

2.2.1 Monotone Games

Usually, adding an agent to an existing coalition can only increase the overall productivity of this coalition; games with this property are called *monotone games*.

Definition 8.3 A characteristic function game $G = (A, v)$ is said to be *monotone* if $v(C') \leq v(C'')$ for every pair of coalitions $C', C'' \subseteq A$ such that $C' \subseteq C''$.

2.2.2 Superadditive Games

A stronger property, which is also enjoyed by many practically useful games, is *superadditivity*: in a *superadditive game*, it is always profitable for two groups of players to join forces.

Definition 8.4 A characteristic function game $G = (A, v)$ is said to be *superadditive* if $v(C' \cup C'') \geq v(C') + v(C'')$ for every pair of disjoint coalitions $C', C'' \subseteq A$.

Since we have assumed that the value of each coalition is non-negative, superadditivity implies monotonicity: if a game $G = (A, v)$ is superadditive, and $C' \subseteq C''$, then $v(C') \leq v(C'') - v(C'' \setminus C') \leq v(C'')$. However, the converse is not necessarily true: consider, for instance, a game where the value of the characteristic function grows logarithmically with the coalition size, i.e., $v(C') = \log |C'|$.

In superadditive games, there is no compelling reason for agents to form a coalition structure consisting of multiple coalitions: the agents can earn at least as much profit by forming the *grand coalition*, i.e., the coalition that contains all agents. Therefore, for superadditive games it is usually assumed that the agents form the grand coalition, i.e., the outcome of a superadditive game is of the form $(\{A\}, \mathbf{x})$ where \mathbf{x} satisfies $\sum_{i=1}^n x_i = v(A)$. Conventionally, $\{A\}$ is omitted from the notation, i.e., an outcome of a superadditive game is identified with a payoff vector for the grand coalition.

2.2.3 Convex Games

The superadditivity property places a restriction on the behavior of the characteristic function v on disjoint coalitions. By placing a similar restriction on v 's behavior on non-disjoint coalitions, we obtain the class of *convex games*.

Definition 8.5 A characteristic function game $G = (A, v)$ is said to be convex if $v(C \cup C') + v(C \cap C') \geq v(C) + v(C')$ for every pair of coalitions $C, C' \subseteq A$.

Convex games have a very intuitive characterization in terms of players' marginal contributions: in a convex game, a player is more useful when it joins a bigger coalition.

Proposition 8.1 A characteristic function game $G = (A, v)$ is convex if and only if for every pair of coalitions C', C'' such that $C' \subset C''$ and every player $a_i \in A \setminus C''$ it holds that $v(C'' \cup \{a_i\}) - v(C'') \geq v(C' \cup \{a_i\}) - v(C')$.

Proof. For the “only if” direction, assume that $G = (A, v)$ is convex, and consider two coalitions C', C'' such that $C' \subset C'' \subset A$ and a player $a_i \in A \setminus C''$. By setting $X = C''$, $Y = C' \cup \{a_i\}$, we obtain

$$v(C'' \cup \{a_i\}) - v(C'') = v(X \cup Y) - v(X) \geq v(Y) - v(X \cap Y) = v(C' \cup \{a_i\}) - v(C'),$$

which is exactly what we need to prove.

The “if” direction can be proved by induction on the size of $X \setminus Y$; we leave the proof as an exercise for the reader. ■

Any convex game is necessarily superadditive: if a game $G = (A, v)$ is convex, and C' and C'' are two disjoint subsets of A , then we have $v(C' \cup C'') \geq v(C') +$

$v(C'') - v(C' \cap C'') = v(C') + v(C'')$ (here we use our assumption that $v(\emptyset) = 0$). To see that the converse is not always true, consider a game $G = (A, v)$, where $A = \{a_1, a_2, a_3\}$, and $v(C) = 1$ if $|C| \geq 2$ and $v(C) = 0$ otherwise. It is easy to check that this game is superadditive. On the other hand, for $C' = \{a_1, a_2\}$ and $C'' = \{a_2, a_3\}$, we have $v(C') = v(C'') = 1$, $v(C' \cup C'') = 1$, $v(C' \cap C'') = 0$.

2.2.4 Simple Games

Another well-studied class of coalitional games is that of simple games: a game $G = (A, v)$ is said to be a *simple game* if it is monotone and the characteristic function only takes values 0 and 1, i.e., $v(C) \in \{0, 1\}$ for every $C \subseteq A$. For instance, the game in Example 8.3 becomes a simple game if we rescale the payoffs so that they become 0 and 1 (instead of 0 and 10^9). In a simple game, coalitions of value 1 are said to be *winning*, and coalitions of value 0 are said to be *losing*. Such games model situations where there is a task to be completed: a coalition is labeled as winning if and only if it can complete the task.

Note that simple games are superadditive if and only if the complement of each winning coalition is losing. Clearly, there exist simple games that are not superadditive. Nevertheless, it is usually assumed that the outcome of a simple game is a payoff vector for the grand coalition, just as in superadditive games.

3 Solution Concepts

Any partition of agents into coalitions and any payoff vector that respects this partition correspond to an outcome of a characteristic function game. However, not all outcomes are equally desirable. For instance, if all agents contribute equally to the value of a coalition, a payoff vector that allocates the entire payoff to one of the agents is less appealing than the one that shares the profits equally among all agents. Similarly, an outcome that incentivizes all agents to work together is preferable to an outcome that some of the agents want to deviate from.

More broadly, one can evaluate the outcomes according to two sets of criteria: (1) *fairness*, i.e., how well each agent's payoff reflects its contribution, and (2) *stability*, i.e., what the incentives are for the agents to stay in the coalition structure. These two sets of criteria give rise to two families of payoff division schemes, or *solution concepts*. We will now discuss each of them in turn.

3.1 Shapley Value

The best-known solution concept that aims to capture the notion of fairness in characteristic function games is the *Shapley value* [64]. The Shapley value is

usually defined for superadditive games. As argued above, for such games an outcome can be identified with a payoff vector for the grand coalition, i.e., the Shapley value prescribes how to share the value of the grand coalition in a fair way.

To present the formal definition of the Shapley value, we need some additional notation. Given a characteristic function game $G = (A, v)$, let Π^A denote the set of all *permutations* of A , i.e., one-to-one mappings from A to itself. Given a permutation $\pi \in \Pi^A$, we denote by $C_\pi(a_i)$ the coalition that consists of all predecessors of a_i in π , i.e., we set $C_\pi(a_i) = \{a_j \in A \mid \pi(a_j) < \pi(a_i)\}$. The *marginal contribution* of an agent a_i with respect to a permutation π in a game $G = (A, v)$ is denoted by $\Delta_\pi^G(a_i)$ and is given by

$$\Delta_\pi^G(a_i) = v(C_\pi(a_i) \cup \{a_i\}) - v(C_\pi(a_i));$$

this quantity measures by how much a_i increases the value of the coalition consisting of its predecessors in π when it joins them. Informally, the Shapley value of a player a_i is its average marginal contribution, where the average is taken over all permutations of A . More formally, we have the following definition.

Definition 8.6 *Given a characteristic function game $G = (A, v)$, the Shapley value of a player $a_i \in A$ is denoted by $\varphi_i(G)$ and is given by*

$$\varphi_i(G) = \frac{1}{n!} \sum_{\pi \in \Pi^A} \Delta_\pi^G(a_i).$$

The Shapley value has many attractive properties. In what follows, we list four of them; the proofs of Propositions 8.2–8.5 are left as an exercise for the reader.

First, the Shapley value is *efficient*, i.e., it distributes the value of the grand coalition among all agents.

Proposition 8.2 *For any characteristic function game $G = (A, v)$, we have $\sum_{i=1}^n \varphi_i(G) = v(A)$.*

Second, the Shapley value does not allocate any payoffs to players who do not contribute to any coalition. Formally, given a characteristic function game $G = (A, v)$, a player $a_i \in A$ is said to be a *dummy* if $v(C) = v(C \cup \{a_i\})$ for every $C \subseteq A$. It is not hard to see that the Shapley value of a dummy player is 0.

Proposition 8.3 *If a player $a_i \in A$ is a dummy in a characteristic function game G , then $\varphi_i(G) = 0$.*

Third, if two players contribute equally to each coalition, then their Shapley values are equal. Formally, given a characteristic function game $G = (A, v)$, we

say that players a_i and a_j are *symmetric* in G if $v(C \cup \{a_i\}) = v(C \cup \{a_j\})$ for every coalition $C \subseteq A \setminus \{a_i, a_j\}$. It turns out that symmetric players have equal Shapley values.

Proposition 8.4 *If players a_i and a_j are symmetric in a characteristic function game G , then $\phi_i(G) = \phi_j(G)$.*

Finally, consider a group of players A that is involved in two coalitional games G' and G'' , i.e., $G' = (A, v')$, $G'' = (A, v'')$. The *sum* of G' and G'' is a coalitional game $G^+ = G' + G''$ given by $G^+ = (A, v^+)$, where for every coalition $C \subseteq A$ we have $v^+(C) = v'(C) + v''(C)$. It can easily be seen that the Shapley value of a player a_i in G^+ is the sum of its Shapley values in G' and G'' .

Proposition 8.5 *Consider two characteristic function games $G' = (A, v')$ and $G'' = (A, v'')$ over the same set of players A . Then for any player $a_i \in A$ we have $\phi_i(G' + G'') = \phi_i(G') + \phi_i(G'')$.*

To summarize, we have argued that the Shapley value possesses four desirable properties:

- (1) *Efficiency*: all the profit earned by the agents in the grand coalitions is distributed among them;
- (2) *Null player*: players with zero marginal contributions to all coalitions receive zero payoff;
- (3) *Symmetry*: all players that have the same marginal contribution to all coalitions receive the same payoff;
- (4) *Additivity*: $\phi_i(G' + G'') = \phi_i(G') + \phi_i(G'')$ for all $a_i \in A$.

Interestingly, the Shapley value is the only payoff division scheme that has these four properties simultaneously [64]. In other words, if we view properties (1)–(4) as axioms, then these axioms characterize the Shapley value.

3.2 Banzhaf Index

Another solution concept that is motivated by fairness considerations is the *Banzhaf index* [7]. The difference between the Shapley value and the Banzhaf index can be described in terms of the underlying coalition formation model: the Shapley value measures the agent's expected marginal contribution if agents join the coalition one by one in a random order, whereas the Banzhaf index measures the agent's expected marginal contribution if each agent decides whether to join the coalition independently with probability $1/2$. This intuition is formally captured by the following definition.

Definition 8.7 Given a characteristic function game $G = (A, v)$, the Banzhaf index of a player $i \in A$ is denoted by $\beta_i(G)$ and is given by

$$\beta_i(G) = \frac{1}{2^{n-1}} \sum_{C \subseteq A \setminus \{a_i\}} [v(C \cup \{a_i\}) - v(C)].$$

It is not hard to verify that the Banzhaf index satisfies properties (2), (3), and (4) in the list above. However, it does not satisfy property (1), i.e., efficiency.

Example 8.4 Consider a characteristic function game $G = (A, v)$, where $v(A) = 1$ and $v(C) = 0$ for every $C \subset A$. We have $\phi_i(G) = \frac{1}{n}$, $\beta_i(G) = \frac{1}{2^{n-1}}$ for each $a_i \in A$.

Since efficiency is a very desirable property of a payoff distribution scheme, some researchers also consider the *normalized Banzhaf index* $\eta_i(G)$, which is defined as

$$\eta_i(G) = \frac{\beta_i(G)}{\sum_{i \in A} \beta_i(G)}.$$

While this version of the Banzhaf index satisfies efficiency, it loses the additivity property.

3.3 Core

We have introduced two solution concepts that attempt to measure the agents' marginal contribution. In contrast, the solution concepts considered in this and subsequent sections are defined in terms of coalitional stability.

Consider a characteristic function game $G = (A, v)$ and an outcome (CS, \mathbf{x}) of this game. Let $x(C)$ denote the total payoff of a coalition C under a payoff vector \mathbf{x} , i.e., $x(C) = \sum_{i: a_i \in C} x_i$. Now, if $x(C) < v(C)$, then the agents in C have an incentive to deviate since they could do better by abandoning CS and forming a coalition of their own. For example, if the agents were to share the extra profit equally among themselves, every agent $a_i \in C$ would receive a payoff of $x_i + \frac{v(C) - x(C)}{|C|}$ instead of x_i . An outcome where no subset of agents has an incentive to deviate is called *stable*, and the set of all such outcomes is called the *core* of G [29].

Definition 8.8 The core of a characteristic function game $G = (A, v)$ is the set of all outcomes (CS, \mathbf{x}) such that $x(C) \geq v(C)$ for any $C \subseteq A$.

In a superadditive game, the outcomes are payoff vectors for the grand coalition, so for such games the core can be defined as the set of all vectors \mathbf{x} that satisfy: (1) $x_i \geq 0$ for all $a_i \in A$, (2) $x(A) = v(A)$, and (3) $x(C) \geq v(C)$ for all $C \subseteq A$.

The outcomes in the core are stable and therefore they are more likely to arise when a coalitional game is played. However, some games have empty cores.

Example 8.5 Consider the game $G = (A, v)$, where $A = \{a_1, a_2, a_3\}$, $v(C) = 1$ if $|C| \geq 2$ and $v(C) = 0$ otherwise. We claim that this game has an empty core. Indeed, suppose that the core of G is non-empty. Since G is superadditive, its core contains a vector $\mathbf{x} = (x_1, x_2, x_3)$, where $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, and $x_1 + x_2 + x_3 = 1$. The latter constraint implies that $x_i \geq \frac{1}{3}$ for some $a_i \in A$. But then for $C = A \setminus \{a_i\}$ we have $v(C) = 1$, $x(C) \leq 2/3$, which means that (x_1, x_2, x_3) is not in the core. This contradiction shows that the core of G is empty.

Observe that the set of all outcomes in the core of a superadditive game can be characterized by the following linear feasibility program (LFP):

$$\begin{aligned} x_i &\geq 0 \quad \text{for each } a_i \in A \\ \sum_{i: a_i \in A} x_i &= v(A) \\ \sum_{i: a_i \in C} x_i &\geq v(C) \quad \text{for each } C \subseteq A \end{aligned} \tag{8.1}$$

This LFP has $2^n + n + 1$ constraints. Therefore, if we want to convert it into an algorithm for checking non-emptiness of the core which runs in time polynomial in n , we need an efficient *separation oracle* for this LFP. Recall that a separation oracle for a linear (feasibility) program is a procedure that, given a candidate solution (x_1, \dots, x_n) , determines whether it is feasible, and, if not, outputs the violated constraint. It is well-known that if a linear program over n variables admits a separation oracle that runs in time $\text{poly}(n)$, then an optimal feasible solution can be found in time $\text{poly}(n)$ [62].

Now, the first $n + 1$ constraints in our LFP are straightforward to check. Therefore, the problem of checking non-emptiness of the core for superadditive games can be reduced to checking whether a candidate solution satisfies the last 2^n constraints, i.e., verifying whether a given outcome is in the core (and, if not, computing the coalition that has an incentive to deviate). In general, checking whether a given outcome is in the core and/or deciding whether the core is non-empty is not easy: in Section 4, we will see examples of classes of coalitional games for which these problems are NP-hard. However, we will now see that for some of the classes of games discussed in Section 2.2, these problems are efficiently solvable.

3.3.1 The Core of Simple Games

Recall that for simple games it is standard to assume that the grand coalition forms, even if the game is not superadditive. Under this assumption, it is easy to characterize the outcomes in the core, and provide a simple criterion for checking whether the game has a non-empty core.

A player a_i in a simple game $G = (A, v)$ is said to be a *veto player* if $v(C) = 0$ for any $C \subseteq A \setminus \{a_i\}$; since simple games are monotone, this is equivalent to requiring that $v(A \setminus \{a_i\}) = 0$. Observe that a game may have more than one veto player: for instance, in the unanimity game, where $v(A) = 1$, $v(C) = 0$ for any $C \subset A$, all players are veto players. We will now show that the only way to achieve stability is to share the payoff among the veto players, if they exist.

Theorem 8.1 *A simple game $G = (A, v)$ has a non-empty core if and only if it has a veto player. Moreover, an outcome (x_1, \dots, x_n) is in the core of G if and only if $x_i = 0$ for any player a_i who is not a veto player in G .*

Proof. Suppose G has a veto player a_i . Then the outcome \mathbf{x} with $x_i = 1$, $x_j = 0$ for $j \neq i$ is in the core: any coalition C that contains a_i satisfies $x(C) = 1 \geq v(C)$, whereas any coalition C' that does not contain a_i satisfies $v(C') = 0 \leq x(C')$.

Conversely, suppose that G does not have a veto player. Suppose for the sake of contradiction that G has a non-empty core, and let \mathbf{x} be an outcome in the core of G . Since $x(A) = 1$, we have $x_i > 0$ for some $a_i \in A$, and hence $x(A \setminus \{a_i\}) = 1 - x_i < 1$. However, since a_i is not a veto player, we have $v(A \setminus \{a_i\}) = 1 > x(A \setminus \{a_i\})$, a contradiction with \mathbf{x} being in the core.

The second statement of the theorem can be proved similarly. ■

The characterization of the outcomes in the core provided by Theorem 8.1 suggests a simple algorithm for checking if an outcome is in the core or deciding non-emptiness of the core: it suffices to determine, for each player a_i , whether it is a veto player, i.e., to compute $v(A \setminus \{a_i\})$. Thus, if the characteristic function of a simple game is efficiently computable, we can answer the core-related questions in polynomial time.

We remark that if the simple game is not superadditive, and we use the more general definition of an outcome, i.e., allow the players to form coalition structures, Theorem 8.1 no longer holds. Moreover, deciding whether an outcome is in the core becomes computationally hard even for fairly simple representation formalisms (see Section 4.1).

3.3.2 The Core of Convex Games

Convex games always have a non-empty core. We will now present a constructive proof of this fact, i.e., show how to obtain an outcome in the core of a convex game.

Theorem 8.2 *If $G = (A, v)$ is a convex game, then G has a non-empty core.*

Proof. Fix an arbitrary permutation $\pi \in \Pi^A$, and set $x_i = \Delta_\pi^G(a_i)$. We claim that (x_1, \dots, x_n) is in the core of G .

Indeed, observe first that any convex game is monotone, so $x_i \geq 0$ for all $a_i \in A$. Moreover, we have $\sum_{i=1}^n x_i = \Delta_\pi^G(a_1) + \dots + \Delta_\pi^G(a_n) = v(A)$. Finally, suppose for the sake of contradiction that we have $v(C) > x(C)$ for some coalition $C = \{a_{i_1}, \dots, a_{i_s}\}$. We can assume without loss of generality that $\pi(a_{i_1}) \leq \dots \leq \pi(a_{i_s})$, i.e., the members of C appear in π ordered as a_{i_1}, \dots, a_{i_s} . We can write $v(C)$ as

$$v(C) = v(\{a_{i_1}\}) - v(\emptyset) + v(\{a_{i_1}, a_{i_2}\}) - v(\{a_{i_1}\}) + \dots + v(C) - v(C \setminus \{a_{i_s}\}).$$

Now, for each $j = 1, \dots, s$, the supermodularity of v implies

$$v(\{a_{i_1}, \dots, a_{i_j}\}) - v(\{a_{i_1}, \dots, a_{i_{j-1}}\}) \leq v(\{a_1, \dots, a_{i_j}\}) - v(\{a_1, \dots, a_{i_{j-1}}\}) = x_{i_j}.$$

By adding up these inequalities, we obtain $v(C) \leq x(C)$, i.e., coalition C does not have an incentive to deviate, which is a contradiction. ■

Observe that the construction used in the proof of Theorem 8.2 immediately implies that in a convex game the Shapley value is in the core: indeed, the Shapley value is a convex combination of outcomes constructed in the proof of Theorem 8.2, and the core can be shown to be a convex set. However, Theorem 8.2 does not, in general, enable us to check whether a given outcome of a convex game is in the core of that game.

3.4 The Least Core

When a given game has an empty core, we may still be interested in finding “the most stable” outcome. In this section, we explore solution concepts that are motivated by this idea. In what follows, we focus on superadditive games; however, many of our definitions also apply to general characteristic function games.

In many situations, a coalition would prefer not to deviate if its gain from a deviation is positive, but tiny. Therefore, we may view outcomes in which no coalition can improve its welfare significantly as stable. This motivates the following definition.

Definition 8.9 *An outcome \mathbf{x} is said to be in the ε -core of a superadditive game G for some $\varepsilon \in \mathbb{R}$ if $x(C) \geq v(C) - \varepsilon$ for each $C \subseteq A$.*

Of course, in practice we are usually interested in finding the smallest value of ε such that the ε -core is non-empty. The corresponding ε -core is called the *least core* of G [39]. More formally, we have the following definition.

Definition 8.10 Given a superadditive game G , let

$$\varepsilon^*(G) = \inf\{\varepsilon \mid \varepsilon\text{-core of } G \text{ is non-empty}\}.$$

The least core of G is its $\varepsilon^*(G)$ -core. The quantity $\varepsilon^*(G)$ is called the value of the least core of G .

To see that the least core is always non-empty, observe that we can modify the linear feasibility program (8.1) so as to obtain a linear program for the value of the least core as well as a payoff vector in the least core. Specifically, we have

$$\begin{aligned} \min \quad & \varepsilon \quad \text{subject to:} \\ & x_i \geq 0 \quad \text{for each } a_i \in A \\ & \sum_{i: a_i \in A} x_i = v(I) \\ & \sum_{i: a_i \in C} x_i \geq v(C) - \varepsilon \quad \text{for each } C \subseteq A. \end{aligned} \tag{8.2}$$

Clearly, if $(\varepsilon, x_1, \dots, x_n)$ is an optimal solution to this linear program, then ε is the value of the least core and (x_1, \dots, x_n) is an outcome in the least core. This shows that we can compute the value of the least core of a superadditive game G as long as we have an algorithm for checking if a given outcome is in the core of G (and, if not, finding the deviating coalition).

Observe that if G has a non-empty core, it may happen that $\varepsilon^*(G) < 0$, in which case the least core is a subset of the core. We remark, however, that some authors require the value of the least core to be non-negative, i.e., they define the least core as the smallest *non-negative* value of ε for which the ε -core is non-empty. Under this definition, to compute the value of the least core we need to add the constraint $\varepsilon \geq 0$ to the linear program (8.2).

3.5 Other Solution Concepts

Besides the Shapley value, the Banzhaf index, the core, and the least core, there are several other solution concepts for characteristic function games. The most prominent among them are the nucleolus, the kernel, and the bargaining set. We will not be able to discuss them in full detail in this chapter due to space constraints; instead, we will provide a brief intuitive description of each of these concepts. For a more comprehensive treatment, the interested reader is referred to [47, 48].

The *nucleolus* [61] can be thought of as a refinement of the least core. Specifically, the least core can be defined as the set of all payoff vectors that minimize the maximum *deficit* $d_1 = \max\{v(C) - x(C) \mid C \subseteq A\}$. Now, among all payoff

vectors in the least core, we can pick the ones that minimize the second highest deficit $d_2 = \max\{v(C) - x(C) \mid C \subseteq A, v(C) - x(C) < d_1\}$, and remove all other payoff vectors. We can continue this procedure until the set of the surviving payoff vectors stabilizes. The resulting set can be shown to consist of a single payoff vector: this payoff vector is known as the *pre-nucleolus*. If, at each step, we only consider imputations (rather than arbitrary payoff vectors), we obtain the *nucleolus*. The nucleolus is an attractive solution concept, as it arguably identifies the most stable outcome of a game. However, its formal definition involves an exponentially long vector, and therefore the nucleolus is not easy to compute from the first principles. However, some classes of games defined on combinatorial structures (see Section 4) admit efficient algorithms for computing the nucleolus: see, e.g., [19, 26, 36].

The *kernel* [17] consists of all outcomes where no player can credibly demand a fraction of another player's payoff. Formally, for any player a_i we define its *surplus* over the player a_j with respect to a payoff vector \mathbf{x} as the quantity

$$sur_{i,j}(\mathbf{x}) = \max\{v(C) - x(C) \mid C \subseteq A, a_i \in C, a_j \notin C\}.$$

Intuitively, this is the amount that a_i can earn without the cooperation of a_j , by asking a set $C \setminus \{a_j\}$ to join it in a deviation, and paying each player in $C \setminus \{a_j\}$ what it used to be paid under \mathbf{x} . Now, if $sur_{i,j}(\mathbf{x}) > sur_{j,i}(\mathbf{x})$, player a_i should be able to demand a fraction of player a_j 's payoff – unless player a_j already receives the smallest payment that satisfies the individual rationality condition, i.e., $v(\{a_j\})$. Following this intuition, we say that an imputation \mathbf{x} is in the *kernel* of a superadditive game G if for any pair of players (a_i, a_j) we have either: (1) $sur_{i,j}(\mathbf{x}) = sur_{j,i}(\mathbf{x})$, or (2) $sur_{i,j}(\mathbf{x}) > sur_{j,i}(\mathbf{x})$ and $x_j = v(\{a_j\})$, or (3) $sur_{i,j}(\mathbf{x}) < sur_{j,i}(\mathbf{x})$ and $x_i = v(\{a_i\})$.

The *bargaining set* [38] is defined similarly to the core. However, in contrast to the definition of the core, we only take into account coalitional deviations that are themselves stable, i.e., do not admit a counterdeviation. Consequently, the bargaining set contains the core, and the containment is sometimes strict. In fact, the bargaining set can be shown to contain the least core [22], which implies that the bargaining set is guaranteed to be non-empty.

4 Representation Formalisms

It would be desirable to have a representation language that allows us to encode all coalitional games so that the description size of each game is polynomial in the number of agents n . However, a simple counting argument shows that no representation formalism can encode each coalitional game using $\text{poly}(n)$ bits; this is true even if we restrict ourselves to simple games. Therefore, one needs

to decide on a trade-off between *expressiveness*, i.e., the formalism's ability to encode many different games, and *succinctness*, i.e., the resulting description size. For instance, one option is to choose a formalism that can only represent games in a certain subclass of coalitional games, but guarantees that each game in this class has a succinct encoding. Alternatively, one can choose a formalism that can represent any coalitional game, but is only guaranteed to produce succinct representation for games that have certain special properties.

In this chapter, we will discuss several formalisms for characteristic function games. We start with *restricted representation languages*, i.e., formalisms that are always succinct, but not fully expressive.

4.1 Weighted Voting Games

In a *weighted voting game*, each player has a certain *weight*, which encodes the amount of resources available to this player. Further, there is a task that can be accomplished by any coalition that has sufficient resources. If a coalition can accomplish the task, it earns a fixed payoff, which can be normalized to 1; otherwise, it earns nothing. Formally, weighted voting games are defined as follows.

Definition 8.11 A weighted voting game G is given by a triple (A, \mathbf{w}, q) , where A is the set of players, $|A| = n$, $\mathbf{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$ is a vector of weights, and $q \in \mathbb{R}$ is a quota. The characteristic function v of a game $G = (A, \mathbf{w}, q)$ is given by $v(C) = 1$ if $\sum_{i: a_i \in C} w_i \geq q$ and $v(C) = 0$ otherwise.

It is usually assumed that all weights and the quota are integers given in binary; it can be shown that this assumption can be made without loss of generality. Further, most of the work on weighted voting games assumes that all weights are non-negative; observe that in this case weighted voting games are simple games.

Weighted voting games are used to model decision making in voting bodies; for instance, the game described in Example 8.3 is a weighted voting game with quota $q = 51$ and weights 40, 22, 30, 9 for players L , M , C , G , respectively. Indeed, the Shapley value and the Banzhaf index in such games are often viewed as measures of a party's voting power in a parliament and have therefore received significant attention from political scientists. In such settings it is usually assumed that the quota q is at least half of the players' total weight; however, in general task execution scenarios the quota q can take any value between 0 and $\sum_{i=1}^n w_i$.

It is important to note that a player's power in a weighted voting game is not necessarily proportional to its weight. Indeed, in Example 8.3, the Liberal party and the Moderate party have the same Shapley value (namely, $1/3$), even though their weights differ by almost a factor of 2. Moreover, the Green party is a dummy and thus its Shapley value is 0, even though it has a non-zero weight. Observe also

that if we changed the quota to, say, $q' = 60$, the balance of power would change: for instance, we would have $v(\{M, C\}) = 0$, but $v(\{M, C, G\}) = 1$, so G would no longer be a dummy.

4.1.1 Computational Issues

The complexity of computing fair and stable outcomes in weighted voting games has received significant attention in the literature.

For instance, the complexity of determining the players' Shapley values has been analyzed by a variety of authors [21, 41, 50]. An easy reduction from the SUBSET SUM problem shows that deciding whether a player is a dummy is coNP-complete; this implies that deciding whether a player's Shapley value is equal to 0 is coNP-complete as well. In fact, one can strengthen this result to show that computing the Shapley value is #P-complete.

Fortunately, the situation is considerably less bleak if we can assume that all weights are at most polynomial in the number of players n , or, equivalently, are given in unary. Under this assumption, we would be satisfied with algorithms whose running time is polynomial in n and the maximum weight, i.e., $\max_{i: a_i \in A} w_i$. It is not too hard to show that such algorithms do exist: a dynamic programming-based approach has been described by Matsui and Matsui [40].

The same easiness and hardness results hold for the Banzhaf index: it is #P-complete to compute when weights are given in binary, but admits an efficient dynamic programming-based algorithm for small weights.

The core-related questions are easy to answer if we make the standard assumption that the grand coalition always forms: indeed, since weighted voting games are simple games, there is a stable way of dividing the payoffs of the grand coalition if and only if the game has veto players. Now, determining if a player a_i is a veto player in a weighted voting game $G = (A, \mathbf{w}, q)$ is easy: it suffices to check whether $\sum_{j: a_j \neq a_i} w_j \geq q$. This implies that there are polynomial-time algorithms for checking if an outcome is in the core or determining whether the core is non-empty.

However, if $q < \sum_{i=1}^n w_i/2$, then forming the grand coalition may be inefficient, and therefore there may exist stable outcomes in which the agents form a non-trivial coalition structure. Indeed, consider the weighted voting game $G = (\{a_1, a_2, a_3, a_4\}, (2, 2, 2, 2), 4)$. This game does not have a veto player, and therefore any outcome in which the grand coalition forms is not stable. On the other hand, it is easy to see that the outcome $(\{\{a_1, a_2\}, \{a_3, a_4\}\}, (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}))$ is stable: any winning coalition contains at least two players, and therefore its payoff is at least 1.

Now, when arbitrary coalition structures are allowed, checking whether a stable outcome exists, or even whether a given outcome is stable, becomes difficult.

Specifically, Elkind et al. [23] showed that that former problem is NP-hard, while the latter problem is coNP-complete. On the positive side, they also showed that if all weights are polynomially bounded, one can check in polynomial time whether an outcome is in the core. It is currently open whether a similar easiness result holds for the problem of checking the non-emptiness of the core; although it is conjectured that this problem remains hard even for small weights [23].

Elkind et al. analyze the complexity of computing the value of the least core and the nucleolus [24, 26]. Again, a familiar picture emerges: both problems are hard when weights are given in binary, but easy when weights are given in unary. Moreover, even for large weights, the value of the least core admits a fully polynomial-time approximation scheme (FPTAS), i.e., an algorithm that, given a weighted voting game $G = (A, \mathbf{w}, q)$ and a parameter δ , outputs a value ϵ' that satisfies $\epsilon \leq \epsilon' \leq (1 + \delta)\epsilon$, where ϵ is the true value of the least core of G , and runs in time that is polynomial in the number of players n , the maximum weight, and $1/\delta$.

4.1.2 Expressivity and Vector Weighted Voting Games

When all weights are non-negative, weighted voting games are simple games. However, one may wonder if the converse is also true, i.e., whether given a simple game $G = (A, v)$ with $|A| = n$ we can always find a vector of weights $\mathbf{w} = \{w_1, \dots, w_n\}$ and a quota q such that G is equivalent to the game (A, \mathbf{w}, q) , i.e., for every $C \subseteq A$ it holds that $v(C) = 1$ if and only if $\sum_{i: a_i \in C} w_i \geq q$.

It is not hard to show that the answer to this question is “no.” Indeed, consider a simple game $G = (A, v)$ with $A = \{a_1, a_2, a_3, a_4\}$, where a coalition is winning if it contains both an even-numbered agent and an odd-numbered agent, or, in symbols, $v(C) = 1$ if and only if $C \cap \{a_1, a_3\} \neq \emptyset$ and $C \cap \{a_2, a_4\} \neq \emptyset$. Suppose that this game can be represented as a weighted voting game (A, \mathbf{w}, q) for some real weights and quota (note that we do not assume that the weights are positive or rational). Since $\{a_1, a_2\}$ and $\{a_3, a_4\}$ are winning coalitions, we have $w_1 + w_2 \geq q$, $w_3 + w_4 \geq q$, and hence $w_1 + w_2 + w_3 + w_4 \geq 2q$. On the other hand, $\{a_1, a_3\}$ and $\{a_2, a_4\}$ are losing coalitions, so we have $w_1 + w_3 < q$, $w_2 + w_4 < q$, and hence $w_1 + w_2 + w_3 + w_4 < 2q$. This contradiction shows that G is not equivalent to any weighted voting game.

Interestingly, the game G discussed in the previous paragraph can be viewed as an *intersection* of two weighted voting games: to win the first game, the coalition must contain an odd-numbered player (this corresponds to the weighted voting game $(A, (1, 0, 1, 0), 1)$), whereas to win the second game, the coalition must contain an even-numbered player (this corresponds to the weighted voting game $(A, (0, 1, 0, 0), 1)$). To win the overall game, the coalition must win both of the component games. Such games are known as *vector weighted voting games*, or

k -weighted voting games, where k is the number of component games.

Definition 8.12 A game $G = (A, v)$ with $|A| = n$ is said to be a k -weighted voting game for some $k \in \mathbb{N}$ if there exists a collection of k weighted voting games $G^1 = (A, (w_1^1, \dots, w_n^1), q^1), \dots, G^k = (A, (w_1^k, \dots, w_n^k), q^k)$ over the set of players A such that $v(C) = 1$ if and only if $\sum_{i: a_i \in C} w_i^j \geq q^j$ for every $j = 1, \dots, k$. The games G^1, \dots, G^k are called the component games of G ; we will write $G = G^1 \wedge \dots \wedge G^k$.

Vector weighted voting games are widely used in practice: for instance, the European Union decision-making system is a 27-player 3-weighted voting game, where the three component games correspond to the commissioners, countries, and population [9].

From the computational perspective, vector weighted voting games are similar to the ordinary weighted voting games if k is bounded by a constant, but become harder to deal with if k is viewed as part of the input: for instance, Elkind et al. [27] show that deciding whether a player is a dummy in a k -weighted voting game is coNP-complete even if all weights are in $\{0, 1\}$ (recall that, in contrast, for weighted voting games this problem is easy as long as all weights are polynomially bounded).

Now, we have seen that vector weighted voting games are more expressive than weighted voting games; but are they fully expressive? We will now show that the answer is “yes,” i.e., *any* simple game can be represented as a k -weighted voting game for a suitable value of k ; this holds even if all weights are required to be in $\{0, 1\}$.

Theorem 8.3 Any simple game $G = (A, v)$ with $|A| = n$ can be represented as a k -weighted voting game $G^1 \wedge \dots \wedge G^k$, where $k \leq 2^n$ and all weights in each component game are either 0 or 1.

Proof. Let C_1, \dots, C_k be the list of all losing coalitions in G . For each coalition C_j in this list, we construct a weighted voting game $G^j = (A, (w_1^j, \dots, w_n^j), q^j)$, where $q^j = 1$ and $w_i^j = 1$ if $a_i \notin C_j$, $w_i^j = 0$ if $a_i \in C_j$. Observe that a coalition C is a winning coalition in G^j if and only if it contains some agent $a_i \in A \setminus C_j$.

We claim that G is equivalent to $G' = G^1 \wedge \dots \wedge G^k$. Indeed, if $C \subseteq A$ is a losing coalition in G , then $C = C_j$ for some $j = 1, \dots, k$, and therefore C loses in the corresponding component game and hence in G' . On the other hand, if $C \subseteq A$ is a winning coalition in G , then, by monotonicity, C is not contained in any losing coalition, i.e., for any coalition C_j in our list we have $C \setminus C_j \neq \emptyset$ and hence C is a winning coalition in C_j . Since this holds for any $j = 1, \dots, k$, C is a winning coalition in G' . To complete the proof, it remains to observe that $k \leq 2^n$. ■

The minimum number of component games in the representation of a given simple game G as a weighted voting game is called the *dimension* of G . Theorem 8.3 shows that the dimension of any simple n -player game G does not exceed 2^n ; on the other hand, there are explicit constructions of simple games whose dimension is exponential in the number of players [69]. Thus, vector weighted voting games are *universally expressive* for the class of all simple games, but are only succinct for some of the games in this class (namely, the games with polynomially small dimension, which includes all weighted voting games). This situation is typical of the universally expressive representation formalisms; we will see some further examples in Section 4.3.

4.2 Combinatorial Optimization Games

Several classes of cooperative games that have been studied in the operations research and theoretical computer science community are defined via a combinatorial structure, such as, for example, a graph. The value of each coalition is obtained by solving a combinatorial optimization problem on the substructure that corresponds to this coalition. We will refer to such games as *combinatorial optimization games*. Just like weighted voting games, such representations are succinct, but not complete. An excellent (though somewhat outdated) survey of combinatorial optimization games can be found in [10]. In this section, we give several examples of the games in this family.

4.2.1 Induced Subgraph Games

In *induced subgraph games* [21], players are vertices of a weighted graph, and the value of a coalition is the total weight of its internal edges. It can be checked that if all weights are non-negative, this game is convex and therefore has a non-empty core. However, if we allow negative weights, the core may be empty, and, moreover, checking whether an outcome is in the core becomes coNP-complete. In contrast, the Shapley value in this game is easy to compute even if the weights can be negative: the Shapley value of a vertex x is half of the total weight of the edges that are incident to x .

4.2.2 Network Flow Games

In *network flow games* [33, 34], the players are edges of a network with a *source* and a *sink*. Each edge has a positive integer capacity, indicating how much flow it can carry. The value of a coalition C is the maximum amount of flow that can be sent from the source to the sink using the edges in C only. Various stability-related

solution concepts for this class of games were studied in [31] and subsequently in [19].

One can also consider a variant of network flow games where the value of a coalition is 1 if it can carry at least k units of flow from the source to the sink, and 0 otherwise. Such games are called *threshold network flow games*, and have been studied in [6] and subsequently in [2].

4.2.3 Matching and Assignment Games

In *assignment games* [65], agents are vertices of a weighted bipartite graph. The value of each coalition is the size of its maximum-weight induced matching. *Matching games* [20] are a generalization of assignment games, where the graph is not required to be bipartite. The complexity of the core, the least core, and the nucleolus in these games has been studied in [36, 68].

4.3 Complete Representation Languages

In this section, we will discuss four representation formalisms for coalitional games that are *complete*, i.e., can be used to describe any coalitional game.

4.3.1 Marginal Contribution Nets

Marginal contribution nets, or *MC-nets* [32], is a rule-based representation; it describes a game with a set of players $A = \{a_1, \dots, a_n\}$ by a collection of rules \mathcal{R} . Each rule $r \in \mathcal{R}$ is of the form $\mathcal{B}_r \rightarrow \vartheta_r$, where \mathcal{B}_r is a Boolean formula over a set of variables $\{b_1, \dots, b_n\}$ and ϑ_r is a real value. We say that a rule $r \in \mathcal{R}$ is *applicable* to a coalition C if \mathcal{B}_r is satisfied by the truth assignment given by $b_i = \top$ if $a_i \in C$ and $b_i = \perp$ if $a_i \notin C$. Let \mathcal{R}^C denote the set of rules that are applicable to C . Then, the characteristic function of the game described by $\mathcal{R} = \{\mathcal{B}_1 \rightarrow \vartheta_1, \dots, \mathcal{B}_k \rightarrow \vartheta_k\}$ is computed as follows:

$$v(C) = \sum_{r \in \mathcal{R}^C} \vartheta_r.$$

Example 8.6 The MC-net that consists of the rules $\mathcal{R} = \{b_1 \wedge b_2 \rightarrow 5, b_2 \rightarrow 2\}$, corresponds to a coalitional game $G = (A, v)$, where $A = \{a_1, a_2\}$, $v(\{a_1\}) = 0$, $v(\{a_2\}) = 2$, $v(\{a_1, a_2\}) = 7$.

An MC-net is said to be *basic* if the left-hand side of any rule is a conjunction of *literals*, i.e., variables and their negations. In this case, we can write a rule $r \in \mathcal{R}$ as $(P_r, N_r) \rightarrow \vartheta_r$, where P_r and N_r are the sets of agents that correspond to positive and negative literals in \mathcal{B}_r , respectively. Thus, r is applicable to coalition

C if C contains every agent in P_r and none of the agents in N_r . It is not hard to see that any coalitional game $G = (A, v)$ with $|A| = n$ can be represented by a basic MC-net with $2^n - 1$ rules: for each non-empty coalition $C \subseteq A$ we create a rule

$$(\bigwedge_{i:a_i \in C} b_i) \bigwedge (\bigwedge_{i:a_i \notin C} \neg b_i) \rightarrow v(C).$$

However, many interesting games admit a more succinct representation, especially if we allow MC-nets that are not basic.

For basic MC-nets, the players' Shapley values can be computed efficiently. The algorithm proceeds by decomposing a game given by k rules into k games – one for each rule; in a game described by a single basic rule, the Shapley value of each player is given by a closed-form expression. This argument extends to *read-once* MC-nets, where in each rule each literal appears at most once [25]. However, if the formulas in the rules can be arbitrary, the Shapley value becomes hard to compute. On the other hand, the core-related questions are NP-hard even for basic MC-nets [32].

4.3.2 Synergy Coalition Groups

Synergy Coalition Group (SCG) Representation [14] is a complete language for superadditive games that is obtained by trimming down the naive representation, i.e., one that lists all coalitions together with their values. It is based on the following idea. Suppose that a game $G = (A, v)$ is superadditive, and consider a coalition $C \subseteq A$. Then we have

$$v(C) \geq \max_{CS \in \mathcal{P}^C \setminus \{C\}} \sum_{C' \in CS} v(C'). \quad (8.3)$$

Now, if the inequality (8.3) holds with equality, then there is no need to store the value of C as it can be computed from the values of the smaller coalitions. Therefore, we can represent G by listing the values of all coalitions of size 1 as well as the values of the coalitions for which there is a *synergy*, i.e., the inequality (8.3) is strict.

By construction, the SCG representation is complete. Moreover, it is succinct when there are only a few groups of agents that can collaborate productively. Further, it allows for an efficient procedure for checking whether an outcome is in the core: it can be shown that if an outcome is not in the core, then there is a “synergetic” coalition, i.e., one whose value is given explicitly in our representation, which can profitably deviate. However, the SCG representation has a major drawback: computing the value of a coalition may involve finding an optimal partition of the players into subcoalitions, and is therefore NP-hard.

4.3.3 Skill-Based Representations

In many settings, the value of a coalition can be defined in terms of the skills possessed by the agents. A simple representation formalism that is based on this idea has been proposed in [46]: there is a set of *skills* S , each agent $a_i \in A$ has a subset of the skills $S^{a_i} \subseteq S$, and there is a function $u : 2^S \rightarrow \mathbb{R}$, which for every subset of skills $S' \subseteq S$ specifies the payoff that can be obtained by a coalition that collectively possesses all the skills in S' . The value of a coalition $C \subseteq A$ is then

$$v(C) = u(\cup_{i:a_i \in C} S^{a_i}).$$

Clearly, this representation is complete, as we can identify each agent a_i with a unique skill s^{a_i} and set $u(S') = v(\{a_i \mid s^{a_i} \in S'\})$ for any subset S' of the skill set. It is succinct when the performance of each coalition can be expressed in terms of a small number of skills possessed by the members of the coalition. Ohta et al. [46] discuss such representations in the context of anonymous environments, where agents can hide skills or split them among multiple identifiers.

A more structured representation was proposed in [5], where coalition values are expressed in terms of skills and tasks. Specifically, in addition to the set of skills S , there is a set of *tasks* Γ , and every task $\tau \in \Gamma$ has a *skill requirement* $S^\tau \subseteq S$ and a payoff. As before, each agent $a_i \in A$ has a set of skills $S^{a_i} \subseteq S$. A coalition $C \subseteq A$ *achieves* a task τ if it has all skills that are required for τ , i.e., if $S^\tau \subseteq \cup_{i:a_i \in C} S^{a_i}$. Finally, there is a *task value function* $F : 2^\Gamma \rightarrow \mathbb{R}$, which for every subset $\Gamma' \subseteq \Gamma$ of tasks specifies the payoff that can be obtained by a coalition that achieves all tasks in Γ' . A *coalitional skill game* [4] is then defined as the coalitional game $\langle A, v \rangle$ where:

$$v(C) = F(\{\tau \mid S^\tau \subseteq \cup_{i:a_i \in C} S^{a_i}\}).$$

This representation is more compact than that of [46] when the number of skills is large (so that the domain of the function u is very large), but the game can be described in terms of a small number of tasks, or if the function F can be encoded succinctly.

4.3.4 Agent-Type Representation

Shrot et al. [67] and Ueda et al. [71] study coalition formation scenarios where agents can be classified into a small number of *types* so that the agents of the same type are symmetric, i.e., make the same contribution to any coalition they belong to. In such settings, the characteristic function can often be specified more succinctly.

More formally, suppose that the set of agents A admits a partition $\{A^1, \dots, A^T\}$ such that for every $i = 1, \dots, T$, every $a_j, a_k \in A^i$ and every coalition C such

that $a_j, a_k \notin C$ it holds that $v(C \cup \{a_j\}) = v(C \cup \{a_k\})$. We will refer to the sets A^1, \dots, A^T as *agent types*. Then the value of any coalition depends solely on how many agents of each type it contains. More precisely, given a coalition $C \subseteq A$, we define the *coalition-type* of C as a vector $\psi = \langle n^1, \dots, n^T \rangle$, where $n^i = |C \cap A^i|$. It is immediate that two coalitions of the same coalition-type have the same value. This means that the conventional characteristic function $v : 2^A \rightarrow \mathbb{R}$ can be replaced with the more concise *type-based characteristic function*, $v^t : \Psi \rightarrow \mathbb{R}$, which is defined on the set

$$\Psi = \{ \langle n^1, \dots, n^T \rangle \mid 0 \leq n^i \leq |A^i| \}$$

of all possible coalition-types. To represent this function, we only need to store $O(n^T)$ coalitional values, since $|\Psi| = (|A^1| + 1) \times \dots \times (|A^T| + 1) < n^T$. Thus, for small values of T , this representation is significantly more succinct than the standard one. On the other hand, it is obviously complete: in the worst case, all agents have different types and v^t coincides with v .

5 Coalition Structure Generation

While the focus so far has been on how to *distribute* the gains from cooperation, in this section we focus on how to *maximize* those gains. To state our computational problem formally, we need some additional notation. Recall that \mathcal{P}^A denotes the space of all coalition structures over the set of agents A ; we extend this notation to subsets of A , and write \mathcal{P}^C to denote the space of all coalition structures over a set $C \subseteq A$. Given a set $C \subseteq A$ and a coalition structure $CS \in \mathcal{P}^C$, let $V(CS)$ denote the *value* of CS , which is calculated as follows: $V(CS) = \sum_{C' \in CS} v(C')$. The *coalition structure generation* problem is then to find an *optimal* coalition structure $CS^* \in \mathcal{P}^A$, i.e., an (arbitrary) element of the set

$$\operatorname{argmax}_{CS \in \mathcal{P}^A} V(CS).$$

This problem is computationally hard. It resists brute-force search, as the number of possible coalition structures over n players, which is known as the *Bell number* B_n [8], satisfies $\alpha n^{n/2} \leq B_n \leq n^n$ for some positive constant α (see, e.g., Sandholm et al. [60] for proofs of these bounds and de Bruijn [18] for an asymptotically tight bound). Moreover, it is NP-hard to find an optimal coalition structure given oracle access to the characteristic function [60]. To date, therefore, a number of algorithms have been developed to try and combat this complexity. In what follows, we will present these algorithms and discuss their relative strengths and weaknesses. However, before we do that, we will present the two main representations of the space of the possible coalition structures as they will provide insight into the way some of these algorithms work.

5.1 Space Representation

To date, there are two main representations of the space of possible coalition structures. The first, proposed by Sandholm et al. [60], is called the *coalition structure graph*. In this undirected graph, every node represents a coalition structure. These nodes are categorized into levels $\mathcal{P}_1^A, \dots, \mathcal{P}_n^A$, where level \mathcal{P}_i^A contains the nodes that represent all coalition structures containing exactly i coalitions. An edge connects two coalition structures if and only if: (1) they belong to two consecutive levels \mathcal{P}_i^A and \mathcal{P}_{i-1}^A , and (2) the coalition structure in \mathcal{P}_{i-1}^A can be obtained from the one in \mathcal{P}_i^A by merging two coalitions into one. A four-agent example can be seen in Figure 8.1.

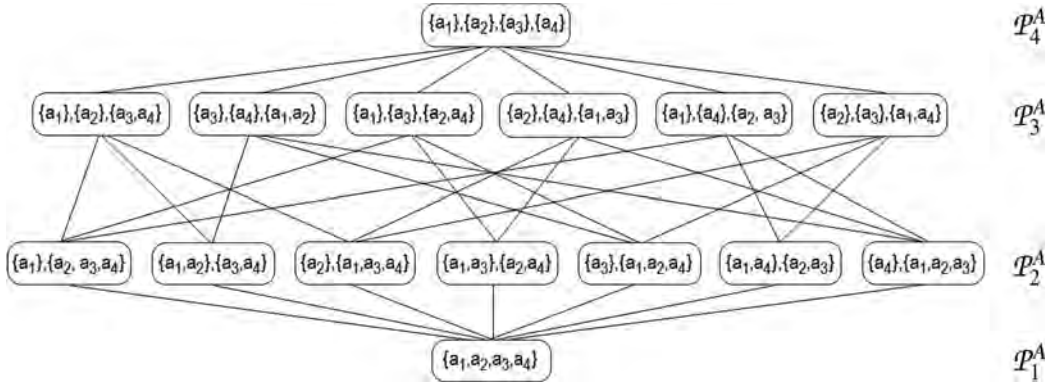


Figure 8.1: The coalition structure graph for four agents.

While the above representation categorizes the coalition structures according to the *number of coalitions* they contain, a different representation was proposed by Rahwan et al. [56] to categorize them based on the *sizes of the coalitions* they contain. More specifically, this representation divides the space of coalition structures into disjoint subspaces that are each represented by an *integer partition* of n . Recall that an *integer partition* of n is a multiset of positive integers, or *parts*, whose sum (with multiplicities) equals to n [1]. For instance, $n = 4$ has five distinct integer partitions, namely, $\{4\}$, $\{1, 3\}$, $\{2, 2\}$, $\{1, 1, 2\}$, and $\{1, 1, 1, 1\}$. Each of these partitions corresponds to the subspace of $\mathcal{P}^{\{a_1, a_2, a_3, a_4\}}$, which consists of all the coalition structures within which the coalition sizes match the parts of the integer partition. We denote by \mathcal{J}^n the set of integer partitions of n , and by \mathcal{P}_I^A the subspace that corresponds to $I \in \mathcal{J}^n$. For instance, $\mathcal{P}_{\{1, 1, 2\}}^{\{a_1, a_2, a_3, a_4\}}$ is the subspace containing all the coalition structures consisting of two coalitions of size 1 and one coalition of size 2. This representation can be encoded by an *integer partition*

graph [52]. This is an undirected graph, where every subspace is represented by a node, and two nodes representing $I, I' \in \mathcal{I}^n$ are connected by an edge if and only if there exist two parts $i, j \in I$ such that $I' = (I \setminus \{i, j\}) \uplus \{i + j\}$ (here \uplus denotes the multiset union operation). For example, Figure 8.2 shows the integer partition graph for four agents, as well as the subspaces that correspond to every node in the graph.

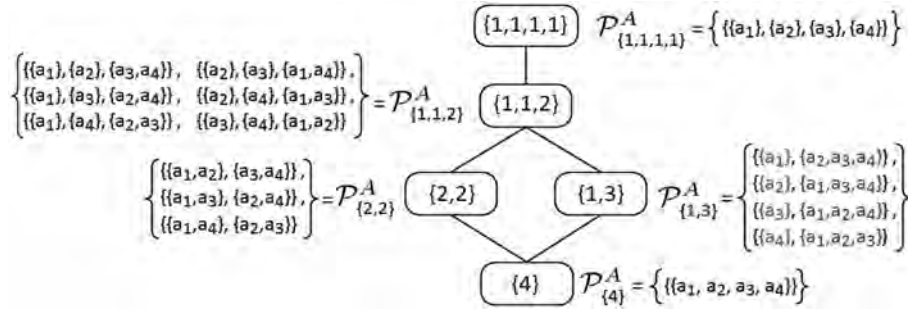


Figure 8.2: The integer partition-based representation for four agents.

Having described the main representations of the search space, in the remaining subsections we will present different approaches to the coalition structure generation problem, some of which are built upon those representations.

5.2 Dynamic Programming Algorithms

The first dynamic programming algorithm, called DP, was proposed by Yeh [72]. This algorithm is based on the following theorem.

Theorem 8.4 *Given a coalition $C \subseteq A$, let $f(C)$ be the value of an optimal partition of C , i.e., $f(C) = \max_{P \in \mathcal{P}^C} V(P)$. Then*

$$f(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max \left\{ v(C), \max_{\{C', C''\} \in \mathcal{P}^C} (f(C') + f(C'')) \right\} & \text{otherwise.} \end{cases} \quad (8.4)$$

Proof. The proof is trivial when $|C| = 1$. Thus, for the remainder of the proof we will assume that $|C| > 1$. Let $\text{opt}(C)$ be some optimal partition of C , i.e., $\text{opt}(C) \in \arg\max_{P \in \mathcal{P}^C} V(P)$. We will make use of the following lemma.

Lemma 8.1 *For any coalition $C \subseteq A$, if $P^* = \{P_1, \dots, P_k\}$ is an optimal partition of C and $k > 1$, then for any $j = 1, \dots, k$ it holds that $P' = \{P_1, \dots, P_j\}$ is an*

optimal partition of $C' = \cup P'$, and $P'' = \{P_{j+1}, \dots, P_k\}$ is an optimal partition of $C'' = \cup P''$.

Proof of Lemma 8.1 To prove the lemma, observe that $P^* = P' \cup P''$ and $V(P^*) = V(P') + V(P'')$. Suppose for the sake of contradiction that P' was not an optimal partition of C' . Then there exists another partition $\hat{P}' \in \mathcal{P}^{C'}$ such that $V(\hat{P}') > V(P')$. However, since $\hat{P}' \cup P''$ is a partition of C , and since $V(\hat{P}' \cup P'') = V(\hat{P}') + V(P'') > V(P^*)$, it follows that P^* cannot be an optimal partition of C , a contradiction. Assuming that P'' is not an optimal partition of C'' leads to a contradiction as well, by a similar argument. Thus, the proof of the lemma is complete. ■

Lemma 8.1 shows that if $|\text{opt}(C)| > 1$, then there exists a coalition structure $\{C', C''\} \in \mathcal{P}^C$ such that $\text{opt}(C) = \text{opt}(C') \cup \text{opt}(C'')$. On the other hand, if $|\text{opt}(C)| = 1$, then surely we would have $\text{opt}(C) = \{C\}$ and $V(\text{opt}(C)) = v(C)$. Equation (8.4) covers both possibilities by taking the maximum over $v(C)$ and $\max_{\{C', C''\} \in \mathcal{P}^C} (f(C') + f(C''))$. ■

The way DP works is by iterating over all the coalitions of size 1, and then over all those of size 2, and then size 3, and so on until size n : for every such coalition C , it computes $f(C)$ using equation (8.4). As can be seen, whenever $|C| > 1$, the equation requires comparing $v(C)$ with $\max_{\{C', C''\} \in \mathcal{P}^C} (f(C') + f(C''))$. The result of this comparison is stored in a table, t , which has an entry for every coalition. In particular, if $v(C)$ was greater, then the algorithm sets $t[C] = C$, so that it can later on remember that it is not beneficial to split C into two coalitions. Otherwise, it sets $t[C] = \text{argmax}_{\{C', C''\} \in \mathcal{P}^C} (f(C') + f(C''))$ to remember the best way of splitting C into two coalitions. By the end of this process, $f(A)$ will be computed, which is by definition equal to $V(CS^*)$. It remains to compute CS^* itself. This is done recursively using the table t . The running time of this algorithm can be shown to be $O(3^n)$.

The execution of the algorithm is illustrated by the following example.

Example 8.7 Given $A = \{a_1, a_2, a_3, a_4\}$, suppose that $t[A] = \{\{a_1, a_2\}, \{a_3, a_4\}\}$, i.e., it is most beneficial to split A into $\{a_1, a_2\}$ and $\{a_3, a_4\}$. Moreover, suppose that $t[\{a_1, a_2\}] = \{\{a_1\}, \{a_2\}\}$, while $t[\{a_3, a_4\}] = \{a_3, a_4\}$, i.e., it is most beneficial to split $\{a_1, a_2\}$ into $\{a_1\}$ and $\{a_2\}$, but it is not beneficial to split $\{a_3, a_4\}$. In this case, $CS^* = \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$.

Although DP is guaranteed to find an optimal coalition structure, Rahwan and Jennings [53] showed that many of its operations are in fact redundant. Based on this, they developed an improved dynamic programming algorithm (IDP) that avoids all redundant operations. To date, IDP is the fastest algorithm that can

find an optimal solution in $O(3^n)$ time. This is significantly less than $\omega(n^{n/2})$ – the time required to exhaustively enumerate all coalition structures. However, the disadvantage is that IDP provides no interim solution before completion, meaning that it is not possible to trade computation time for solution quality.

5.3 Anytime Algorithms

Generally speaking, an *anytime algorithm* is one whose solution quality improves gradually as computation time increases [73]. In our case, this is particularly desirable as the agents might not always have sufficient time to run the algorithm to completion due to the exponential size of the search space. Moreover, being anytime makes the algorithm robust against failure; if the execution is stopped before the algorithm would normally have terminated, then it can still return a solution that is better than the initial – or any intermediate – one.

In this subsection, we will focus on anytime algorithms that return optimal solutions, or at least provide worst-case guarantees on the quality of their solutions.

5.3.1 Identifying Subspaces with Worst-Case Guarantees

A number of researchers have attempted to answer the following question:

*If the solution space is too large to be fully searched, can we search through only a subset of this space, and be guaranteed to find a solution CS^{**} that is within a certain bound β from optimum, that is, $\frac{V(CS^*)}{V(CS^{**})} \leq \beta$?*

This problem can be approached by (1) dividing the space into subsets, and (2) identifying a sequence in which these subsets are searched so that the worst-case bound on solution quality is guaranteed to improve after each subset. The first such algorithm was developed by Sandholm et al. [60], and is mainly based on the following theorem.

Theorem 8.5 *To establish a worst-case bound β , it is sufficient to search the lowest two levels of the coalition structure graph, i.e., \mathcal{P}_1^A and \mathcal{P}_2^A . With this search, the bound is $\beta = n$, and the number of searched coalition structures is 2^{n-1} . Furthermore, no algorithm can establish any bound by searching a different set of at most 2^{n-1} coalition structures.*

Proof. For a partial search to establish a bound on solution quality, every coalition $C \subseteq A$ must appear in at least one of the searched coalition structures. This is due to the possibility of having a single coalition whose value is arbitrarily greater than the values of other coalitions. Now, since the grand coalition appears in \mathcal{P}_1^A , and every other coalition $C \subset A$ appears in $\{C, A \setminus C\} \in \mathcal{P}_2^A$, the value of the

best coalition structure in $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ is at least $\max_{C \subseteq A} v(C)$. On the other hand, since CS^* can include at most n coalitions, its value cannot be greater than $n \times \max_{C \subseteq A} v(C)$. This means $\frac{V(CS^*)}{\max_{CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A} V(CS^*)} \leq n$.

As for the number of searched coalition structures, the reader can check that $|\mathcal{P}_1^A \cup \mathcal{P}_2^A| = 2^{n-1}$. What remains is to show that no bound can be established by searching a different set of at most 2^{n-1} coalition structures. This is done by proving that $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ is the unique subset of \mathcal{P}^A of size at most 2^{n-1} in which every coalition appears in some coalition structure. We leave this as an exercise for the reader. ■

Based on this theorem, the algorithm starts by searching the bottom two levels. After that, if additional time is available, the algorithm searches the remaining levels one by one, starting from the top level and moving downward. Sandholm et al. proved that the bound improves with this search. In particular, once the algorithm completes searching level \mathcal{P}_i^A , the bound becomes $\beta = \lfloor n/h \rfloor$, where $h = \lfloor (n-i)/2 \rfloor + 2$. The only exception is when $n \equiv h-1 \pmod{h}$ and $n \equiv i \pmod{2}$, in which case the bound becomes $\beta = \lceil n/h \rceil$. Importantly, this means that after searching the bottom two levels and establishing the bound $\beta = n$, one can very easily drop (i.e., improve) the bound to $\beta = \lceil n/2 \rceil$ by searching the top level, which only contains one coalition structure.

A different approach was proposed by Dang and Jennings [16]. Their algorithm starts by searching the bottom two levels, as well as the top one (as Sandholm et al.'s algorithm does). After that, however, instead of searching the remaining levels one by one (as Sandholm et al. do), the algorithm searches through certain subsets of all remaining levels. Specifically, it searches the coalition structures that have at least one coalition of size at least $\lceil n(d-1)/d \rceil$ (with d running from $\lfloor (n+1)/4 \rfloor$ down to 2). Dang and Jennings proved that, for any given value of d , the algorithm establishes a bound of $2d-1$.

So far, we have seen how certain bounds can be established by searching certain subsets of the search space. However, with the exception of $\beta = n$ and $\beta = \lceil n/2 \rceil$, we still do not know the *minimum* subset that must be searched in order to establish a desired bound. To this end, let us introduce the following notation. For any integer partition $I \in \mathcal{J}^n$, let \mathcal{P}^I denote the set of possible partitions of I . For instance, $\mathcal{P}^{\{1,1,2\}}$ consists of the following four partitions: $\{\{1,1,2\}\}$, $\{\{1,1\}, \{2\}\}$, $\{\{1,2\}, \{1\}\}$, and $\{\{1\}, \{1\}, \{2\}\}$. Moreover, for any set of integer partitions $\mathcal{J}' \subseteq \mathcal{J}^n$, let $\mathcal{S}(\mathcal{J}')$ be the set that consists of every non-empty subset of every integer partition in \mathcal{J}' , i.e.,

$$\mathcal{S}(\mathcal{J}') = \bigcup_{I \in \mathcal{J}'} \bigcup_{J \subseteq I, J \neq \emptyset} \{J\}.$$

For example, given $\mathcal{J}' = \{\{1, 1, 2\}, \{1, 3\}\}$, the set $\mathcal{S}(\mathcal{J}')$ consists of the following subsets: $\{\{1\}\}$, $\{\{2\}\}$, $\{\{3\}\}$, $\{\{1, 1\}\}$, $\{\{1, 2\}\}$, $\{\{1, 3\}\}$, $\{\{1, 1, 2\}\}$. Finally, for any integer partition $I \in \mathcal{J}^n$ and any set of integer partitions $\mathcal{J}' \subseteq \mathcal{J}^n$, let $\eta(\mathcal{J}', I)$ denote the minimum number of subsets in $\mathcal{S}(\mathcal{J}')$ that are required to construct a partition in \mathcal{P}^I . Formally,

$$\eta(I, \mathcal{J}') = \begin{cases} \min_{S \subseteq \mathcal{S}(\mathcal{J}') : S \in \mathcal{P}^I} |S| & \text{if } \exists S \subseteq \mathcal{S}(\mathcal{J}') : S \in \mathcal{P}^I \\ +\infty & \text{otherwise.} \end{cases}$$

For example, given $I = \{1, 1, 1, 3\}$ and $\mathcal{J}' = \{\{1, 1, 2\}, \{1, 3\}\}$, the minimum number of subsets in $\mathcal{S}(\mathcal{J}')$ that are required to construct a partition of I is 2, and those subsets are $\{1, 1\}$ and $\{1, 3\}$. Therefore, we have $\eta(\mathcal{J}', I) = 2$. Rahwan et al. [55] showed that this definition is crucial when determining the minimum subset that must be searched in order to establish a certain bound. Specifically, they prove the following theorem.

Theorem 8.6 *For any real value b , $1 \leq b \leq n$, and for any $\mathcal{J}' \subseteq \mathcal{J}^n$, we can establish a bound $\beta = b$ by searching $\cup_{I \in \mathcal{J}'} \mathcal{P}_I^A$ if and only if the following holds:*

$$\forall I \in \mathcal{J}^n, \eta(\mathcal{J}', I) \leq b. \quad (8.5)$$

Furthermore, the minimum set of coalition structures that must be searched in order to establish a bound $\beta = b$ is $\cup_{I \in \mathcal{J}^n(b)} \mathcal{P}_I^A$, where $\mathcal{J}^n(b)$ is defined as follows:

$$\mathcal{J}^n(b) \in \arg \min_{\mathcal{J}' \subseteq \mathcal{J}^n : \forall I \in \mathcal{J}^n, \eta(\mathcal{J}', I) \leq b} \left| \cup_{I \in \mathcal{J}'} \mathcal{P}_I^A \right|.$$

In other words, to establish a bound $\beta = b$, all we need to do is to find a set of integer partitions $\mathcal{J}' \subseteq \mathcal{J}^n$ such that, if we take every possible subset of every $I \in \mathcal{J}'$, then with these subsets we can partition every $I \in \mathcal{J}^n$ into at most b parts. One can optimize this by looking for the set of integer partitions that minimizes $\left| \cup_{I \in \mathcal{J}'} \mathcal{P}_I^A \right|$.

We omit the proof of Theorem 8.6 due to space constraints. However, the intuition is similar to the proof of Theorem 8.5, where we proved that a bound $\beta = n$ can be established by searching $\mathcal{P}_1^A \cup \mathcal{P}_2^A$. This was done by showing that CS^* contains at most n coalitions, and that every possible coalition appears in some $CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A$. The proof of Theorem 8.6 generalizes this idea by replacing “coalitions” with “combinations of coalitions”. More specifically, equation (8.5) means that CS^* contains at most b combinations, and that every one of those combinations appears in some $CS \in \cup_{I \in \mathcal{J}'} \mathcal{P}_I^A$.

Theorem 8.6 enables us to describe the set to be searched when establishing a given bound in terms of subspaces that are represented by integer partitions.

Therefore, it would be useful to have an algorithm that can efficiently search those subspaces. In what follows, we present an algorithm that does exactly that.

5.3.2 Integer Partition-Based Search

An anytime algorithm, called IP, was developed by Rahwan et al. [58] based on the integer partition-based representation from Section 5.1. In particular, it uses the observation that, for any subspace \mathcal{P}_I^A , it is possible to compute upper and lower bounds on the value of the best coalition structure in that subspace. More formally, let Max_s^A and Avg_s^A be the maximum and average values of all coalitions of size s , respectively. It turns out that one can compute the average value of the coalition structures in \mathcal{P}_I^A without inspecting these coalition structures [58].

Theorem 8.7 *For any $I \in \mathcal{I}^n$, let $I(i)$ be the multiplicity of i in I . Then:*

$$\frac{\sum_{CS \in \mathcal{P}_I^A} V(CS)}{|\mathcal{P}_I^A|} = \sum_{i \in I} I(i) \cdot Avg_i^A. \quad (8.6)$$

Proof. For any $C \subseteq A$, the number of coalition structures in \mathcal{P}_I^A that contain C depends solely on the size of C . In other words, this number is equal for any two coalitions that are of the same size. Let us denote this number by $\mathcal{N}_I^{|C|}$. Formally, for every $C \subseteq A$ we set $\mathcal{N}_I^{|C|} = |\{CS \in \mathcal{P}_I^A \mid C \in CS\}|$. Then we have

$$\sum_{CS \in \mathcal{P}_I^A} V(CS) = \sum_{i \in I} \sum_{C: |C|=i} \mathcal{N}_I^i \cdot v(C) = \sum_{i \in I} \mathcal{N}_I^i \sum_{C: |C|=i} v(C) = \sum_{i \in I} \mathcal{N}_I^i \cdot \binom{n}{i} \cdot Avg_i^A,$$

where $\binom{n}{i}$ is the binomial coefficient (i.e., the number of possible coalitions of size i). Thus, to prove (8.6) it suffices to prove that

$$\frac{\sum_{i \in I} \mathcal{N}_I^i \cdot \binom{n}{i} \cdot Avg_i^A}{|\mathcal{P}_I^A|} = \sum_{i \in I} I(i) \cdot Avg_i^A.$$

This can be done by proving that the following holds for all $i \in I$:

$$\mathcal{N}_I^i \cdot \binom{n}{i} = I(i) \cdot |\mathcal{P}_I^A|. \quad (8.7)$$

Observe that every $CS \in \mathcal{P}_I^A$ contains exactly $I(i)$ coalitions of size i . Thus:

$$\sum_{C: |C|=i} \mathcal{N}_I^{|C|} = \sum_{C: |C|=i} \sum_{CS \in \mathcal{P}_I^A: C \in CS} 1 = \sum_{CS \in \mathcal{P}_I^A} \sum_{C \in CS: |C|=i} 1 = \sum_{CS \in \mathcal{P}_I^A} I(i) = |\mathcal{P}_I^A| \cdot I(i).$$

We have shown that $\sum_{C:|C|=i} \mathcal{N}_I^{|C|} = |\mathcal{P}_I^A| \cdot I(i)$. On the other hand, since $\mathcal{N}_I^{|C|}$ is equal for all coalitions of size $|C|$, we obtain $\sum_{C:|C|=i} \mathcal{N}_I^{|C|} = \binom{n}{i} \cdot \mathcal{N}_I^i$. Thus, equation (8.7) holds. ■

Based on this theorem, for every $I \in \mathcal{J}^n$, it is possible to compute a lower bound LB_I on the value of the best coalition structure in \mathcal{P}_I^A as follows: $LB_I = \sum_{s \in I} I(s) \text{Avg}_s^A$. This is simply because the best value is always greater than, or equal to, the average one. Similarly, it is possible to compute an upper bound UB_I on the value of the best coalition structure in \mathcal{P}_I^A as $UB_I = \sum_{s \in I} I(s) \text{Max}_s^A$. Using these bounds, the algorithm computes an upper bound $UB^* = \max_{I \in \mathcal{J}^n} UB_I$ and a lower bound $LB^* = \max_{I \in \mathcal{J}^n} LB_I$ on the value of the optimal coalition structure CS^* . Computing UB^* allows for establishing a bound on the quality of the best coalition structure found at any point in time, denoted CS^{**} ; this bound is $\beta = UB^*/V(CS^{**})$. On the other hand, computing LB^* allows for identifying subspaces that have no potential of containing an optimal coalition structure, which are $\mathcal{P}_I^A : UB_I < LB^*$. These subspaces are pruned from the search space. As for the remaining subspaces, the algorithm searches them one at a time, unless a coalition structure is found that has a value greater than, or equal to, the upper bound of some subspace, in which case that subspace no longer needs to be searched. Searching a subspace is done using an efficient process that applies a *branch-and-bound* technique to avoid examining every coalition structure in that subspace whenever possible. A distributed version of IP has also been developed, see [43] for more details.

The IP algorithm can, in the worst case, end up searching the entire space, i.e., it runs in $O(n^n)$ time. In practice, however, IP has been shown to be significantly faster than IDP given popular coalition-value distributions, and the bound that it generates, i.e., $\beta = UB^*/V(CS^{**})$, has been shown to be significantly better than those obtained by searching particular subsets as per the previous subsection.

An extended version of IP, called IDP-IP, was developed by Rahwan and Jennings [52]. As the name suggests, this algorithm is a combination of IDP and IP; it is based on the observation that IDP, even if not run to completion, can still provide useful information. Thus, the basic idea is to partially run IDP, and then use a modified version of IP that can continue the search from where IDP has stopped. This results in a hybrid performance that can be controlled by simply setting the point at which IDP stops. In so doing, one can control the trade-off between the desired features of both IDP and IP. For more details, see [52].

5.3.3 Integer Programming

A different anytime approach, compared to what we have discussed so far, is to formulate the coalition structure generation problem as an integer program. More

specifically, let C_1, C_2, \dots, C_{2^n} denote the possible coalitions. Let z be an $n \times 2^n$ binary matrix, where every row represents an agent and every column represents a coalition, so that $z_{i,j} = 1$ if and only if $a_i \in C_j$. Finally, let us have 2^n decision variables, x_1, x_2, \dots, x_{2^n} , where $x_j = 1$ corresponds to C_j being selected in the solution. The coalition structure generation problem can then be modeled as:

$$\begin{aligned} \max \quad & \sum_{j=1, \dots, 2^n} v(C_j) \cdot x_j \quad \text{subject to:} \\ & \sum_{j=1, \dots, 2^n} z_{i,j} \cdot x_j = 1 \quad \text{for } i = 1, 2, \dots, n \\ & x_j \in \{1, 0\} \quad \text{for } j = 1, 2, \dots, 2^n \end{aligned}$$

With this formulation, it is possible to apply any integer programming solver. However, this approach has been shown to be inefficient, e.g., even an industrial-strength solver such as ILOG's CPLEX was shown to be significantly slower than both IDP and IP, and quickly runs out of memory as the number of agents increases [57].

5.4 Metaheuristic Algorithms

In all the algorithms that were presented so far, the focus was on finding an optimal solution, or a solution that is within a bound from optimum. However, as the number of agents increases, the problem becomes too hard, and the only practical option would be to use metaheuristic algorithms. Such algorithms do not guarantee that an optimal solution is ever found, nor do they provide any guarantees on the quality of their solutions. However, they can usually be applied for very large problems. Next, we outline some of these algorithms.

Sen and Dutta [63] developed a genetic algorithm for coalition structure generation. This algorithm starts with an initial, randomly generated set of coalition structures, called a *population*. After that, the algorithm repeats the following three steps: (1) evaluation, (2) selection, and (3) recombination. More specifically, the algorithm evaluates every member of the current population, selects members based on the outcome of the evaluation, and constructs new members from the selected ones by exchanging and/or modifying their contents.

Keinänen [35] proposed an algorithm based on Simulated Annealing – a generic stochastic local search technique. At each iteration, the algorithm moves from the current coalition structure to a coalition structure in its neighborhood, where neighborhoods can be defined using a variety of criteria. More specifically, the algorithm starts by generating a random coalition structure CS . Then, at every iteration, it samples a random coalition structure CS' in the neighborhood of CS . If CS' is better than CS , then the algorithm sets $CS = CS'$. Otherwise, it sets

$CS = CS'$ with a probability $e^{\frac{V(CS') - V(CS)}{\tau}}$, where τ is the *temperature* parameter that decreases after each iteration according to an *annealing schedule* $\tau = \alpha\tau$, where $0 < \alpha < 1$.

A decentralized, greedy algorithm was proposed by Shehory and Kraus [66]. This algorithm ignores coalitions containing more than a certain number of agents. It returns a coalition structure CS that is constructed iteratively in a greedy manner; at every iteration, the best of all candidate coalitions is added to CS , where a candidate coalition is one that does not overlap with any of the coalitions that were added to CS in previous iterations. The search for the best candidate coalition is done in a distributed fashion; the agents negotiate over which one of them searches which coalitions. A significantly improved distribution mechanism was later on proposed in [51].

Another greedy algorithm, which was put forward by Di Mauro et al. [42], is based on GRASP – a general purpose greedy algorithm, which after each iteration performs a quick local search to try and improve its solution [28]. In the coalition structure generation version of GRASP, a coalition structure CS is constructed iteratively. Every iteration consists of two steps. The first step is to add the best candidate coalition to CS , resulting in a set of pairwise disjoint, but not necessarily exhaustive, coalitions, i.e., $\cup CS \subseteq A$. The second step is to explore different neighborhoods of CS . These two steps are repeated until $\cup CS = A$. Furthermore, the whole process of constructing CS is repeated over and over to try and find better solutions. This algorithm has been shown to work particularly well, with empirical results suggesting that it is the best metaheuristic algorithm for coalition structure generation to date.

5.5 Coalition Structure Generation under Compact Representations

So far, we focused on the coalition structure generation problem under the characteristic function representation (where the input consists of a value for every possible coalition). In what follows, we briefly discuss several papers that consider alternative, often more concise, representations.

5.5.1 Distributed Constraint Optimization

The Distributed Constraint Optimization Problem (DCOP) framework has recently become a popular approach for modeling cooperative agents [44]. In this framework: (1) each agent has a choice of actions, (2) reward is determined by the combination of actions, and (3) the goal is for every agent to choose an action so as to maximize the sum of the rewards. Ueda et al. [70] considered the coalition

structure generation problem where the multiagent system is represented as one big DCOP, and every coalition's value is computed as the optimal solution of the DCOP among the agents of that coalition.

At first glance, this might seem too computationally expensive since there are 2^n possible coalitions. Thus, to find the optimal coalition structure, one might need to solve 2^n instances of the NP-hard DCOP problem. Interestingly, however, Ueda et al. showed that the process of finding an optimal, or near optimal, coalition structure does not have to be divided into two independent stages: (1) computing all coalition values, and (2) finding an optimal combination of disjoint and exhaustive coalitions. Instead, the big DCOP that represents the multiagent system can be modified so that those two stages are merged. This means the desired coalition structure can be obtained by solving a single, modified DCOP.

The modification is controlled by a single parameter, called σ , which specifies the maximum number of coalitions that are allowed to contain more than one agent. We will call these *multiagent* coalitions. The basic idea behind the modification is to change every agent's *domain*, i.e., set of possible actions. Specifically, every action d_j in the original domain is replaced by σ actions, $d_{j,1}, \dots, d_{j,\sigma}$, where $d_{j,i}$ means that the agent performs action d_j while joining the i^{th} multiagent coalition. The new domain also contains an action called “*independent*”, which means that the agent acts independently. The modified DCOP can be solved using any existing algorithm that can obtain an optimal solution, e.g., ADOPT [44] or DPOP [49]. Assuming that the original number of possible actions per agent is d , the search space size for the original DCOP is d^n , while for the modified DCOP it is $(\sigma d + 1)^n$. The following theorem implies that the optimal solution of the modified DCOP is within a bound $\beta = \lfloor \frac{n}{2} \rfloor / \sigma$ from optimum.

Theorem 8.8 *Let $\mathcal{I}_k^n \subseteq \mathcal{I}^n$ be a set in which every integer partition contains at most k integers that are greater than 1. Then, the best coalition structure in $\cup_{I \in \mathcal{I}_k^n} \mathcal{P}_I^A$ is within a bound $\beta = \lfloor \frac{n}{2} \rfloor / k$ from optimum.*

Proof. Assume that CS^* contains ℓ multiagent coalitions, where $\ell > k$. Let $C_1, \dots, C_{\ell-k}$ be the $\ell - k$ coalitions with the smallest values in CS^* . Let us split each coalition $C_i, i = 1, \dots, \ell - k$, into single-agent coalitions; denote the resulting coalition structure by CS'_k . Clearly, $CS'_k \in \cup_{I \in \mathcal{I}_k^n} \mathcal{P}_I^A$. Furthermore, the total value of $C_1, \dots, C_{\ell-k}$ is at most $\frac{\ell-k}{\ell} V(CS^*)$, and the values of the single-agent coalitions are non-negative. Hence, we have $V(CS'_k) \geq \frac{k}{\ell} V(CS^*)$. It remains to observe that $\ell \leq \lfloor \frac{n}{2} \rfloor$. ■

5.5.2 Marginal Contribution Nets

Ohta et al. [45] studied the coalition structure generation problem under the basic MC-net representation (see Section 4.3.1). Recall that a basic MC-net rule can be written as $(P_r, N_r) \rightarrow \vartheta_r$: the interpretation is that a coalition that contains all agents in P_r and none of the agents in N_r can earn a profit of ϑ_r . Ohta et al. consider a restricted class of basic MC-nets, where for each r we have $P_r \neq \emptyset$ and $\vartheta_r > 0$; it can be shown that any characteristic function can be represented by such a restricted MC-net. They define a set of rules $\mathcal{R}' \subseteq \mathcal{R}$ to be *feasible* if all the rules in \mathcal{R}' are applicable *at the same time* to some coalition structure. In other words, \mathcal{R}' is feasible if there exists a coalition structure CS such that every rule $r \in \mathcal{R}'$ is applicable to some $C \in CS$. The problem of finding an optimal coalition structure is then equivalent to the problem of finding a feasible set of rules \mathcal{R}' such that $\sum_{r \in \mathcal{R}'} \vartheta_r$ is maximized. While this problem is NP-hard, Ohta et al. showed that it admits a mixed integer programming (MIP) formulation. Their MIP is based on the observation that, for any two rules r, r' , the possible relations between r and r' can be classified into the following four cases:

- **Compatible on different coalitions (CD):** This is when $P_r \cap P_{r'} = \emptyset$ and $(P_r \cap N_{r'} \neq \emptyset \text{ or } P_{r'} \cap N_r \neq \emptyset)$. For example, $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_3, a_4\}, \{a_1\}) \rightarrow \vartheta_2$ are applicable at the same time in some CS as long as a_1, a_2 appear in a coalition $C \in CS$ and a_3, a_4 appear in a different coalition $C' \in CS$.
- **Incompatible (IC):** This is when $P_r \cap P_{r'} \neq \emptyset$ and $(P_r \cap N_{r'} \neq \emptyset \text{ or } P_{r'} \cap N_r \neq \emptyset)$. For example, $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_2, a_3\}, \{a_1\}) \rightarrow \vartheta_2$ are not applicable at the same time, because the first requires a_1 and a_2 to appear together in a coalition, while the second requires a_2 and a_3 to appear together in a coalition that does not contain a_1 .
- **Compatible on the same coalition (CS):** This is when $P_r \cap P_{r'} \neq \emptyset$ and $P_r \cap N_{r'} = P_{r'} \cap N_r = \emptyset$. For example, $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_2, a_3\}, \{a_4\}) \rightarrow \vartheta_2$ are applicable at the same time to some coalition structure CS as long as there exists $C \in CS$ such that $\{a_1, a_2, a_3\} \subseteq C$ and $a_4 \notin C$. Note that both rules apply to the same coalition.
- **Independent (ID):** This is when $P_r \cap P_{r'} = P_r \cap N_{r'} = P_{r'} \cap N_r = \emptyset$.

Consider a graphical representation of an MC-net in which every node is a rule, and between any two nodes there exists an edge whose type is one of the four cases described above. Then, the following holds:

Theorem 8.9 *A set of rules \mathcal{R}' is feasible if and only if (1) it includes no pair of rules that are connected by an edge of type **IC**, and (2) for any two rules in \mathcal{R}' that are connected by an edge of type **CD**, it is not possible to reach one from the other via a series of edges of type **CS**.*

To understand the intuition behind the proof, consider an example of three rules, r_1, r_2, r_3 . Suppose that for $i = 1, 2, 3$ we have $r_i = (P_i, N_i) \rightarrow \vartheta_i$, where $P_1 = \{a_1, a_2\}$, $N_1 = \emptyset$, $P_2 = \{a_2, a_3\}$, $N_2 = \emptyset$, and $P_3 = \{a_3, a_4\}$, $N_3 = \{a_1\}$. Here, r_1 and r_2 are connected by an edge of type **CS**. Thus, they must be applicable to a single coalition in **CS**, say C' , such that $P_1 \cup P_2 \subseteq C'$. Similarly, an edge of type **CS** connects r_2 and r_3 , and so they must be applicable to a single coalition in **CS**, say C'' , such that $P_2 \cup P_3 \subseteq C''$. Now, since $P_1 \cup P_2$ overlaps with $P_2 \cup P_3$, and since the coalitions in **CS** are pairwise disjoint, we must have $C' = C''$. This means that r_1, r_2, r_3 must all be applicable to the same coalition, i.e., the edge between r_1 and r_3 must *not* be of the type **IC** or **CD**. However, in our example, we happen to have an edge of type **CD** between r_1 and r_3 . Therefore, any rule set containing r_1, r_2, r_3 is not feasible.

Based on Theorem 8.9, Ohta et al. proposed the following MIP formulation.

$$\max \sum_{r \in R} \vartheta_r \cdot x_r \quad \text{subject to:}$$

$$x_{r_i} + x_{r_j} \leq 1 \quad \text{for each edge } (r_i, r_j) \text{ of type **IC**} \quad (8.8)$$

$$y_{r_i}^e = 0 \quad \text{for each edge } e = (r_i, r_j) \text{ of type **CD** with } j > i \quad (8.9)$$

$$y_{r_j}^e \geq 1 \quad \text{for each edge } e = (r_i, r_j) \text{ of type **CD** with } j > i \quad (8.10)$$

$$y_{r_k}^e \leq y_{r_\ell}^e + (1 - x_{r_k}) + (1 - x_{r_\ell})$$

for each edge (r_k, r_ℓ) of type **CS** (8.11)

$$y_{r_\ell}^e \leq y_{r_k}^e + (1 - x_{r_k}) + (1 - x_{r_\ell})$$

for each edge (r_k, r_ℓ) of type **CS** (8.12)

$$x_r \in \{0, 1\} \quad \text{for each } r \in R$$

Here, we have a binary variable x_r for every rule r , where $x_r = 1$ means that r is selected in the solution. Thus, condition (1) of Theorem 8.9 is enforced by the constraint (8.8), which ensures that two rules connected by an edge of type **IC** are never selected at the same time. Moreover, for every edge e of type **CD** or **CS** and every rule r that is adjacent to this edge we define a variable y_r^e . These variables are used in constraints (8.9)–(8.12) to enforce condition (2) of Theorem 8.9. In more detail, for every edge $e = (r_i, r_j)$ of type **CD** with $j > i$ constraints (8.9) and (8.10) ensure that $y_{r_i}^e \neq y_{r_j}^e$. Furthermore, for every edge (r_k, r_ℓ) of type **CS** the constraints (8.10) and (8.11) ensure that, if both r_k and r_ℓ are selected, then $y_{r_k}^e = y_{r_\ell}^e$. Thus, by enforcing both conditions of Theorem 8.9, we guarantee that every solution to this MIP is a feasible rule set.

5.5.3 Coalitional Skill Games

Bachrach et al. [4] considered the coalition structure generation problem in *coalitional skill games* (see Section 4.3.3). While this problem is, in general, very hard computationally, Bachrach et al. showed that it admits an efficient algorithm as long as the number of tasks m and the *treewidth* of a certain associated hypergraph are small. To describe their algorithm, we need a few additional definitions.

Given a skill game with a skill set S , its *skill graph* is a hypergraph $g = \langle V, E \rangle$ in which every agent corresponds to a vertex, and every skill $s_i \in S$ is represented as a hyperedge $e_{s_i} \in E$ that connects all agents that possess this skill. The “complexity” of a hypergraph can be measured using the notion of *treewidth*. The following definition is reproduced from [30] (an illustration is provided in Figure 8.3).

Definition 8.13 Given a hypergraph $g = \langle V, E \rangle$, a tree decomposition of g is a tuple (Q, \mathbf{B}) , where \mathbf{B} is a family of subsets of V (each such subset $B_i \in \mathbf{B}$ is called a bag), and Q is a tree whose node set is \mathbf{B} such that: (1) for each $e \in E$ there is a bag $B_i \in \mathbf{B}$ such that $e \subseteq B_i$; (2) for each $v_j \in V$ the set $\{B_i \in \mathbf{B} \mid v_j \in B_i\}$ is non-empty and connected in Q . The width of (Q, \mathbf{B}) is $\max_{B_i \in \mathbf{B}} |B_i| - 1$. The treewidth of g is the minimum width of (Q, \mathbf{B}) over all possible tree decompositions (Q, \mathbf{B}) of g .

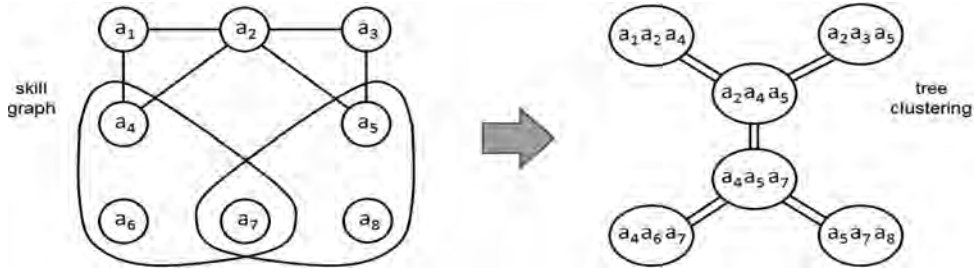


Figure 8.3: A skill graph and its tree decomposition with width 2.

Let $CSG(m, w)$ be the class of all coalitional skill games where the number of tasks is at most m and the treewidth of the corresponding skill graph is at most w . We will now show that, for fixed m and w , the coalition structure generation problem for a game in $CSG(m, w)$ can be solved in time polynomial in the number of agents n and the number of skills k (but exponential in m and w).

To start, observe that a single task can be performed multiple times by a single coalition structure CS . To be more precise, a task that requires a skill which only x agents share can be performed at most x times (this is when each one of those

x agents appears in a different coalition in CS). Let d denote the largest number of agents sharing a single skill; note that $d \leq w + 1$. Then a coalition structure can accomplish at most dm tasks. Based on this, we will define a *candidate task solution* as a set $\{\Gamma_i\}_{i=1}^h$ where each Γ_i is a subset of Γ , and $h \leq dm$. For every coalition structure $CS = \{C_i\}_{i=1}^h$, we say that CS *accomplishes* $\{\Gamma_i\}_{i=1}^h$ if C_i accomplishes all tasks in Γ_i , for $i = 1, \dots, h$. We say that $\{\Gamma_i\}_{i=1}^h$ is *feasible* if there exists at least one coalition structure that accomplishes it. Clearly, the total value obtained by accomplishing these tasks is $\sum_{i=1}^h F(\Gamma_i)$. The problem of finding an optimal coalition structure is thus equivalent to the problem of finding a feasible set of task subsets that maximizes $\sum_{i=1}^h F(\Gamma_i)$. To solve this problem, it is sufficient to iterate over all possible choices of $\{\Gamma_i\}_{i=1}^h$: for each such choice we find the coalition structure that accomplishes it, or determine that it is not feasible. Next, we show how this can be done for a fixed set $\{\Gamma_i\}_{i=1}^h$ in time polynomial in n and k ; the bound on the running time follows as the number of candidate task solutions is bounded by $(2^m)^{dm} \leq (2^m)^{(w+1)m}$.

To this end, observe that every coalition structure can be viewed as a *coloring* of the agents, where all agents with the same color form a coalition. Based on this, for each choice of $\{\Gamma_i\}_{i=1}^h$, let us define a *constraint satisfaction problem*² whose underlying graph is *the skill graph* g , where:

- the **variables** correspond to the agents;
- the **domain** (i.e., the possible values) of each variable (i.e., agent) consists of the possible colors (i.e., the possible coalitions that the agent can join);
- For each skill s , we have the following **constraint**: *For each $i = 1, \dots, h$, if some task in Γ_i requires s , then at least one agent in C_i possesses s .*

To solve this “*primal*” constraint satisfaction problem, we first check if the treewidth of g is bounded by w , and if so return a tree decomposition (this can be done in time polynomial in n and k , see [30]). Then, to solve the primal problem, we define a “*dual*” problem. This is another constraint satisfaction problem whose underlying graph is the tree decomposition of g and

- the **variables** correspond to the bags in the tree decomposition;
- the **domain** of every bag consists of the possible colorings of the agents in the bag. The size of this domain is $O(h^{w+1}) = O((w+1)m^{w+1})$ since every bag contains at most $w + 1$ agents, and every agent has h possible colors;

²For more details on constraint satisfaction problems, see [59].

- the **constraints** are of two types. The first prevents an agent from getting different colors in two neighboring bags. This, in turn, ensures that every agent gets the same color in *all* bags (due to the structure of the tree decomposition). The second type of constraints is exactly the same as the one in the primal problem (i.e., *if a skill is required for at least one task in Γ_i , then at least one agent in C_i possesses that skill*).

Note that a solution to the dual problem is in fact a valid solution to the primal problem. Since the underlying graph of the dual problem is a tree, it can be solved in time polynomial in n and k [4, 59].

5.5.4 Agent-Type Representation

Aziz and de Keijzer [3] and Ueda et al. [71] studied the coalition structure generation problem under the agent-type representation (see Section 4.3.4). Recall that under this representation the game is given by a partition of the set of agents A into T types A^1, \dots, A^T and a type-based characteristic function $v^t : \Psi \rightarrow \mathbb{R}$, where $\Psi = \{\langle n^1, \dots, n^T \rangle \mid 0 \leq n^i \leq |A^i|\}$. Thus, a coalition structure can be viewed as a partition of $\langle |A^1|, \dots, |A^T| \rangle$. Formally, we have the following definition.

Definition 8.14 A type-partition of a coalition-type $\psi = \langle n^1, \dots, n^T \rangle$ is a set of coalition-types $\lambda = \{\langle n_i^1, \dots, n_i^T \rangle\}_{i=1}^\ell$ such that $\langle \sum_{i=1}^\ell n_i^1, \dots, \sum_{i=1}^\ell n_i^T \rangle = \psi$. The value of λ is computed as $V^t(\lambda) = \sum_{i=1}^\ell v^t(\langle n_i^1, \dots, n_i^T \rangle)$.

For example, $\{\langle 0, 1, 2 \rangle, \langle 4, 3, 2 \rangle\}$ is one of the possible type-partitions of $\langle 4, 4, 4 \rangle$, and $V^t(\{\langle 0, 1, 2 \rangle, \langle 4, 3, 2 \rangle\}) = v^t(\langle 0, 1, 2 \rangle) + v^t(\langle 4, 3, 2 \rangle)$.

Thus, while we typically deal with “coalitions” and “coalition structures,” in an agent-type representation we deal with “coalition-types” and “type-partitions.” The problem of finding an optimal coalition structure is then equivalent to that of finding an optimal type-partition of $\langle |A^1|, \dots, |A^T| \rangle$. For example, if we have four types and five agents of each type, we need to find an optimal type-partition of $\langle 5, 5, 5, 5 \rangle$. Two dynamic programming algorithms were proposed for this problem; both run in $O(n^{2T})$ time [3, 71]. We will present the one given in [3], since it is easier to describe.

For any coalition-type $\psi \in \Psi$, let us denote by $f^t(\psi)$ the value of the optimal type-partition of ψ . Then, we can compute $f^t(\psi)$ recursively as follows [3]:

$$f^t(\psi) = \begin{cases} 0 & \text{if } n^i = 0 \text{ for } i = 1, \dots, T \\ \max\{f^t(\langle n^1 - x^1, \dots, n^T - x^T \rangle) + v^t(\langle x^1, \dots, x^T \rangle) & (8.13) \\ \quad | x^i \leq n^i \text{ for } i = 1, \dots, T\} & \text{otherwise.} \end{cases}$$

Based on this recursive formula, we can compute the optimal type-partition by dynamic programming. Specifically, the algorithm works by filling two tables, namely R and Q , each with an entry for every coalition-type. Entry $R[\langle n^1, \dots, n^T \rangle]$ of table R stores an optimal type-partition of $\langle n^1, \dots, n^T \rangle$, whereas entry $Q[\langle n^1, \dots, n^T \rangle]$ of table Q stores the value of this type-partition. The algorithm fills out these tables using (8.13), where “lower” entries are filled in first, i.e., if $m^i \leq n^i$ for all $i = 1, \dots, T$, then $\langle m^1, \dots, m^T \rangle$ is dealt with before $\langle n^1, \dots, n^T \rangle$. For each $\langle n^1, \dots, n^T \rangle$, the algorithm finds a coalition type $\langle x^1, \dots, x^T \rangle$ that maximizes the max-expression of (8.13), and then sets

$$\begin{aligned} Q[\langle n^1, \dots, n^T \rangle] &= Q[\langle n^1 - x^1, \dots, n^T - x^T \rangle] + v^t(\langle x^1, \dots, x^T \rangle), \\ R[\langle n^1, \dots, n^T \rangle] &= R[\langle n^1 - x^1, \dots, n^T - x^T \rangle], \langle x^1, \dots, x^T \rangle. \end{aligned}$$

By the end of this process, we compute $Q[|A^1|, \dots, |A^T|]$ and $R[|A^1|, \dots, |A^T|]$, which provide the solution to the coalition structure generation problem. Filling out each cell of R and Q requires $O(n^T)$ operations, and the size of each table is $|\Psi| < n^T$. Hence, the algorithm runs in time $O(n^{2T})$.

5.6 Constrained Coalition Formation

So far, we assumed that agents can split into teams in any way they like. However, in practice some coalition structures may be inadmissible. To deal with this issue, Rahwan et al. [54] proposed the *constrained coalition formation (CCF)* framework, which allows one to impose constraints on the coalition structures that can be formed. Formally, a CCF game is a tuple $\langle A, \mathcal{CS}, v \rangle$, where A is the set of agents, \mathcal{CS} is the set of coalition structures that are *feasible* (i.e., allowed to form), and v is the characteristic function that assigns a real value to every coalition that appears in some feasible coalition structure. Note that, in the general case, the notion of feasibility is defined for coalition structures rather than coalitions. For instance, if $A = \{a_1, a_2, a_3, a_4\}$ and we define \mathcal{CS} as the set of all coalition structures in which all coalitions have the same size, then the coalition structure $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ is not feasible, even though each of its component coalitions may be a part of a feasible coalition structure.

There are, however, many settings of interest where the constraints implied by \mathcal{CS} can be reduced to constraints on individual coalitions. More formally, a CCF game $G = \langle A, \mathcal{CS}, v \rangle$ is *locally constrained* if there exists a set of coalitions $\mathcal{C} \subseteq 2^A$ such that $\mathcal{CS} = \{CS \in \mathcal{P}^A \mid CS \subseteq \mathcal{C}\}$. We will refer to the coalitions in \mathcal{C} as *feasible coalitions*.

To represent the constraints succinctly, the authors propose the use of propositional logic. More formally, let $B_A = \{b_i \mid a_i \in A\}$ be a set of Boolean variables, and let ϕ be a propositional formula over B_A , constructed using the usual classical

connectives ($\wedge, \vee, \neg, \rightarrow, \dots$). A coalition C *satisfies* φ if φ is satisfied under the truth assignment that sets all b_i with $a_i \in C$ to **true** and all b_i with $a_i \notin C$ to **false**. For example, any coalition containing a_1 and a_2 satisfies $\varphi = b_1 \wedge b_2$. It has been shown that this language can represent any locally constrained CCF game, and that it can be extended so as to represent any CCF game [54].

Rahwan et al. then define a natural subclass of locally constrained CCF games, which they call *basic* CCF games. Intuitively, the constraints in a basic CCF game are expressed in the form of (1) *sizes* of coalitions that are allowed to form, and (2) *subsets* of agents whose presence in any coalition is viewed as desirable/prohibited. The constraints of the former type are called *size constraints*, denoted as $\mathcal{S} \subseteq \{1, \dots, n\}$. As for the latter type of constraints, the desirable subsets of agents are called *positive constraints*, denoted as $\mathcal{P} \subseteq 2^A$, while the prohibited subsets are called *negative constraints*, denoted as $\mathcal{N} \subseteq 2^A$. Thus, a coalition C is feasible if (1) its size is permitted, i.e., $|C| \in \mathcal{S}$, and (2) it contains at least one of the desirable subsets and none of the prohibited ones, i.e., $\exists P \in \mathcal{P} : P \subseteq C$ and $\forall N \in \mathcal{N}, N \not\subseteq C$. We will denote the set of all such feasible coalitions as $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$.

The set of constraints in a basic CCF game can be transformed into another, isomorphic set so as to facilitate both the process of identifying feasible coalitions and the process of searching for an optimal feasible coalition structure [54]. This transformation is based on the observation that, for any agent $a_i \in A$, the coalitions in $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ can be divided into:

- coalitions that contain a_i . For those, any constraint $P \in \mathcal{P} : a_i \in P$ has the same effect as $P \setminus \{a_i\}$. Similarly, any constraint $N \in \mathcal{N} : a_i \in N$ has the same effect as $N \setminus \{a_i\}$. Thus, every such P or N can be replaced with $P \setminus \{a_i\}$ or $N \setminus \{a_i\}$, respectively;
- coalitions that do not contain a_i . For those, every positive or negative constraint that contains a_i has no effect, and so can be removed.

Thus, the problem of dealing with $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ is replaced with two simpler problems; we can then apply the same procedure recursively. This process can be visualized as a tree, where the root is $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$, and each node has two outgoing edges: one leads to a subtree containing some agent a_j and the other leads to a subtree that does not contain a_j . As we move down the tree, the problems become simpler and simpler, until one of the following two cases is reached: (1) a case where one can easily generate the feasible coalitions, which is called a *base case*, or (2) a case where one can easily verify that there are no feasible coalitions (i.e., the constraints cannot be satisfied), which we call an *impossible case* (see [54] for more details). This is illustrated in Figure 8.4 (A), where the edge labels a_i and \bar{a}_i indicate whether the branch contains, or does not contain, a_i , respectively. By

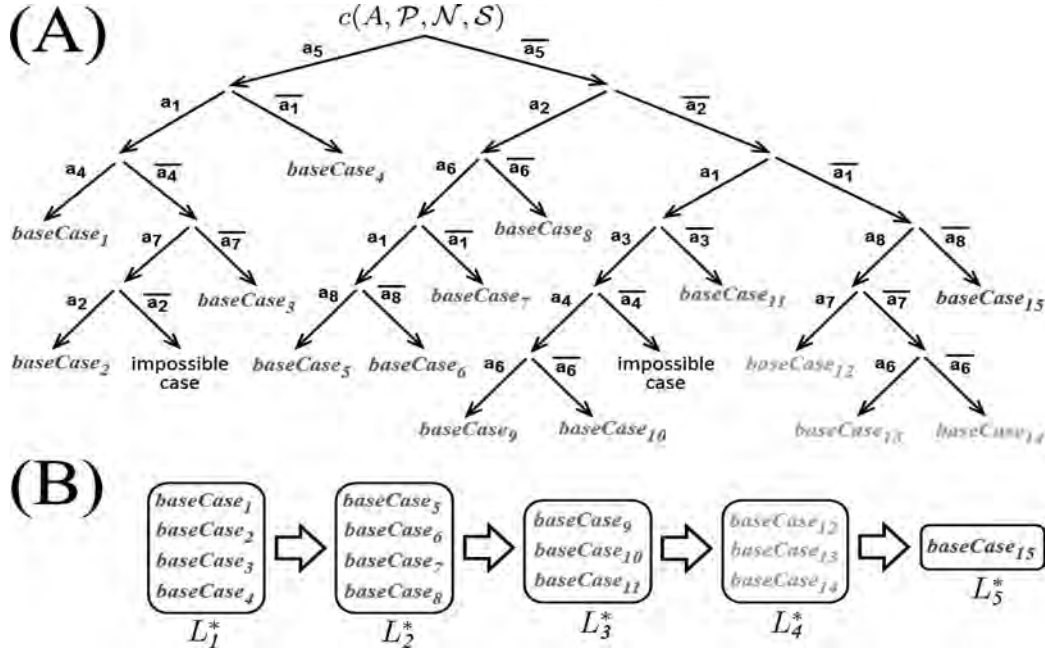


Figure 8.4: Feasible coalitions and coalition structures: given a basic CCF, (A) shows how to generate feasible coalitions, while (B) shows how to generate feasible coalition structures.

generating the feasible coalitions in all base cases, one ends up with the feasible coalitions in $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$.

The tree structure described above also facilitates the search for an optimal feasible coalition structure. Indeed, observe that every such tree contains exactly one path that (1) starts with the root, (2) ends with a leaf, and (3) consists of edges that are each labeled with \bar{a}_i for some $a_i \in A$. In Figure 8.4, for example, this path is the one connecting $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ to baseCase₁₅. Now, let us denote by A^* the sequence of agents that appear in the labels of this path. For instance, in Figure 8.4, we have $A^* = \langle a_5, a_2, a_1, a_8 \rangle$. Finally, let us denote by a_i^* the i^{th} agent in A^* .

With these definitions in place, we can now present the coalition structure generation algorithm in [54]; we will call this algorithm DC as it uses a *divide-and-conquer* technique. The basic idea is to create lists, $L_1^*, \dots, L_{|A^*|+1}^*$, where L_1^* consists of the base cases that contain a_1^* , each L_i^* , $i = 1, \dots, |A^*|$, consists of the base cases that contain a_i^* but not a_1^*, \dots, a_{i-1}^* , and $L_{|A^*|+1}^*$ consists of the base cases that do not contain $a_1^*, \dots, a_{|A^*|}^*$. This is illustrated in Figure 8.4 (B). Importantly, by constructing the lists in this way, we ensure that every feasible coalition structure contains *exactly* one coalition from L_1^* , and *at most* one coalition from

each L_i^* , $i > 1$. Thus, the algorithm picks a coalition, say C_1 , from some base case in L_1^* , and checks whether $\{C_1\}$ is a feasible coalition structure. If not, then the agents in C_1 are added to the negative constraints of all base cases in L_2^* . This places further constraints on the coalitions in those base cases, so as to ensure that they do not overlap with C_1 . Next, the algorithm picks a coalition, say C_2 , from some base case in L_2^* , and checks whether $\{C_1, C_2\}$ is a feasible coalition structure, and so on. Eventually, all feasible coalition structures are examined. To speed up the search, the algorithm applies a branch-and-bound technique (see [54] for more details). This algorithm was compared to the integer programming formulation in Section 5.3.3, where z contains a column for every *feasible* coalition, instead of a column for every *possible* coalition. This comparison showed that DC outperforms the integer programming approach by orders of magnitude.

6 Conclusions

We gave a brief overview of basic notions of cooperative game theory, followed by a discussion of a number of representation formalisms for coalitional games that have been proposed in the literature. We then presented several algorithms for finding an optimal coalition structure, both under the standard representation, and under the more succinct encodings discussed earlier in the chapter. There are several other approaches to the optimal coalition structure generation problem, which we were unable to cover due to space constraints; this problem continues to attract a lot of attention from the multiagent research community due to its challenging nature and numerous applications.

We would like to conclude this chapter by giving a few pointers to the literature. Most standard game theory textbooks provide some coverage of cooperative game theory; the well-known text of Osborne and Rubinstein [47] is a good example. There are also several books that focus exclusively on cooperative games [11, 15, 48]. A very recent book by Chalkiadakis et al. [13] treats the topics covered in the first part of this chapter in considerably more detail than we do, and also discusses coalition formation under uncertainty. However, its coverage of the coalition structure generation problem is much less comprehensive than ours.

7 Exercises

1. **Level 1** Compute the Shapley values of all players in the two variants of the ice cream game described in Example 8.2. Do these games have non-empty cores?

2. **Level 1** Argue that any n -player induced subgraph game can be represented as a basic MC-net with $O(n^2)$ rules.
3. **Level 1** Given the characteristic function shown in Table 8.1, where the value of the grand coalition is 165, identify the optimal coalition structure using the same steps as those of the integer partition-based (IP) algorithm.
4. **Level 1** Write the pseudo-code of the dynamic programming (DP) algorithm for coalition structure generation.
5. **Level 2** Prove Propositions 8.2–8.5.
6. **Level 2** Construct a non-monotone game in which some player's Shapley value is 0, even though this player is not a dummy.

$C: C =1$	$v(C)$	$C: C =2$	$v(C)$	$C: C =3$	$v(C)$	$C: C =4$	$v(C)$
$\{a_1\}$	20	$\{a_1, a_2\}$	40	$\{a_1, a_2, a_3\}$	70	$\{a_1, a_2, a_3, a_4\}$	110
$\{a_2\}$	10	$\{a_1, a_3\}$	30	$\{a_1, a_2, a_4\}$	70	$\{a_1, a_2, a_3, a_5\}$	140
$\{a_3\}$	30	$\{a_1, a_4\}$	30	$\{a_1, a_2, a_5\}$	60	$\{a_1, a_2, a_4, a_5\}$	100
$\{a_4\}$	30	$\{a_1, a_5\}$	40	$\{a_1, a_3, a_4\}$	60	$\{a_1, a_3, a_4, a_5\}$	150
$\{a_5\}$	10	$\{a_2, a_3\}$	40	$\{a_1, a_3, a_5\}$	40	$\{a_2, a_3, a_4, a_5\}$	100
		$\{a_2, a_4\}$	20	$\{a_1, a_4, a_5\}$	80		
		$\{a_2, a_5\}$	30	$\{a_2, a_3, a_4\}$	70		
		$\{a_3, a_4\}$	20	$\{a_2, a_3, a_5\}$	50		
		$\{a_3, a_5\}$	65	$\{a_2, a_4, a_5\}$	75		
		$\{a_4, a_5\}$	35	$\{a_3, a_4, a_5\}$	75		

Table 8.1: Sample characteristic function given five agents.

7. **Level 2** Consider two simple games $G^1 = (A, v^1)$ and $G^2 = (A, v^2)$ with the same set of players A . Suppose that a player $i \in A$ is not a dummy in both games. Can we conclude that i is not a dummy in the game $G^\cap = (A, v^\cap)$, with the characteristic function v^\cap given by $v^\cap(C) = \min\{v^1(C), v^2(C)\}$? What about the game $G^\cup = (A, v^\cup)$, where v^\cup is given by $v^\cup(C) = \max\{v^1(C), v^2(C)\}$?
8. **Level 2** Prove that any outcome in the core maximizes the social welfare, i.e., for any coalitional game G it holds that if (CS, \mathbf{x}) is in the core of G , then $CS \in \operatorname{argmax}_{CS \in \mathcal{P}A} V(CS)$.

9. **Level 2** Argue that the problem of finding an optimal coalition structure in a weighted voting game is NP-hard.
10. **Level 2** Prove that the running time of the dynamic programming algorithm described in Section 5.2 is $O(3^n)$.
11. **Level 2** Provide a formal proof of Theorem 8.6.
12. **Level 3** For every pair $(\mathcal{L}_1, \mathcal{L}_2)$ of complete representation languages considered in Section 4.3, find a family of games that can be compactly represented in \mathcal{L}_1 , but not in \mathcal{L}_2 , or prove that any game that admits a succinct encoding in \mathcal{L}_1 also admits a succinct encoding in \mathcal{L}_2 .
13. **Level 3** Write an implementation of the different metaheuristic algorithms outlined in Section 5.4, and run experiments to compare those algorithms and identify their relative strengths and weaknesses. Are there other metaheuristic algorithms that can be used for coalition structure generation?
14. **Level 3** All the algorithms in Section 5 were developed for settings where (1) overlapping coalitions are prohibited, and (2) every coalition's value is not influenced by the coalition structure to which it belongs (unlike in a partition function game, where a coalition's value in one coalition can be different than that in another). Extend one of those algorithms so as to deal with settings where the aforementioned assumptions do not hold.
15. **Level 4** Elkind et al. [24] show that the problem of computing the least core of a weighted voting game admits a pseudopolynomial algorithm as well as an FPTAS. The pseudopolynomial algorithm extends to the nucleolus [26]; however, it is not known if the problem of computing the nucleolus admits an FPTAS. Develop an FPTAS for this problem, or prove that this is not possible (under a suitable complexity assumption).

References

- [1] George E. Andrews and Kimmo Eriksson. *Integer Partitions*. Cambridge University Press, Cambridge, UK, 2004.
- [2] Haris Aziz, Felix Brandt, and Paul Harrenstein. Monotone cooperative games and their threshold versions. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1107–1114, 2010.
- [3] Haris Aziz and Bart de Keijzer. Complexity of coalition structure generation. In *AAMAS'11: Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 191–198, 2011.

- [4] Yoram Bachrach, Reshef Meir, Kyomin Jung, and Pushmeet Kohli. Coalitional structure generation in skill games. In *AAAI'10: Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 703–708, 2010.
- [5] Yoram Bachrach and Jeffrey S. Rosenschein. Coalitional skill games. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1023–1030, 2008.
- [6] Yoram Bachrach and Jeffrey S. Rosenschein. Power in threshold network flow games. *Autonomous Agents and Multi-Agent Systems*, 18(1):106–132, 2009.
- [7] John F. Banzhaf. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19:317–343, 1965.
- [8] Eric T. Bell. Exponential numbers. *American Mathematical Monthly*, 41:411–419, 1934.
- [9] J. M. Bilbao, J. R. Fernández, N. Jiminéz, and J. J. López. Voting power in the European Union enlargement. *European Journal of Operational Research*, 143:181–196, 2002.
- [10] Peter Borm, Herbert Hamers, and Ruud Hendrickx. Operations research games: A survey. *TOP*, 9:139–199, 2001.
- [11] Rodica Brânzei, Dinko Dimitrov, and Stef Tijs. *Models in Cooperative Game Theory*. Springer, 2005.
- [12] Georgios Chalkiadakis, Edith Elkind, Evangelos Markakis, Maria Polukarov, and Nicholas R. Jennings. Cooperative games with overlapping coalitions. *Journal of Artificial Intelligence Research (JAIR)*, 39:179–216, 2010.
- [13] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan and Claypool, 2011.
- [14] Vincent Conitzer and Tuomas Sandholm. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence*, 170(6–7):607–619, 2006.
- [15] Imma Curiel. *Cooperative Game Theory and Applications*. Kluwer, 1997.
- [16] Viet D. Dang and Nicholas R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS'04: Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 564–571, 2004.
- [17] Morton Davis and Michael Maschler. The kernel of a cooperative game. *Naval Research Logistics Quarterly*, 12(3):223–259, 1965.

- [18] Nicolaas G. de Bruijn. *Asymptotic Methods in Analysis*. Dover, 1981.
- [19] Xiaotie Deng, Qizhi Fang, and Xiaoxun Sun. Finding nucleolus of flow game. In *SODA'06: 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 124–131, 2006.
- [20] Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research*, 24(3):751–766, 1999.
- [21] Xiaotie Deng and Christos Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.
- [22] Ezra Einy, Ron Holzman, and Dov Monderer. On the least core and the Mas-Colell bargaining set. *Games and Economic Behavior*, 28:181–188, 1999.
- [23] Edith Elkind, Georgios Chalkiadakis, and Nicholas R. Jennings. Coalition structures in weighted voting games. In *ECAI'08: Eighteenth European Conference on Artificial Intelligence*, pages 393–397, 2008.
- [24] Edith Elkind, Leslie Ann Goldberg, Paul Goldberg, and Michael Wooldridge. On the computational complexity of weighted voting games. *Annals of Mathematics and Artificial Intelligence*, 56(2):109–131, 2009.
- [25] Edith Elkind, Leslie Ann Goldberg, Paul Goldberg, and Michael Wooldridge. A tractable and expressive class of marginal contribution nets and its applications. *Mathematical Logic Quarterly*, 55(4):362–376, 2009.
- [26] Edith Elkind and Dmitrii Pasechnik. Computing the nucleolus of weighted voting games. In *SODA'09: 20th ACM-SIAM Symposium on Discrete Algorithms*, 2009.
- [27] Piotr Faliszewski, Edith Elkind, and Michael Wooldridge. Boolean combinations of weighted voting games. In *AAMAS'09: 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 185–192, 2009.
- [28] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [29] Donald B. Gillies. Solutions to general non-zero-sum games. In H. W. Kuhn, A. W. Tucker, and L. D. Luce, editors, *Contributions to the Theory of Games, volume IV*, pages 47–85. Princeton University Press, 1959.
- [30] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In *MFCS'01: 26th International Symposium on Mathematical Foundations of Computer Science*, pages 37–57, 2001.

- [31] Daniel Granot and Frieda Granot. On some network flow games. *Mathematics of Operations Research*, 17(4):792–841, 1992.
- [32] Samuel Ieong and Yoav Shoham. Marginal contribution nets: a compact representation scheme for coalitional games. In *ACM EC'05: 6th ACM Conference on Electronic Commerce*, pages 193–202, 2005.
- [33] Ehud Kalai and Eitan Zemel. Generalized network problems yielding totally balanced games. *Operations Research*, 30(5):998–1008, 1982.
- [34] Ehud Kalai and Eitan Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, 7(3):476–478, 1982.
- [35] Helena Keinänen. Simulated annealing for multi-agent coalition formation. In *KES-AMSTA'09: Third KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 30–39, 2009.
- [36] Walter Kern and Daniël Paulusma. Matching games: the least core and the nucleolus. *Mathematics of Operations Research*, 28(2):294–308, 2003.
- [37] William Lucas and Robert Thrall. n -person games in partition function form. *Naval Research Logistic Quarterly*, pages 281–298, 1963.
- [38] Andreu Mas-Colell. An equivalence theorem for a bargaining set. *Journal of Mathematical Economics*, 18:129–139, 1989.
- [39] Michael Maschler, Bezalel Peleg, and Lloyd S. Shapley. Geometric properties of the kernel, nucleolus, and related solution concepts. *Mathematics of Operations Research*, 4:303–338, 1979.
- [40] Tomomi Matsui and Yasuko Matsui. A survey of algorithms for calculating power indices of weighted majority games. *Journal of the Operations Research Society of Japan*, 43(1):71–86, 2000.
- [41] Yasuko Matsui and Tomomi Matsui. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1-2):305–310, 2001.
- [42] Nicola Di Mauro, Teresa M. A. Basile, Stefano Ferilli, and Floriana Esposito. Coalition structure generation with GRASP. In *AIMSA'10: Fourteenth International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 111–120, 2010.
- [43] Tomasz Michalak, Jacek Sroka, Talal Rahwan, Michael Wooldridge, Peter McBurney, and Nicholas R. Jennings. A distributed algorithm for anytime coalition structure generation. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1007–1014, 2010.

- [44] Pragnesh Jay Modi. *Distributed Constraint Optimization for Multiagent Systems*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 2003.
- [45] Naoki Ohta, Vincent Conitzer, Ryo Ichimura, Yuko Sakurai, Atsushi Iwasaki, and Makoto Yokoo. Coalition structure generation utilizing compact characteristic function representations. In *CP'09: Fifteenth International Conference on Principles and Practice of Constraint Programming*, pages 623–638, 2009.
- [46] Naoki Ohta, Atsushi Iwasaki, Makoto Yokoo, Kohki Maruono, Vincent Conitzer, and Tuomas Sandholm. A compact representation scheme for coalitional games in open anonymous environments. In *AAAI'06: Twenty-First National Conference on Artificial Intelligence*, pages 697–702, 2006.
- [47] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [48] David Peleg and Peter Sudhölter. *Introduction to the Theory of Cooperative Games*. Springer, 2007.
- [49] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI'05: Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.
- [50] Kislaya Prasad and Jerry S. Kelly. NP-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, 1990.
- [51] Talal Rahwan and Nicholas R. Jennings. An algorithm for distributing coalitional value calculations among cooperative agents. *Artificial Intelligence*, 171(8–9):535–567, 2007.
- [52] Talal Rahwan and Nicholas R. Jennings. Coalition structure generation: Dynamic programming meets anytime optimisation. In *AAAI'08: Twenty-Third AAAI Conference on Artificial Intelligence*, pages 156–161, 2008.
- [53] Talal Rahwan and Nicholas R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1417–1420, 2008.
- [54] Talal Rahwan, Tomasz P. Michalak, Edith Elkind, Piotr Faliszewski, Jacek Sroka, Michael Wooldridge, and Nicholas R. Jennings. Constrained coalition formation. In *AAAI'11: Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 719–725, 2011.
- [55] Talal Rahwan, Tomasz P. Michalak, and Nicholas R. Jennings. Minimum search to establish worst-case guarantees in coalition structure generation. In *IJCAI'11: Twenty-Second International Joint Conference on Artificial Intelligence*, pages 338–343, 2011.

- [56] Talal Rahwan, Sarvapali D. Ramchurn, Viet D. Dang, and Nicholas R. Jennings. Near-optimal anytime coalition structure generation. In *IJCAI'07: Twentieth International Joint Conference on Artificial Intelligence*, pages 2365–2371, 2007.
- [57] Talal Rahwan, Sarvapali D. Ramchurn, Andrea Giovannucci, Viet D. Dang, and Nicholas R. Jennings. Anytime optimal coalition structure generation. In *AAAI'07: Twenty-Second Conference on Artificial Intelligence*, pages 1184–1190, 2007.
- [58] Talal Rahwan, Sarvapali D. Ramchurn, Andrea Giovannucci, and Nicholas R. Jennings. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research (JAIR)*, 34:521–567, 2009.
- [59] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 2003.
- [60] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst-case guarantees. *Artificial Intelligence*, 111(1–2):209–238, 1999.
- [61] David Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 17:1163–1170, 1969.
- [62] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.
- [63] Sandip Sen and Partha Dutta. Searching for optimal coalition structures. In *ICMAS'00: Sixth International Conference on Multi-Agent Systems*, pages 286–292, 2000.
- [64] Lloyd S. Shapley. A value for n -person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games, volume II*, pages 307–317. Princeton University Press, 1953.
- [65] Lloyd S. Shapley and Martin Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1:111–130, 1972.
- [66] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.
- [67] Tammar Shrot, Yonatan Aumann, and Sarit Kraus. On agent types in coalition formation problems. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 757–764, 2010.
- [68] Tamas Solymosi and T. E. S. Raghavan. An algorithm for finding the nucleolus of assignment games. *International Journal of Game Theory*, 23:119–143, 1994.

- [69] Alan D. Taylor and William S. Zwicker. *Simple Games*. Princeton University Press, 1999.
- [70] Suguru Ueda, Atsushi Iwasaki, Makoto Yokoo, Marius Calin Silaghi, Katsutoshi Hirayama, and Toshihiro Matsui. Coalition structure generation based on distributed constraint optimization. In *AAAI'10: Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 197–203, 2010.
- [71] Suguru Ueda, Makoto Kitaki, Atsushi Iwasaki, and Makoto Yokoo. Concise characteristic function representations in coalitional games based on agent types. In *IJCAI'11: Twenty-Second International Joint Conference on Artificial Intelligence*, pages 393–399, 2011.
- [72] D. Yun Yeh. A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, 26(4):467–474, 1986.
- [73] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

Chapter 9

Trust and Reputation in Multiagent Systems

Jordi Sabater-Mir and Laurent Vercouter

1 Introduction

In open multiagent systems, agents are autonomous and therefore their behavior is not deterministic. On the one hand, this is a desirable feature essential to the nature of the multiagent paradigm. The designer of an agent has to take into account the autonomy of other agents while programming the way it will interact with them. This relaxes constraints on other agents' behavior and makes adaptation during run-time a must for any agent that wants to be competitive. On the other hand, autonomy also brings vulnerabilities. An agent cannot assume that the other individuals will behave following the same code of conduct. Each agent has different interests that do not necessarily agree with the interests of the others.

Similar to what happens in human societies, artificial societies need some kind of mechanism to guarantee a certain degree of control. Traditionally, the control in electronic environments has been approached only from a computational security perspective. The field of trust-based computing has been interested in developing secure systems aimed at preventing a set of well-defined attacks. Proposed solutions often rely on cryptography algorithms. Public-key infrastructures and trusted platform modules are examples of secure systems using asymmetric key algorithms [9]. Even if these techniques can be used to ensure specific properties such as authentication or message integrity, they do not secure a multiagent system regarding aspects like the truth of messages or the subjective fulfillment of a

service. Furthermore, with this approach it is required that there exists a few reliable trusted third parties to provide public and private keys, credentials, or secure deployment infrastructures. These assumptions are unrealistic if decentralization and openness are necessary features.

Control has then to be implemented with an approach compatible with the specific characteristics of open multiagent systems. The term *soft security* has been proposed by Rasmusson and Jansson [32] to refer to control techniques that provide a degree of protection without being too restrictive for the system development. The general idea is to provide social control mechanisms that do not prevent every occurrence of undesirable events but that are able to adapt the system to prevent them from appearing again in the future. Trust and reputation mechanisms have been working in human societies for a long time to implement soft security.

Inspired by its importance in human relations, the use of social trust and reputation in agent interactions has been proposed. Their use is dual. From a local perspective, they are integrated into an agent decision process when it involves other agents in order to decide with whom to interact. From a global perspective, they can be used as social control mechanisms [5]. The particularity of social control mechanisms is that the individuals themselves are in charge of supervising each other. Fear of ostracism is used as a deterrent. An agent that does not behave as expected would be distrusted by its neighbors and socially excluded.

The implementation of trust and reputation mechanisms for software agents needs first an explicit representation of these social evaluations. Section 2 summarizes the different formalisms used to represent them. Then, sections 3 and 4 describe the processes required to implement, respectively, trust and reputation mechanisms. Finally, the connections between trust and other agreement technologies are described in section 5.

2 Computational Representation of Trust and Reputation Values

There is not a unique way to represent trust and reputation values. Existing models use different formalisms, its choice being justified by the type of reasoning the agents have to do. The two main characteristics of this choice are the *simplicity* and the *expressiveness* of values. It is, however, difficult to combine these two characteristics as they are rather opposite, and a trade-off between them has usually to be done. The advantages of having a simple representation is that it facilitates the calculation functions and the reasoning mechanisms. However, simplicity implies less information and therefore the kind of reasoning that can be

done is less sophisticated. On the other hand, expressive values require more computational and storage capacity as well as complex reasoning algorithms to take advantage of that expressiveness. This section shows some of the representations that can be used for trust and reputation values.

2.1 Boolean Representation

The most simple representation for a trust evaluation is using a Boolean. *True* means the trustee is trustworthy (regarding some behaviors) while *False* means the other way around. This representation is very limited and rarely used in current models. Trust (like reputation) is a notion eminently graded and therefore it is important to be able to express how much you trust. It is true that humans use expressions like “I trust him” where the notion of trust seems to be Boolean. However, this kind of sentence refers to the issue of a trust decision process rather than an internal mental representation of the trust value. Section 3 explores this dual nature of trust. Notice that this kind of representation is not useful for reputation values.

2.2 Numerical Values

The second representation in terms of simplicity, both applicable to trust and reputation models, is the use of a real or an integer value. In this case we say that the trust in an agent X is 0.4 or that the reputation of agent Y is -1 . This is by far the most used representation. Usually real values in the interval $[-1, 1]$ (e.g., the ReGreT model [35]) or $[0, 1]$ (Sen & Sajja [40]) are used. Less frequently, you can also find integer values in the intervals $[0, 10]$ or $[-5, 5]$ although it is not restricted to these ranges (see for example the Sporas model [50], which uses the range $[0, 3000]$). In this representation the value is the degree of trust (distrust) or the degree of good (bad) reputation.

The complete order attached to this data type is also useful in comparing several agents. We can then say that two agents A and B are trusted (distrusted) but that we trust (distrust) A more than B if its trust value is higher (lower) than the one attached to B.

The range of possible values may attach a specific semantics to some values. Case in point, it is often assumed that the value 0 in the range $[-1, 1]$ represents a neutral behavior. The values below 0 would then be considered as degrees of distrust (bad reputation) while the ones above are degrees of trust (good reputation). However, the semantics of numerical values is ambiguous and not well defined. The exact meaning of a trust or reputation value is left to the agents' local interpretation. For instance, a given agent may consider that 0.6 represents a very good trust value while another agent would interpret it only as a correct value. This

ambiguity brings about interoperability problems in the event that agents have to exchange their own values.

2.3 Qualitative Labels

Although the numerical representation makes the implementation of the model easier, one aspect that has to be considered is that trust and reputation are vague by nature. Humans usually use expressions like “Her reputation is very good” or “I trust her quite a lot.” They are based on subjective and imprecise information and therefore some authors claim it makes no sense to talk in terms of exact values to describe trust and reputation. Is a trust of 0.6 really different from a trust of 0.7 in terms of making trust decisions? Probably not, so in order to express this lack of precision inherent to both concepts, some models use finite sets of labels in an ordered set to give value to trust and reputation. Usually the set {bad, neutral, good} or the set {very_bad, bad, neutral, good, very_good} are used. These sets are mapped to integer numbers so in fact it is a way of reducing the number of output values to simplify the decision-making process. An example of a model using this kind of representation is that of Abdul Rahman & Hailes [1]. With this representation, the loss of a fine grain comparison of trust and reputation values is compensated by a universally recognized semantics. Of course, it requires that the agents share the same set of labels. Another advantage of this representation is that it is more friendly in case the values have to be interpreted by humans.

2.4 Probability Distribution and Fuzzy Sets

The previous representations have some limitations in terms of expressiveness. For example you cannot express polarized behaviors (always good or bad, never in the middle). Therefore, some models advocate for a richer representation and one of these representations is the use of a discrete probability distribution over a sorted discrete set. A paradigmatic example of this kind of representation is the one used in RepAge [37]. Figure 9.1 shows a graphical representation of evaluations as used in RepAge. The probability distribution is represented using a bar chart over the discrete set of labels. Figure 9.1-a shows an evaluation that states that with a probability of 0.75 the behavior of the agent will be very bad (vb) and with a probability of 0.25 it will be bad (b). Figure 9.1-b states that the behavior of the agent has a probability of 0.5 of being very bad (vb) and a probability of 0.5 of being very good (vg), that is, an agent that is capable of the best and the worse but will never behave in between. Finally Figure 9.1-c typifies an agent that is completely unpredictable.

Similar is the use of fuzzy sets as in the AFRAS [3] model. In AFRAS, a reputation value is a fuzzy set over a range that goes from 0 to 100. The representation

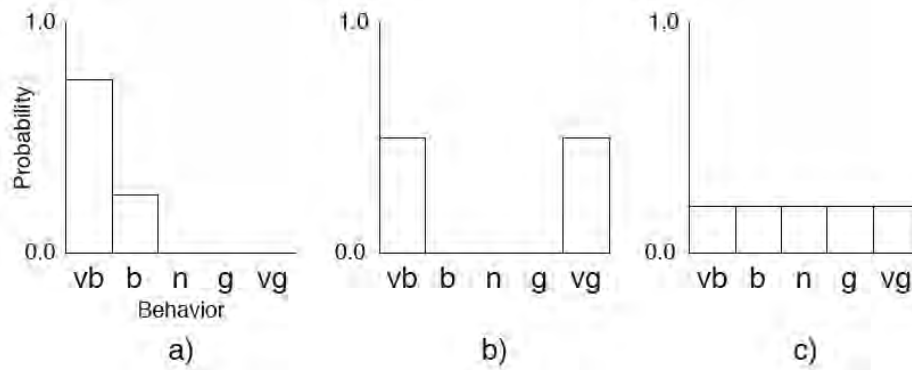


Figure 9.1: Representing reputation as a probability distribution.

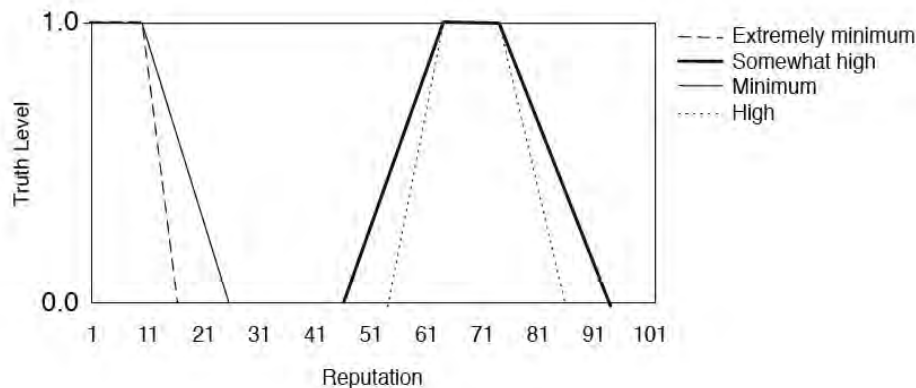


Figure 9.2: Representing reputation using fuzzy sets.

takes advantage of the linguistic modifiers in fuzzy sets. Figure 9.2 shows how linguistic modifiers affect a fuzzy set representing a reputation value. The idea is that in adding these modifiers the issuer expresses the degree of precision of the reputation value represented by the fuzzy set (for example, “somewhat right” expresses less precision than “extremely”). This is reflected in the wideness of the fuzzy set and is interpreted as a measure of reliability of the reputation value, in other words, “the reliability of reputation is implicitly represented in the shape of the fuzzy set.”

2.5 Trust and Reputation as Beliefs

The final objective for an agent regarding trust and reputation values is to be able to reason about them in order to make decisions. This requires a certain degree

of integration between the trust and reputation model and the rest of the elements of the agent. Integration is essential if we consider deliberative architectures, and, more specifically, one of the most successful architectures in the MAS field, the BDI architecture (see Chapter 1). As we said, trust and reputation are meant to be used in agents' reasoning mechanisms, so they have to be represented in the same way as any other mental states. Therefore in a BDI architecture, the trust and reputation values should be represented in terms of beliefs. Using beliefs to represent trust or reputation raises two main issues. The first one is to define the content and the semantics of this specific belief. The second issue consists of linking the belief to the aggregated data grounding it.

The content of a trust or a reputation belief has to include in the representation all the constitutive elements of that belief. For instance, if we rely on the socio-cognitive theory proposed by Castelfranchi and Falcone [6] claiming that "an agent i trusts another agent j in order to do an action α with respect to a goal ϕ ", the formal representation has to be able to express all this information. More specifically, it means that trust is about an agent and has to be relative to a given action and a given goal. Such a formalization has been done in the ForTrust model [16] by the definition of a specific predicate $\text{OccTrust}(i, j, \alpha, \phi)$ holding for specific instances of a trustor (i), a trustee (j), an action (α), and a goal (ϕ). The $\text{OccTrust}(i, j, \alpha, \phi)$ predicate is used to represent the concept of *occurent trust*, which refers to a trust belief holding *here and now*.

The problem of linking beliefs to an aggregation and weighting of values (for example, using a numerical representation of trust and reputation) has been tackled in the BDI+RepAge [30] by the integration of the RepAge model with a BDI architecture. The link consists of transforming each one of the probability values of the probability distribution used in RepAge into a belief. Following the definition that reputation is "what a social entity says about a target regarding his/her behavior" (see Section 4 for a discussion of this definition), the final belief that will be introduced in the belief data base of the agent will be a belief that reflects the certainty that the corresponding evaluation is communicated among the agents in the group. The approach used in BDI+RepAge is based on L_{BC} [27], a belief language and logic that makes it possible to ground the reputation values into beliefs and then reason about them. Among other things, L_{BC} introduces a belief predicate S representing what is *believed to be said in the community*.

For example, from a reputation value calculated by RepAge that says that the reputation of agent j playing the role *seller* is $\text{Rep}(j, \text{seller}, [0.6, 0.1, 0.1, 0.1, 0.1])$ where the probability distribution is defined over the discrete set

$$\{V\text{BadProduct}, \text{BadProduct}, \text{OKProduct}, \text{GoodProduct}, V\text{GoodProduct}\},$$

the system would generate the set of beliefs shown in Figure 9.3. The predicate $S(\text{buy}(j), \text{GoodProduct}, 0.1, \text{seller})$ for example, represents the belief "people say

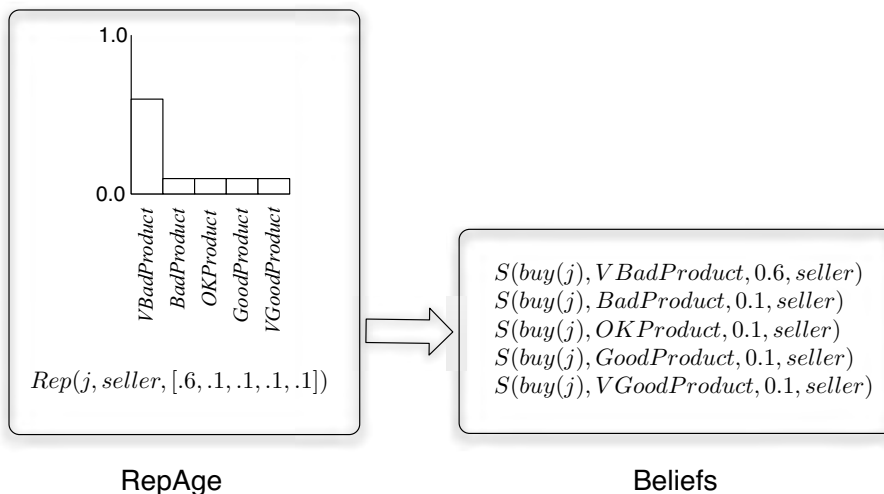


Figure 9.3: Reputation as beliefs in the BDI+RepAge model.

that the probability of receiving a good product from the *sellerj* is 0.1.” Notice that the belief is about what people say, and not about the quality of *j* as a seller.

One relevant aspect here is that because the reputation/trust can be represented in the agent’s mind as a belief (or set of beliefs), it can be used like any other belief to perform a standard BDI reasoning without extending the BDI model.

2.6 The Reliability of a Value

Till now we have been talking about how to represent a trust or reputation value. However, the fact that the model can give us a value doesn’t mean that that value is reliable. To what extent do we have to take into account a trust or reputation value in order to make a decision? Are the foundations of that value strong enough to base a decision on it?

To solve that problem, some models add a measure of the reliability that the trust or reputation value has. The most used and straightforward approach is to associate a number to the trust or reputation value that reflects how reliable it is. The calculation of this number is based on several aspects regarding how, in turn, the trust or reputation value has been calculated. Aspects that can be taken into account are the number of opinions grounding the value, variance of those opinions (more variance implies less reliability), recency of the opinions, credibility of the informers, and so forth.

As we said, if the representation method is based on fuzzy sets, an alternative is to associate the representation of the reliability to the wideness of the fuzzy sets. Therefore, a wide fuzzy set represents a value that is not very reliable, whereas a

narrow fuzzy set reflects the contrary. In other words, the shape of the fuzzy set reflects implicitly the reliability of the reputation value.

3 Trust Processes in Multiagent Systems

A definition of trust that is commonly accepted in the field of multiagent systems is the one proposed by Gambetta [13]: “Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends.” In this definition, trust is an estimation, a prediction of the future or, as Marsh [23] says, “an *expectation* about an uncertain behavior.” From a cognitive perspective, this *expectation* is a psychological attitude.

Castelfranchi and Falcone [6] stress a second nature of trust: trust as an act. Trust is also the “decision and the act of relying on, counting on, depending on [the trustee].” Therefore, trust is both (i) a mental state of the agent (an evaluation) and (ii) a “decision and intention based on that disposition.” This second nature of trust links the *evaluation* with the *decision-making* process. Analyzing the computational trust models in the literature, it is obvious that almost all the effort has been directed to calculating trust evaluations ignoring the motivational aspect. Only recently some models have been proposed which aim at considering trust in its full dimension.

3.1 General Overview of Trust-Related Processes

The dual nature of trust, both epistemic and motivational, should be taken into account while adopting a multiagent perspective, since it allows us to separate naturally two stages: trust evaluation and trust decision. Figure 9.4 shows an overview of the main processes and data involved in multiagent trust models. The figure shows how an agent, called the *trustor*, can use the different sources of information to decide if other agents, called the *trustees*, are trustworthy or not in order to take some actions.

Trust evaluations and trust decisions are the central parts of the model and correspond to the two steps that a trust process has to follow: evaluation and decision making. Trust evaluations can be seen as a summary of all the beliefs, mental states, and values known by the trustor to assess trustees. Two elements feed trust evaluations: *images* and *reputation*. An *image*, as defined by Conte and Paolucci [7], is “an evaluative belief; it tells whether the target is good or bad with respect to a given behavior.” Images are the result of an internal reasoning on different sources of information that leads the agent to create a belief about the behavior of another agent. As shown in Figure 9.4, there are three sources that agents can use to create images.

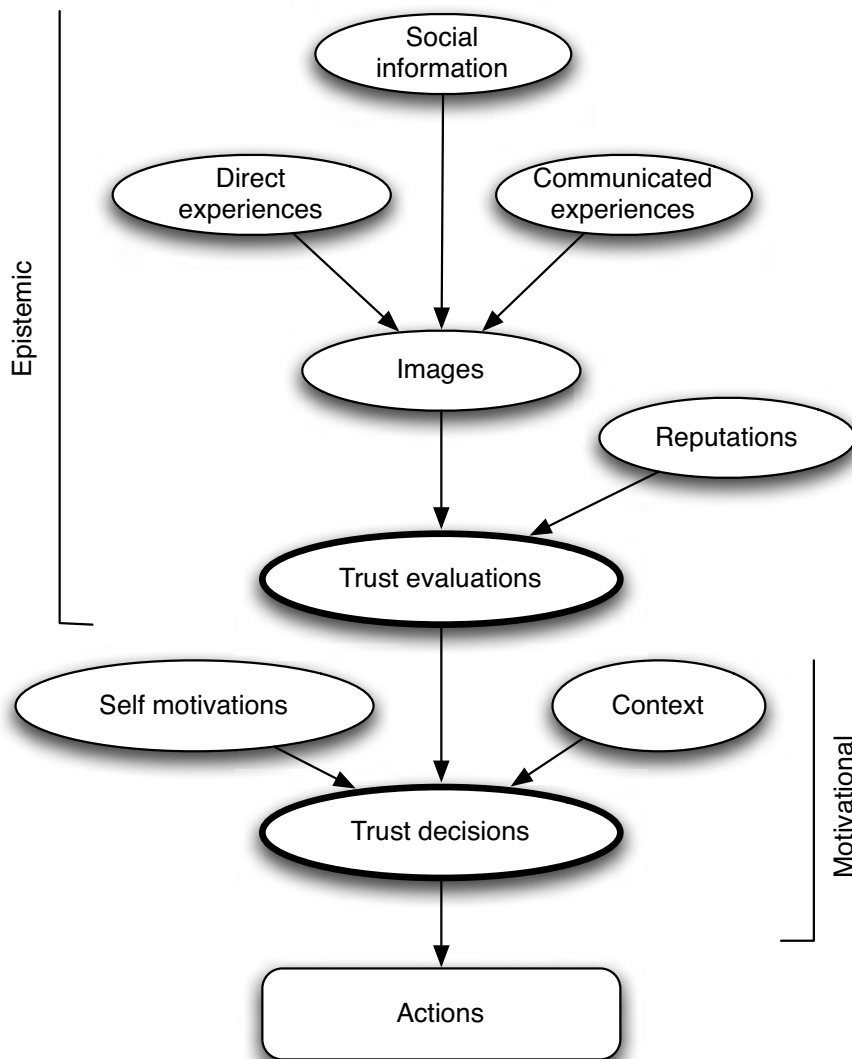


Figure 9.4: The dual nature of trust.

- **Direct experiences:** direct interactions between the trustor and the trustee. This source is usually considered as the most reliable because it comes from a direct perception without intermediaries. An exception would be if the sensors of the agent that allow it to perceive the interaction are not reliable. Almost every multiagent trust model uses the agents' direct experiences as a source for image calculation.
- **Communicated experiences:** information coming from other agents that is

communicated to the trustor. This information describes an interaction between the trustee and an agent different from the trustor. It is possible that this information is transmitted through several agents following a communication chain. Sharing experiences is a way to improve the convergence of image calculation as it increases the number of inputs. However, this information source may be unreliable since agents can provide false reports. Most of the existing trust models [1, 17, 35, 40, 46, 49] use third-party information for image calculation.

- **Social information:** the hierarchical position of the trustee in the society, the social relations with other known individuals, or the role the agent plays in the society are social aspects that can be incorporated in a trust model to calculate images. A prototypical example of a model that makes extensive use of this kind of information is the ReGreT model [35].

Besides images, a trustor can use *reputation* as a source for trust. Reputation is available since the agent evolves in a society in which social evaluations are communicated. Reputation should not be confused with communicated experiences. This concept and its social construction are detailed in section 4.

The second step described in the figure is the *trust decision process*. The goal of this process is to determine if a trustee should be trusted for a given task. The trustor is now in a situation where it is in its own interest that the trustee behaves in an expected way, and the trustor has to decide if it will intend to rely on the trustee. Obviously, trust evaluations are considered to make this decision. But the decision should also take into account the current context since trustworthiness may depend on it. For example, I can trust my car mechanic to fix correctly my car (the trust evaluations indicate he or she is a good mechanic), but I need my car urgently and I see that my mechanic has many cars waiting to be repaired. Although he or she is a good mechanic I know that my mechanic will not be able to do a good job in that context. In that context probably it is better to choose another mechanic with worse trust evaluations but with more availability.

We have seen the general structure of a trust model. In the following subsections we will go in depth in the different aspects of trust.

3.2 Trust Evaluations

The trust calculation process consists of building images from inputs coming from several information sources. Each information source reports experiences with the trustee. Two kinds of sources must be distinguished here: the experiences coming from interactions between the trustee and the trustor themselves, and the experiences communicated by other agents. The first case is often considered as

the most reliable source of experience because it is assumed that the trustor has a correct perception of its own interactions and of its local satisfaction criteria. Communicated experiences are useful to increase the input size of the trust calculation process so that the resulting value is more accurate. However, they may also introduce noise or false information in case agents have different satisfaction criteria (e.g., one agent considers an interaction as satisfactory while another one may have considered it differently), or if some agents create fake experience reports (e.g., a group of malicious agents trying to recommend each other).

The nature of the inputs as well as the chosen formalisms for trust evaluations have an influence on the trust calculation process.

3.2.1 Filtering the Inputs

Although it can be useful at some moment to talk about trust as a general property, almost all of the time the agent will consider trust associated with a certain behavior. For example, I may trust a given student to write a good article but not to cook a good meal, and I maintain both evaluations to be used depending on the context. Trust models [16, 17, 33, 35, 46] that include such contextual elements usually associate the experiences and the images with agents' skills, agents' personal traits, or to general conditions. Therefore, in order to build different images associated with different behaviors or agent characteristics, a trust calculation function will take into account only a subset of the available experiences about the trustee. On the other hand, communicated experiences are not always reliable and their reliability depends on the source of the communication. Given that, the selection of the experiences to be considered as inputs should rely on:

- the *context* of the experience, including descriptive elements of the interaction (e.g., a kind of query or a delegated task), the satisfaction criteria used to evaluate the experience, or the environmental conditions;
- the *information source* that has generated the experience.

The computation of images from communicated experiences should be made with caution. Sen et al. [39] have shown that it is not in the interest of selfish agents to share their opinions. The effect of several malicious agents collaborating to send fake experiences may be disastrous for the accuracy of trust decisions. To avoid this, an agent has to learn which agents are reliable sources and which ones are not. In fact, it has to evaluate the trustworthiness of other agents as information sources. For that purpose, a trustor usually compares, for a given trustee, its own direct experiences with the communicated experiences sent by a recommender. The distance between the experiences is used to assess the relevance of the recommender as an information source for future communicated experiences.

Depending on the models, this trust evaluation of agents as information sources is used to weight the influence of communicated experiences [37, 43, 49] or to adjust the content of communicated experiences [1].

One problem that can appear when considering communicated experiences is the *correlated evidence* problem [26]. This happens when several agents observe a single interaction and then communicate about it. Because normally the interaction is not identified in a communication, a third-party agent can receive several communications from those agents, which seem to be different experiences while in fact all of them refer to the same interaction. This problem is difficult to solve if the identification of the interaction cannot be made explicit (something that is quite usual). A possible solution [35] is to use social network analysis to identify clusters of socially related agents. It is known that social groups tend to unify their point of view due to the large amount of common experiences and the intense communication among their members. Because of this, we can assume that the individuals belonging to one of these clusters probably share a common opinion regarding third-party agents. By asking only the most relevant member of the group, we can obtain a representative opinion of the whole group while avoiding the correlated evidence problem.

3.2.2 Statistical Aggregation

The most direct way to define an image is to represent it as a single value. In that case, trust calculation consists of considering, as inputs, several reports about the trustee and in using a function that produces, as an output, the image. The calculation function strongly depends on the type of inputs (experiences) and outputs (images). Thereby, the inputs and outputs are often represented in a similar way.

In the widespread approach that consists of representing trust within a discrete or a continuous set, the calculation function uses data type operators to merge all the experiences into a single value. One approach is to represent images in tuples with the form $t(a, c, td)$ in which a is the trustee, c a context, and td a value where $td \in \{vt, t, u, vu\}$. These possible values correspond to an evaluation, respectively, of *very trustworthy*, *trustworthy*, *untrustworthy*, and *very untrustworthy*. The calculation of an image is done by a trustor by considering the number of direct experiences it had with the trustee a in the context c , estimated as *very good*, *good*, *bad*, or *very bad*. For instance, if most of its experiences have been judged *very good*, it will consider the trustee as *very trustworthy*. Among the first works in the field, the trust model proposed by Abdul-Rahman and Hailes [1] is an example of that approach.

Numerical representations offer a larger set of operators to be used for trust calculation. The most used operator by far is the average of the inputs. In Schillo et al. [38], for instance, an image of a trustor i about a trustee j is calculated as

the percentage of the number of positive behaviors (noted p) of j observed by i among all the behaviors (noted n) of j observed by i , that is, $T_i^j = \frac{p}{n}$.

Usually the inputs are graduated (for example in the range $[-1, 1]$) and instead of a simple average, a weighted average is used. The weights depend on parameters like the recency of the input, the credibility of the source, and so on. This is the case, for example, of the ReGreT system [35] that uses a time dependent function to weight the inputs of the average.

The use of probability distributions opens new possibilities of aggregation as explored in the work by Sierra and Debenham [41]. In what they call “Ideal Enactment,” the agent has an ideal probability distribution that represents “the best that the agent could reasonably expect to happen.” Given that, the measure of trust is the relative entropy between this ideal distribution and the outcomes (that are expressed as punctual probability distributions) and communications about the trustee behavior (represented also as probability distributions). The same approach can be used also to measure the level of certainty (or reliability) of a trustee by calculating the relative entropy between the outcomes. In both cases, a lower entropy means the agent is more trustworthy/reliable.

3.2.3 Logical Beliefs Generation

When trust evaluations are represented as beliefs, the problem of determining the trustworthiness of a trustee consists of generating these beliefs. The trust calculation process cannot be implemented here as a statistical function. It should rather define which elementary constitutive beliefs are necessary to build trust.

Only very few trust models represent formally the evaluations as beliefs and perform logical inference on them. For many applications a numerical approach based on statistical aggregations is enough. However, given the nature of trust, a cognitive view is an interesting perspective that makes it possible to build agents that deal with trust in a more “human like” way. This makes it easier for the owners of the agents to understand and foresee how the agents behave, which increases their confidence in them. It also has the advantage of using a formalism that can be easily integrated into BDI reasoning processes.

The distinction between *trust evaluation* and *trust decision* implies here that we have to consider two different kinds of beliefs. We focus here on the case of trust evaluation (the formal logic representation of trust decision is covered in section 3.3.2). In order to keep persistent beliefs, trust evaluations have to be represented in a way such that all the specific conditions for which an agent is trusted are encapsulated in the belief. This kind of belief is called *dispositional trust*. An agent is thus considered as trustworthy *whenever some conditions are satisfied*.

The constitutive elements of trust have been identified in the socio-cognitive

theory of trust presented in [6]. They consist of internal and external attributions about the trustee:

- the *internal attribution* of the trustee means that it should have the intention of behaving as expected;
- an *external attribution* of the trustee means that it should be able to act as expected and to achieve the goal for which a trust decision is required.

Bringing together all these parameters, dispositional trust of a trustor toward a trustee is relative to an action to perform, a goal to achieve, and some contextual conditions that should hold. An informal example of dispositional trust belief is $\text{DispTrust}(\text{Alice}, \text{Tom}, \text{inform}(\text{weather}), \text{know}(\text{Alice}, \text{weather}), \text{asked}(\text{Alice}, \text{Tom}, \text{weather}))$. This belief mentions that Alice trusts Tom for performing the action $\text{inform}(\text{weather})$ and by this action to achieve the goal $\text{know}(\text{Alice}, \text{weather})$ whenever the condition $\text{asked}(\text{Alice}, \text{Tom}, \text{weather})$ holds.

A prototypical example of a model that follows this approach is the ForTrust model [16], which proposes an epistemic formalization of dispositional trust:

$$\text{DispTrust}(i, j, \alpha, \phi, \kappa) \stackrel{\text{def}}{=} \text{PotGoal}_i(\phi, \kappa) \wedge$$

$$\text{Bel}_i G^*((\kappa \wedge \text{Choice}_i F\phi) \rightarrow (\text{Intends}_j(\alpha) \wedge \text{Capable}_j(\alpha) \wedge \text{After}_{j:\alpha}\phi))$$

Dispositional trust is believed to be true for a trustor i toward a trustee j for doing an action α in order to achieve a goal ϕ when the conditions κ hold if the trustor has potentially the goal ϕ when κ holds ($\text{PotGoal}_i(\phi, \kappa)$), and if the trustor believes that when it has the goal ϕ and when κ holds ($\kappa \wedge \text{Choice}_i F\phi$), the trustee j intends to perform α ($\text{Intends}_j(\alpha)$), is capable of doing it ($\text{Capable}_j(\alpha)$), and has the power to satisfy ϕ by doing α ($\text{After}_{j:\alpha}\phi$). The condition κ is used to contribute to the multidimensionality of the model (together with the action α). It represents a general context that can be a state of the world, for example, the previous sending of a query from i to j to perform α .

3.3 Trust Decision

The last step of a trust model is the decision process that corresponds to the concept of trust *as an act*. If the decision is positive (negative), the trustor will intend to act in a way such that it trusts (distrusts) the trustee. Of course, trust evaluations are considered in a prominent way in that decision. However, they have to be taken into account according to the specific conditions of the current situation, mainly the contextual elements and the motivations of the trustor. The representation formalism used for trust evaluations directs the form of the decision process.

3.3.1 Single Trust Values and Probability Distributions

Trust evaluations represented as a single value, either Boolean, qualitative, or numerical, require a simple decision process. This is usually done by comparing the value to a given threshold. If the trust value (corresponding to the context of the decision if different trust values are maintained for a given trustee but in different contexts) is above the threshold, the issue is to trust the trustee. Otherwise, it is distrusted. Marsh and Briggs [24] propose to use two thresholds: one for trust and one for distrust. The distrust threshold is obviously lower than the trust threshold. This makes it possible to have three possible outputs of the decision: trust, distrust, or uncertainty (in the case that the trust value is in between the two thresholds).

The value of the thresholds depends on the importance of the decision for the trustor and the risk tolerance. This is a way to integrate its self-motivations in the decision process. If the stake is high, the trustor should use a high trust threshold, whereas if the stake is low, it can be less careful in its trust decision.

When trust is represented as a probability distribution, a usual approach is to transform the probability distribution to a single value so it can be compared with the threshold. One possibility is to calculate the center of mass of the distribution and use that value for the comparison with the threshold.

3.3.2 Trust Beliefs

When trust is represented as a belief in a BDI-like architecture, the result of a trust decision influences the agent's intentions. Intentions are the mental states that drive an agent's actions. Thus, trust "as an act" consists of having the intention to rely on a trustee, the content of the intention depending on the nature of the decision. It can be a task to delegate, a query to send, an expected behavior, etc.

The trust belief resulting from a trust decision corresponds to the concept of *occurrent trust* as it has been introduced in section 2.5. Occurrent trust is a trust attitude that holds *here and now*. It means that, given the *current* contextual conditions and a goal that the trustor wants to achieve *now*, the trustor trusts a trustee for performing *now* a given action that will achieve the goal. Occurrent trust is obviously linked to the concept of dispositional trust. More precisely, occurrent trust is inferred from dispositional trust if the trustor has currently the associated goal and if the specific conditions of the dispositional trust hold. Formally, this relation is represented in the ForTrust framework [16] by

$$\text{DispTrust}(i, j, \alpha, \varphi, \kappa) \wedge \text{Choice}_i F\varphi \wedge \text{Bel}_i \kappa \rightarrow \text{OccTrust}(i, j, \alpha, \varphi)$$

This rule states that the trustor i should decide to trust a trustee j to perform an action α in order to achieve a goal φ if agent i : (i) has the corresponding

dispositional trust when the conditions κ hold; (ii) has the goal ϕ ; (iii) believes that the conditions κ currently hold.

3.4 Coping with the Diversity of Trust Models

This section has shown that there exists various ways to represent and calculate trust. But if models differ in their formalisms or the implementation of calculation and decision processes, they all follow mainly the same steps represented in Figure 9.4 – consisting of gathering experiences, computing trust evaluations, and then deciding if an agent should be trusted or not.

However, differences exist and the existent diversity generates two important challenges: how to compare trust models and how agents using different trust models can exchange information about trust. The exchange of trust information is discussed in section 5.5.

Regarding the comparison of trust models one of the most successful initiatives is the ART testbed. The ART testbed [12] was developed as a common comparative tool in order to evaluate the performances of trust (and reputation) models. As the authors claim, the ART testbed was designed to serve in two roles:

- as a competition forum in which researchers can compare their technologies against objective metrics, and
- as an experimental tool, with flexible parameters, allowing researchers to perform customizable, easily-repeatable experiments.

Several competitions were organized between 2005 and 2008. But the ART competitions have also emphasized the limitations of statistical approaches. The best trust models performed well in the specific scenario of the testbed but it was not really possible to explain the reasons for this good performance nor to exploit lessons for real applications. Moreover, it has been shown that different trust models were not able to exchange trust information because of the high heterogeneity of representations. Interoperability between heterogeneous trust models [45] requires a rigorous semantical definition of trust concepts. Sociological groundings are necessary to define a precise semantics to trust concepts that would facilitate interoperability with other software agents but also with a human trust decision.

4 Reputation in Multiagent Societies

A Vietnamese proverb says: “After death, a tiger leaves behind his skin, a man his reputation.” Reputation has been present in human societies since the beginning

of time as a social mechanism that helps these societies to regulate the behavior of their individuals. Apart from this social functionality, reputation also has an individual dimension, acting as one of the main sources used by individuals to build trust relations.

Similar to what happens with the notion of trust, many definitions of what reputation is can be found in the literature, some of them mixing (wrongly) the concept of reputation with that of trust or assuming as general some properties that cannot always be taken for certain. We will base our definition on the work by Pinyol et al. [27]. In this work the authors define an ontology for reputation that at the same time is based on the previous work by Sabater et al. in the RepAge model [37].

We define reputation as “what a social entity says about a target regarding his or her behavior.” Let’s take apart this definition. A “social entity” is defined as a set of individuals plus a set of social relations among these individuals or properties that identify them as a group in front of its own members and the society at large. Ruben [34] defines a social entity as “a group which is irreducible to the sum of its individual members, and so must be studied as a phenomenon in its own right.” Examples of social entities in human societies are companies, sport clubs, neighborhoods, street bands, etc. Notice we are talking about a group of individuals without trying to individualize. It is not what *A*, *B*, or *C* are saying at an individual level but what they say in the name of the group as a whole. When we talk about reputation we lose track of the single individuals, which constitute the group that is responsible for the reputation value. This property is important because this allows reputation to be an efficient mechanism to spread social evaluations by reducing retaliation fear [28]. Because it is the social entity that is responsible for the evaluation and not the issuer at an individual level, the degree of responsibility that the issuer takes is much less. This allows the issuer to be more inclined to spread the evaluation even if it is not so sure about it.

The second important aspect is regarding the word “says.” In the computational reputation models literature, reputation is also defined as the “opinion” that a social entity has about a target. We prefer to use the word “says” because it stresses two other important aspects of reputation. First that the reputation is not necessarily a belief of the issuer and second that reputation cannot exist without communication. It makes no sense to talk about reputation if there is no exchange of evaluations. You could imagine a social entity in which its individuals share an evaluation but there is no communication among them and with the rest of the society. In that case we cannot talk about reputation but only about a *shared evaluation*. Reputation only appears when the evaluation is communicated and *circulates*, being identified as an evaluation associated with the social entity and not with the individual who is performing the communicative act. In this case the

fact that the opinion is believed to be true or not by the communicator is irrelevant. In fact, an individual can help to spread a reputation believing that the truth is the contrary. What is relevant is that it is communicated and attributable to the social entity.

Finally, reputation (like trust) is always associated with a specific behavior/property. It makes no sense to talk about reputation as a general property and, when it is done, it is because the object of the reputation is implicit.

4.1 Reputation-Building Process

The number of reputation models has increased a lot the last few years and it will continue to increase given the relevance these models have for multiagent societies. The goal of this section is not to make an exhaustive analysis of each current model (for these you can refer to one of the many reviews available in the literature [19, 21, 31, 36]) but to show how the calculation of reputation has been approached from the point of view of multiagent systems and to present the main advantages and drawbacks of each approach.

Figure 9.5 shows the main elements that can be taken into account to calculate reputation values. Basically there are three sources that can be used to evaluate reputation: communicated image/third-party image, communicated reputation, and social information.

4.1.1 Communicated Image as a Source for Reputation

This is the most used mechanism to calculate the reputation of a target in computational models. Basically it consists of aggregating the images that other members in the society communicate and taking this aggregation as the reputation value. The communication can be the image that the agent who is performing the communication has regarding the trustee, or a third-party image that is being communicated. The use of image to evaluate a reputation requires that at a certain moment the agent moves from a shared evaluation to a reputation. This implies losing track of the individual agents and generalizing to the social entity. In other words, it implies moving from “individuals A, B, and C have an image regarding agent D that is very good” to “the reputation that agent D has from the point of view of the social entity α is very good,”¹ where A, B, and C (and probably many more individuals) belong to social entity α .

Although many models use this approach, very few take the step from the communicated images to reputation correctly. Taking the step from a set of communicated images to a reputation implies two assumptions: (i) that the evaluation

¹For the sake of clarity we have omitted the object of the reputation in this example.

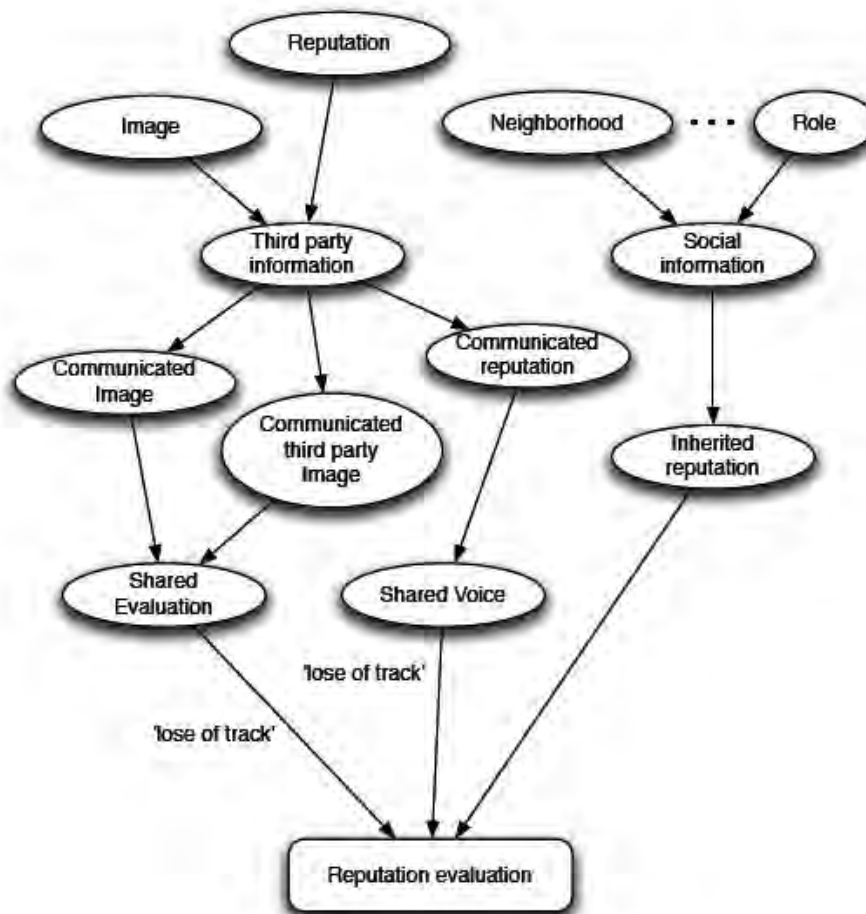


Figure 9.5: Reputation evaluation.

is being communicated and (ii) that the individuals who share the image are a good sample of what the whole social entity thinks. The former is usually fulfilled (we know the image exists because it is communicated) but very few models take into account the latter. Usually, as soon as the first communicated image is received, the model already gives a reputation value. Generalizing a reputation value just from a few images is not very reliable. To lessen the effects of this, some models incorporate a measure of the reliability that a reputation value has (see section 2.6). An example of a model that takes this into account is the ReGreT system [35]. What in ReGreT is called the *witness reputation* is nothing else but a reputation calculated from communicated images. ReGreT uses social information to detect the most representative and credible members of a social entity (those that can represent better the general thinking of the group) and uses

those images to build the reputation. Moreover, it assigns to each reputation value a reliability value that reflects how confident the model is on that calculation. The reliability value is based on the credibility of the individuals whose images are used to build the reputation.

4.1.2 Communicated Reputation

This is probably the most straightforward source to evaluate reputation. It is based on the aggregation of information about reputation received from third parties. This information source depends heavily on having trusted informants that have a good knowledge of the society. This should not be confused with the reputation based on communicated images. Here the informants are communicating directly about reputation values. They are claiming what the reputation value is without necessarily agreeing that the value is coherent with the image that they really have regarding that target. The level of individual compromise the informant is making here is quite less than that in the communication of images. Although the agent can still be judged by the fact that it is spreading that reputation value, it is detached from the value of the reputation. In other words, the agent is only a transmitter of the information, whereas in the case of an image it is also the generator. Again, as in the case of communicated images, at some moment the system has to move from “agents A, B, and C say that the reputation of D in the social entity α is good” (what we call a *shared voice*) to “The reputation of D according to the social entity α is good,” which is a *reputation evaluation*. This step is dependent on the context and is usually based on the number of communications as well as the credibility of the informers.

4.1.3 Inherited Reputation

We call inherited reputation the reputation that is directly inherited from third-party agents with whom the subject has some kind of social relation or the reputation associated with the role the subject is playing in the society. In all these cases the reputation does not depend on the behavior of the subject but on certain properties the subject is supposed to inherit from its social position. For example, the director of a research institute is supposed to be a good researcher because the role of director is supposed to be assigned to good researchers. So even before interacting with him or her, the reputation as a good researcher is assumed. Similarly, social relationships can transmit default reputation. For instance, an employee who works for a certain company inherits the reputation of that company or a member of a family inherits the reputation of its ancestors.

Although social information is not yet a usual source to calculate reputation, some current models have started to use it to improve the lack of more direct

knowledge. The ReGreT [35] model makes an extensive use of inherited reputation. Specifically it takes into account inherited reputation coming from the role the agent is playing (what they call *system reputation*) and inherited reputation coming from the social relations (what they call *neighborhood reputation*). The FIRE [17] model also uses the information about the role to calculate what they call *role-based trust*. This *role-based trust* is, in spite of its name, a reputation value inherited from the reputation that that role has in the society and that is automatically transformed into a trust value.

When we talk about an object instead of an individual, reputation can be inherited also through other kinds of structural relations. For example, if a scientific journal has a good reputation, a paper published in it inherits part of this reputation just because there is a relation “part of.” One model that takes into account the structural relation to calculate reputation is that of Osman et al. [25]. The proposed algorithm makes it possible to calculate how the reputation propagates in a structural graph taking into account the structural relations among individuals/objects.

Inherited reputation is usually used as a starting approach to the actual reputation value. Once the agent starts interacting with the other agents and has enough information to use images as a source for reputation or starts receiving communicated reputations, inherited reputation should be overwritten by those more reliable sources.

4.1.4 Putting It All Together

As it happens in the trust evaluation, to obtain a reputation evaluation the agent has to mix the different sources that influence reputation (communicated images, communicated reputation, and inherited reputation). There is no specific rule that says what the right way is to mix the sources, and this is because this process is highly dependent on the context and the self-motivations (or, in other words, the environment and the beliefs and goals of the agent). Many models use a weighted mean to aggregate the sources. For instance, in the ReGreT [35] model the reputation of an agent b from the point of view of an agent a is calculated using the formula

$$R_{a \rightarrow b}(\varphi) = \sum_{i \in \{W, N, S, D\}} \xi_i \cdot R_{a \rightarrow b}^i(\varphi)$$

where $\{W, N, S, D\}$ correspond to the different types of reputation considered (in this case, witness, neighborhood, system, and default reputation, respectively). The weights ξ_i are calculated taking into account the reliability of the value ($RL_{a \rightarrow b}^i(\varphi)$) and a heuristic that establishes a preference order among the source types.

$$\begin{aligned}
\xi_W &= RL_{a \rightarrow b}^w(\varphi) \\
\xi_N &= RL_{a \rightarrow b}^N(\varphi) \cdot (1 - \xi_W) \\
\xi_S &= RL_{a \rightarrow b}^s(\varphi) \cdot (1 - \xi_W - \xi_N) \\
\xi_D &= 1 - \xi_W - \xi_N - \xi_S
\end{aligned}$$

Using a fixed heuristic can be a problem in highly changing environments. In the previous example the information coming from third-party agents (witness reputation) is considered to be more relevant than the reputation inherited from the role (system reputation). What will happen if the agent starts operating in an environment full of liars and where the information obtained from third parties is always wrong? The agent should be able to adapt the weights to this new scenario giving more relevance to the sources that are providing more reliable information.

4.2 Centralized vs. Decentralized Models

One aspect that has special relevance and that clearly separates computational reputation models in two groups is the kind of architecture for which the model is designed. There is indeed a clear distinction between centralized and decentralized reputation systems. Both approaches have benefits and drawbacks and are adapted to different environmental conditions.

4.2.1 Centralized Approaches

A central service is responsible for collecting the raw information (images, social relations, etc.) from individuals in a social entity and for calculating a reputation value using an aggregation mechanism. This value is then available to the individuals as a measure of reputation.

A centralized reputation system has the following advantages:

- The reputation value is supposed to be calculated using all the information available to the individuals. All of the community is contributing to improving the reputation value calculation and therefore that value should be more accurate.
- Given that usually there is a lot of information to make the calculation, possible wrong or biased information provided by a few individuals has less of a bad impact on the final value.

- The fact that reputation values are public allows newcomers to benefit from this information even without having any previous knowledge about the society.

But it also brings some drawbacks:

- The individuals have to trust the central service regarding the impartiality of the calculation.
- The mechanism used to calculate the reputation is not taking into account personal preferences and biases.
- The central repository is a bottleneck for the system. That means that it introduces a system vulnerability in case of failure of the central service. It may also cause a system overload or slowness in case agents require very frequent access to reputation values and send a huge amount of queries and data to the central repository.
- Providing information to a central authority can create a security problem in certain domains. Aspects like social relations, for example, are difficult to collect in a central repository given their sensitive nature. Related to this, it has been observed that when the source for calculating reputation is based on images, people avoid giving bad evaluations in central repositories because there is fear of retaliation [8], making the calculated reputation too optimistic.

The most well-known centralized reputation mechanisms are those used in online auctions like eBay [10] or Internet review sites like epinions [11]. These models, clearly oriented towards human users, rely on the possibility of reading the textual comments that others have left. Therefore the reputation value usually is used as an indicator that is complemented and fine tuned by reading the textual comments. Another characteristic of these models is that they are used in environments with many (thousands or even millions) users where the repeated interactions among the same partners are scarce. All these models only use communicated images as a source for reputation evaluation.

4.2.2 Decentralized Approaches

The decentralized approach relies on the individual information that each agent can obtain about the society. This information can be images and reputation values coming from third-party agents but also can be information about social relations, roles, etc.

Decentralized approaches have the following advantages:

- It is not necessary that a centralized service stores the individual evaluations and performs the calculation of the reputation value. No trust of an external central entity is necessary.
- They are suited to build scalable systems (such as peer-to-peer networks) as they do not introduce any bottleneck.
- Each agent can decide the method that it wants to follow to calculate reputation.

But they imply the following drawbacks:

- It can take some time for the agent to obtain enough information from the rest of the society to calculate a reliable reputation value. It is not so easy for newcomers to start using reputation in a society that does not have a centralized reputation service.
- It demands more complex and “intelligent” agents as they need to encapsulate processes for reasoning on reputation messages received, calculating reputation, and deciding when to communicate reputation to others.

Many multiagent models fall in this category (ReGret [35], Travos [43], FIRE [17]).

4.3 Using Reputation

The concept of reputation is tightly linked to the one of trust. It has a dual function of being used *for* and *with* trust. In the first case reputation is considered as a source for building trust. In the second case reputation is seen as “an intermediate solution to the problem of social order” [7].

4.3.1 Reputation as a Source of Trust

As we have seen in section 3, reputation is one of the elements that can contribute to building trust in a trustee. Usually reputation is used when there is a lack of direct information. This is how it is used for instance in the ReGreT model [35]. In that model what is called *direct experience* is the main source for building trust, but if that source is not available, the model relies on reputation. Specifically, the model considers three types of reputation values, depending on the information source: information from third parties, social relations, and roles.

4.3.2 Reputation for Social Order

The word *social order* is used in sociology to refer to a set of linked social structures, social institutions, and social practices that conserve, maintain, and enforce “normal” ways of relating and behaving. The same nature of reputation incentivizes “socially acceptable conducts (like benevolence or altruism) and/or forbids socially unacceptable ones.”

The possibility of being excluded is the main deterrent used by reputation mechanisms. In a society that uses reputation, if you follow the norms and have an acceptable behavior² you will be rewarded with greater opportunities to interact with the other members of the society, while if you behave against the norms you will be excluded and ostracized.

Notice that both functions of reputation are intimately related. The fact that reputation is used from an individual point of view to evaluate the behavior of others is exactly what allows reputation to be a mechanism for social order from a social perspective and vice versa.

4.4 Pitfalls When Using Reputation

As any mechanism for social order, reputation is prone to receive attacks from malicious agents. In this section we describe the most common attacks identified in the literature [18]. We note that defense methods against attacks are usually prone to give false positives if they are not used cautiously. For example, when the unfair ratings attack (see Section 4.4.1) is combined with collusion (see Section 4.4.5), it is easy to regard those that are actually correct ones to be unfair. The solution for that is to delay the decisions coming from the defense mechanisms until there is a high degree of certainty and several indicators (not only one) have given clear signals. A compromise, depending on the context, between waiting for clearer signals and acting against the attack is necessary.

4.4.1 Unfair Ratings

An unfair ratings attack occurs when an agent sends deliberately wrong feedback about interactions with another agent. This can be both sending bad feedback while knowing the interaction was good (*badmouthing*) or sending good feedback while knowing that it was bad. A typical approach described in the literature to decrease the effect of unfair ratings is to give more weight to the opinions of those agents that in the past have demonstrated more accuracy in their opinions. In other words, more weight is given to those agents that have acquired a better image/reputation as informers.

²The definition of what is an acceptable behavior is fixed by the society itself.

4.4.2 Ballot-Stuffing

Ballot-stuffing is an attack in which an agent sends more feedback than interactions it has been involved in as a partner. The main counterattacks described in the literature are filtering feedback that comes from peers suspected of ballot-stuffing, and using feedback per interaction rates instead of per accumulation of feedback.

4.4.3 Dynamic Personality

An agent that achieves a high reputation can attempt to deceive other agents by taking advantage of this high reputation. A specific case of this attack is what is known as the “value imbalance exploitation” where the agent provides a number of low value, high-quality services (to get a good reputation) and a small number of deceptive, high-value services. A commonly used technique to make the reputation mechanism robust to dynamic personality attacks is to have a memory window so that not all the past history is taken into account. An even more robust mechanism is to have a dynamic memory window that is shortened when the reputation is lowered. In the case of a “value imbalance exploitation,” the key is to use the value of the service to weight the impact that that specific service will have on the final reputation.

4.4.4 Whitewashing

Whitewashing occurs when an agent changes its identifier in order to escape previous bad feedback. A more sophisticated attack happens when whitewashing is combined with collusion and unfair ratings. A group of malicious agents collude to allow whitewashing by using unfair ratings to increase the reputation of the agent who has just changed personality. This is no different than a Sybil attack (see Section 4.4.6).

4.4.5 Collusion

Collusion occurs when a group of agents cooperate with one another in order to take advantage of the system and other agents. This is not an attack “per se” but an enhancer of other attacks. Collusion is difficult to counter. A possibility is to detect an important and recurrent deviation in the feedbacks of different agents regarding the same targets. If that detection leads to identifying a small group of agents that keeps recommending each other (while other agents provide a different feedback), this group can be suspected of collusion and excluded from the system. However, this detection is hard to perform if there is no global view of the agent society or if there is a high number of colluding agents.

4.4.6 Sybil Attacks

Sybil attacks are a sort of security threat that can be launched in scenarios in which it is easy to create fake identities. The attack consists of creating enough identities so that a single agent can subvert the normal functioning of the system. An example of a Sybil attack would be the improvement of one's reputation through artificial feedback from many Sybil identities.

4.4.7 Reputation Lag Exploitation

All reputation mechanisms have an inertia, that is, when the behavior of an agent changes, it takes some time for the reputation to reflect the new reality. This is necessary to avoid having reputation values that oscillate from one value to another in reacting to the minimum change, and depending on the model, this inertia can be bigger or smaller. This attack consists of using this lag that the reputation mechanism needs to reflect the new reality and exploiting it to obtain benefit. The malevolent agent increases the reputation to a certain point and then starts defrauding – taking advantage of the good reputation while the reputation value is still high. Once the reputation decreases, it starts again to show a good behavior to increase again the reputation, repeating the cycle. The solution to this attack can rely on two aspects. First, adjusting the reaction time of the reputation mechanism so it reacts quickly enough to changes in the behavior. Second, giving the agent the possibility of detecting patterns that show this cyclic behavior in the reputation value.

5 Trust, Reputation, and Other Agreement Technologies

Trust and reputation contribute to agents' representation models and decision processes. It is natural to integrate them into an agent architecture in which they have to be combined with other agent reasoning processes. The combination of trust and reputation concepts with other agreement technologies is promising (and at the same time necessary) as there can be a reciprocal contribution of the technologies. We sketch in this section existing connections between trust and reputation and other subfields of multiagent systems: argumentation, negotiation, norms, organizations, and semantics.

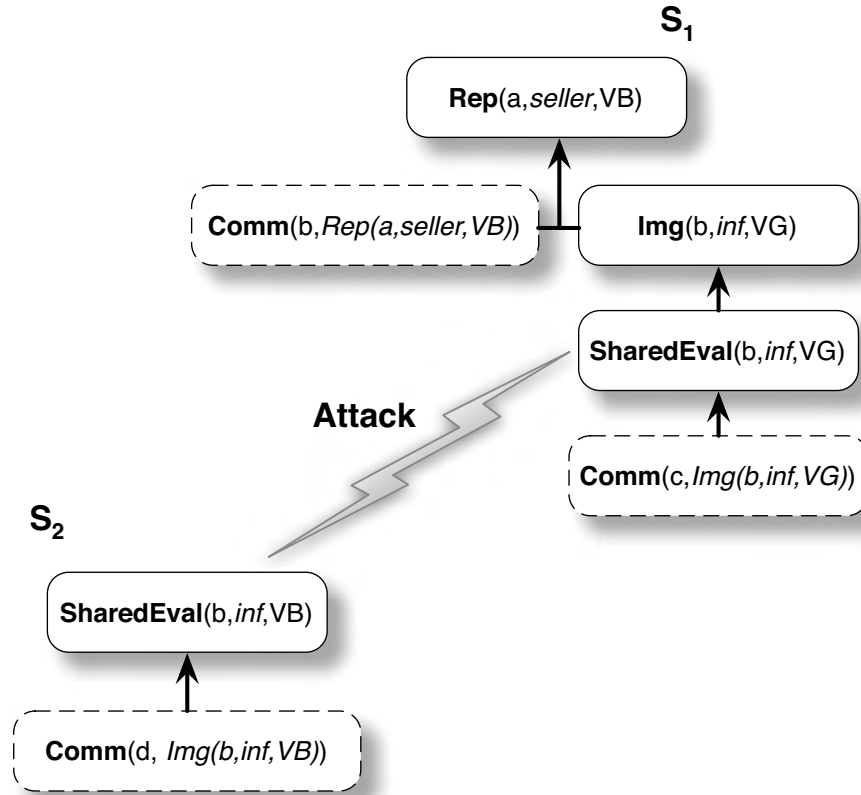


Figure 9.6: Example of argument for a reputation value.

5.1 Argumentation

In a scenario with autonomous entities that have to interact and at the same time maintain self-interest goals, it is normal to have disagreements. This makes it necessary to establish dialogues to try to reach a consensus. This is a central problem in the field of argumentation (see Chapter 5).

We can see the relation of trust and reputation models to argumentation from two different perspectives (that at the same time are complementary):

- **Argumentation for trust/reputation:** Arguments can be used to explain a trust/reputation value. A straightforward approach is to provide to the other agent the “raw” data that has been used to calculate the value [38], where by “raw” data we mean the data that still has not been evaluated by the agent. This raw data can be seen as the argument that gives support to the value. The main drawback of this approach consists of the problem of privacy in providing basic data like contracts or other kinds of sensitive information,

which in very few occasions can be distributed. A more sophisticated approach is to use arguments built using more abstract and generic elements that hide the sensitive information but at the same time give enough information to participate in an argumentation dialogue and reach an agreement [27]. For example, instead of giving the details of the contracts that are responsible for a bad reputation, you can communicate something like the argument S_1 in Figure 9.6, where the exact details (the contracts) are substituted by qualitative and more abstract predicates.

In the figure, we can see in argument S_1 that the reputation regarding agent a as a seller is very bad ($Rep(a, seller, VB)$) and that this value has been obtained (i) from a communication from agent b that says that the reputation of agent a as a seller is very bad ($Comm(b, Rep(a, seller, VB))$) and (ii) from one image that says that agent b is a very good informer ($Img(b, inf, VG)$). The image is supported by a shared evaluation ($SharedEval(b, inf, VG)$) that at the same time is supported by a communication from agent c saying that the image of agent b as an informer is very good ($Comm(c, Img(b, inf, VG))$). In the example, this argument is attacked by argument S_2 , which says that agent b as an informer is very bad ($SharedEval(b, inf, VB)$) and that this is supported by a communication from agent d ($Comm(d, Img(b, inf, VB))$). Notice that the kind of information provided in the dialogue is much more abstract than the transmission of specific interaction details. The nice thing here is that the agent can decide the level of detail that it wants to reveal depending on who the partner is: from high-level reputation and image values as in the example, to low-level contract outcomes where the details of the interaction are made explicit.

- Trust/reputation for argumentation: In this case trust and reputation are not the object of the argumentation process but mechanisms that help to decide about the reliability of the arguments. During an argumentation dialogue about a social evaluation, it can be either that the object of the evaluation is known by the two agents but they disagree on the value (that is the case in the example in Figure 9.6 where there is a disagreement about the quality of b as an informer) or that the attack is based on information that the first agent did not know before (for example new information that reveals that b is not a good informer). In both cases (especially in the latter) the trust in the agents that are sending the information and their reputation as informers are crucial elements that can be used to decide about the acceptance of the argument/attack. Therefore, if the sender of the information is a trustworthy agent, I can accept what it says without having any further certainty or even if the information contradicts what I know.

5.2 Negotiation

As stated in Chapter 4, the capacity to negotiate with other agents is a skill usually demanded of an autonomous agent. Similar to what happens with argumentation, the relation between trust/reputation and negotiation models can be at different levels:

- Trust negotiation was first introduced in trust management systems in the field of computational security. Trust management systems, such as KeyNote [2], propose to assess trust based on credentials provided by the trustee. Negotiation is involved when there is an exchange between the trustor and the trustee to agree on a given set of credentials corresponding to the stake or the risk of the transaction [47]. A matter of concern for the trustee may be to deliver the minimum set of required credentials to preserve as much as possible its privacy [48].
- Trust/reputation models can be used to evaluate a partner as a negotiator. This is not very different from using them to evaluate any other characteristic of that partner. Knowing the trustworthiness of the possible partners as negotiators can help an agent to balance (i) the quality of the service the partners potentially can offer and (ii) the actual possibilities for the agent to obtain that quality after a negotiation process.

5.3 Norms

According to the definition of Marsh [23], trust (and by extension reputation) refers to an expectation about an uncertain behavior. Trust evaluations are built based on the compliance of past behavior of the trustee and are used by the trustor to predict the trustee's future behavior. Even if the expectations are often hidden in the trust calculation function, they can also be made explicit by using *norms*. The use of norms is covered in more depth in Chapter 2, in particular, for situations in which norms are used by electronic institutions in order to control agents' behaviors.

The concept of norms has various implementations in multiagent systems. In the categories of norms identified by Tuomela [44], rules (r-norms for Tuomela) represent hard constraints that have to be respected in a society, social norms (s-norms) consist of preferences shared by a group of individuals, and personal norms (p-norms or m-norms) are individual expectations. Rules are usually enforced by electronic institutions. But this solution requires an intrusive control of agents and the centralization of some tasks by trusted third parties. If such characteristics are not feasible or not desired, trust and reputation acting as social control mechanisms can be used to enforce agents to behave as expected.

Trust is then used to assess the agent's norm obedience [22]. When individual norms are considered, trust evaluations are subjective and performed locally by agents. Contracts may also be used to make explicit the trustor's expectations. The distance between the outcome of a transaction and the initial contract is then an experience to consider as an input of trust calculation (e.g., Regret [35] uses such experiences). With global norms attached to groups or to the whole society, the expectations are attached to a set of agents. Each agent of the group should use trust mechanisms to contribute to social control by supervising the compliance of its neighbors' behavior to rules and social norms. Trust models that attach trust evaluations to given norms [14, 42] are able to represent different trust values about the same trustee and provide a multidimensional evaluation covering local and global norms.

5.4 Organizations

Besides norms, other organizational concepts bring interesting properties when combined with trust and reputation. When agents are organized in *groups*, it is therefore possible to place some information at a group level. Reputation for instance can be attached to an explicit group of agents, and an agent may have different reputations in different groups. The reputation concept is represented in this way in the ForTrust model [16] by the predicate $Reput(I, j, \alpha, \phi, \kappa)$ stating that agent j has the reputation in the group I to achieve ϕ by doing α when the conditions κ hold. An explicit representation of the group I is required to be integrated with the formal representation of reputation.

A generalization of trust assessments can be done using the concept of roles. Trust is then attached to agents but also to general roles. There is a mutual relation between role-based and agent-based trust. An agent's behaviors influence the trust attached to its role but the evaluation of the agent also depends on the trust about its roles. It is then possible to estimate the trustworthiness of an agent without any prior experience with it, by considering the role it is playing. However, it is essential that roles are defined and explicitly attached to agents. Sometimes, roles represent stereotypes of agents and are built dynamically by clustering agents according to exhibited characteristics [15].

5.5 Ontologies and Semantics

We have already mentioned the large number of trust and reputation models available and that this number is increasing quickly. We have seen that they are very different from one another and do not only represent trust and reputation using different mathematical formalisms but also define the essential concepts in different ways. Therefore, it seems clear that when two agents in an open MAS exchange

information regarding trust and reputation, some kind of ontology mechanism is necessary to guarantee that the exchanged information has meaning for both. Expecting that both agents are using the same model is not realistic in open MAS, but even if they are using the same model, there is no guarantee that the information can be directly reused. The fact that agents can have different goals makes the evaluation of an interaction something completely subjective and therefore the trust and reputation values can be based on elements that not necessarily coincide with each partner's interests.

One solution to this problem is the use of a common ontology. Till now there have been only two ontologies [4, 29] specifically oriented to trust and reputation. The ontologies define a set of terms related to trust and reputation unambiguously so every agent that adopts the ontology can use it as a bridge to communicate with the other agents regarding trust and reputation. The ontology of Pinyol et al. [29] also establishes how to make the conversion between different types of trust and reputation values representations. For example, it provides a way to represent a reputation value initially represented as a probability distribution as a real number. This way, a model using internally a real value to represent trust values can use information coming from an agent that uses probability distributions.

This, however, does not solve the problem of subjectivity. Although the agents can share the meaning of what is an image, a reputation, or a shared voice, this does not mean that they also share the scale of values associated with each concept. Moreover, the interests of the agents may be different, and so their evaluations of a particular event may be different as well. Something that is good for one agent, but can be very bad for another. One possible solution, as proposed by Koster et. al. [20], is that the agent aligns its trust model with that of its partner prior to starting to consider social evaluations coming from it. This alignment process however is not an easy task and requires a considerable amount of shared information between the two agents. In order to reduce the amount of previously shared information, another solution is to use argumentation as we saw in the previous section when we were talking about the use of argumentation for trust/reputation. Justifying a communicated value can give clues to the receiver about what the motivation is for that value and if that motivation can adapt to its situation. The idea is to use argumentation dialogues to “understand” why the other agent is giving that social evaluation. By doing that, the agent can decide to modify its own beliefs, adapt the information that it receives, or simply discard that information.

6 Conclusions

There is no doubt that computational trust and reputation models have become an important topic in the area of MASs. Currently it is almost inconceivable to deploy an agent in an open multiagent system without giving that agent the capability of evaluating the trust of partners. At the same time, the nature of a MAS makes necessary mechanisms of social control like reputation to guarantee its good functioning. It is this relevance that resulted in a large number of computational trust and reputation models being proposed in the last few years. However, the fact that this topic is a crossroad of different disciplines (psychology, sociology, cognitive science, artificial intelligence, economics, etc.) makes it difficult and slow to achieve a consensus even on the main definitions.

The initial and mainstream approach was to define mathematical functions computing a single or a small set of trust values from a set of inputs (usually observations of agents' behaviors). Recently this tendency has changed and now much more effort is dedicated to semantical aspects with the objective of building trust and reputation representations that are closer to the real concepts involved in human relations. This is mainly due to the fact that MAS are currently seen as a global system populated not only by artificial entities but also by human beings. Therefore, the old agent that knew about computational protocols and languages and was using black boxes to make the calculations is not yet enough. The new challenge is a new generation of agents that can communicate (in the broad sense of the word) with humans. To achieve that, more cognitive approaches that take into account not only the final value but also the "path" that carries one to that value are appearing. The motivations behind these two approaches mainly depend on the nature of the trust decisions to be made. For instance, if we need an automatic decision that should make accurate predictions, mathematical approaches are relevant. If it is important to be precise in the semantics of the concepts a symbolic socio-cognitive approach seems more suited. It is however nonsense to oppose these two approaches. They should indeed be seen as complementary and modern trust and reputation models should now provide accurate computation mechanisms that lead to the instantiation of well-defined concepts.

Another important challenge is the integration of the trust and reputation models with the rest of the elements of the agent. The black box and reactive approach, where the trust and reputation model is a passive element of the architecture waiting for queries about the trust or reputation value assigned to a possible partner, may be satisfactory for simplistic applications but is clearly insufficient for more complex MAS. We have to see the trust and reputation module as an element that can justify the returned values if it is necessary and participate in the decision making in a proactive way, proposing actions that improve the knowledge the agent has about the society.

This integration has to be also at the level of other agreement technologies: How can the agent use argumentation mechanisms to improve the reliability of trust and reputation models? How does the observance (and violation) of norms affect trust and reputation? How can agents with different models communicate efficiently? These are only a few questions that are still open and that, for sure, will be the focus of an important part of the research on computational trust and reputation models in years to come.

7 Exercises

1. **Level 1** Provide a critical discussion on the advantages/drawbacks of adopting a mathematical approach to trust representation and calculation versus a symbolic one.
2. **Level 1** Enumerate and describe the inputs of a trust calculation process resulting in the generation of trust evaluations.
3. **Level 1** What is the problem of correlated evidence? How can it be solved?
4. **Level 1** Explain the difference between *shared image* and *reputation*.
5. **Level 2** Gossip or recommendations coming from other agents are sometimes used for trust calculation. However, they are a particular source of information that bring specific risks. Explain why it is interesting to take them into account and what the associated risks are. Propose an efficient way to integrate them into a trust calculation process.
6. **Level 2** A distinction between dispositional trust and occurrent trust is done in the ForTrust model. Explain why it is interesting to distinguish the two and how they are linked.
7. **Level 2** Design a heuristic for the aggregation function like that in Section 4.1.4 to obtain a reputation evaluation that takes into account the context.
8. **Level 3** Trust and reputation models are meant to be used in an agent reasoning process besides other agreement technologies. Explain how they can be combined with argumentation techniques and what the contribution of trust/reputation is for argumentation and vice versa.
9. **Level 3** Design a trust model that takes into account the elements depicted in Figure 9.4 to calculate trust evaluations. Consider that the agents use

integers $[-1, 1]$ to represent the evaluations where -1 is complete distrust and 1 complete trust.

10. **Level 4** Evaluating and comparing existing trust models is a difficult task for which there is not yet any satisfactory solution. Describe the main difficulties that prevent the development of an evaluation tool and propose your ideas to overcome them.
11. **Level 4** Using a multiagent platform, implement a centralized reputation mechanism that represents reputation as a probability distribution. Then implement a few agents that feed the system with image values and use the calculated reputation to select a partner.
12. **Level 4** Implement an agent that will send unfair ratings to the centralized reputation mechanism implemented in the previous exercise. Extend this agent implementation in order to simulate a collusion in which agents use unfair ratings to recommend each other. Try several executions of a MAS composed of a different number of attackers to estimate the proportion of agents for which your centralized reputation mechanism does not provide reliable reputation values.

References

- [1] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, volume 6, page 6007, Washington, DC, USA, 2000. IEEE Computer Society.
- [2] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proceedings of the 1998 Security Protocols International Workshop*, volume 1550, pages 59 – 63, Cambridge, England, April 1998. Springer LNCS.
- [3] J. Carbo, J. M. Molina, and J. Davila. Trust management through fuzzy reputation. *Int. Journal in Cooperative Information Systems*, 12(1):135–155, 2003.
- [4] Sara J. Casare and Jaime S. Sichman. Using a functional ontology of reputation to interoperate different agent reputation models. *Journal of the Brazilian Computer Society*, 11(2):79–94, November 2005.
- [5] Cristiano Castelfranchi. Engineering social order. In *Proceedings of the 2nd International Workshop on Engineering Societies in the Agent's World (ESAW'00)*, 2000.

- [6] Cristiano Castelfranchi and Rino Falcone. *Trust Theory: A Socio-Cognitive and Computational Model; electronic version*. Wiley Series in Agent Technology. John Wiley & Sons Ltd., Chichester, 2010.
- [7] R. Conte and M. Paolucci. *Reputation in Artificial Societies: Social Beliefs for Social Order*. Kluwer Academic Publishers, 2002.
- [8] Chrysanthos Dellarocas and Charles A. Wood. The sound of silence in online feedback: Estimating trading risks in the presence of reporting bias. *Manage. Sci.*, 54:460–476, March 2008.
- [9] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, November 1976.
- [10] eBay. *eBay*. <http://www.eBay.com>, 2011.
- [11] Epinions. *Epinions*. <http://www.epinions.com>, 2011.
- [12] Karen Fullam, Tomas Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, Suzanne Barber, Jeffrey Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the Agent Reputation and Trust (ART) testbed: Experimentation and competition for trust in agent societies. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005)*, pages 512–518, Utrecht, Netherlands, July 2005. ACM Press.
- [13] Diego Gambetta. *Can We Trust Trust?*, pages 213–237. Basil Blackwell, 1988.
- [14] Amandine Grizard, Laurent Vercouter, Tiberiu Stratulat, and Guillaume Muller. A peer-to-peer normative system to achieve social order. In V. Dignum, N. Fornara, and P. Noriega, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems*, volume LNCS 4386 of *Lecture Notes in Computer Science*, pages 274–289, Berlin, Germany, 2007. Springer-Verlag.
- [15] Ramón Hermoso, Holger Billhardt, and Sascha Ossowski. Role evolution in open multi-agent systems as an information source for trust. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, volume 1, pages 217–224, 2010.
- [16] Andreas Herzig, Emiliano Lorini, Jomi F. Hübner, and Laurent Vercouter. A logic of trust and reputation. *Logic Journal of the IGPL, Normative Multiagent Systems*, 18(1):214–244, February 2010.
- [17] Trung Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.

- [18] Audun Jøsang and Jennifer Golbeck. Challenges for robust of trust and reputation systems. In *5th International Workshop on Security and Trust Management (STM 2009)*, 2009.
- [19] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43:618–644, March 2007.
- [20] Andrew Koster, Jordi Sabater-Mir, and Marco Schorlemmer. Trust alignment: a sine qua non of open multi-agent systems. In Robert Meersman, Tharam Dillon, and Pilar Herrero, editors, *Proceedings of On the Move to Meaningful Internet Systems (OTM 2011, Part I)*, volume 7044 of *LNCS*, pages 182–191, Herssonissos, Greece, 2011. Springer.
- [21] Eleni Koutrouli and Aphrodite Tsalgatidou. Reputation-based trust systems for P2P applications: Design issues and comparison framework. In Simone Fischer-Hübner, Steven Furnell, and Costas Lambrinoudakis, editors, *Trust and Privacy in Digital Business*, volume 4083 of *Lecture Notes in Computer Science*, pages 152–161. Springer Berlin / Heidelberg, 2006. 10.1007/11824633_16.
- [22] Emiliano Lorini and Robert Demolombe. Trust and norms in the context of computer security: Toward a logical formalization. In R. van der Meyden and L. van der Torre, editors, *International Workshop on Deontic Logic in Computer Science (DEON)*, volume 5076 of *LNCS*, pages 50–64. Springer-Verlag, 2008.
- [23] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Department of Mathematics and Computer Science, University of Stirling, 1994.
- [24] Stephen Marsh and Pamela Briggs. Examining trust, forgiveness and regret as computational concepts. In Jennifer Golbeck, editor, *Computing with Social Trust*, Human-Computer Interaction Series, pages 9–43. Springer London, 2009.
- [25] Nardine Osman, Carles Sierra, and Jordi Sabater-Mir. Propagation of opinions in structural graphs. In *19th European Conference on Artificial Intelligence (ECAI-10)*, 2010.
- [26] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
- [27] Isaac Pinyol. *Milking the Reputation Cow: Argumentation, Reasoning and Cognitive Agents*. Number 44 in Monografies de l’Institut d’Investigació en Intel·ligència Artificial. IIIA-CSIC, 2011.
- [28] Isaac Pinyol, Mario Paolucci, Jordi Sabater-Mir, and Rosaria Conte. Beyond accuracy. Reputation for partner selection with lies and retaliation. In *Multi-Agent-Based Simulation VIII*, volume 5003, pages 128–140. Springer, 2008.

- [29] Isaac Pinyol, Jordi Sabater-Mir, and Guifre Cuni. How to talk about reputation using a common ontology: From definition to implementation. In *Ninth Workshop on Trust in Agent Societies*, pages 90–102, 2007.
- [30] Isaac Pinyol, Jordi Sabater-Mir, Pilar Dellunde, and Mario Paolucci. Reputation-based decisions for logic-based cognitive agents. *Autonomous Agents and Multi-Agent Systems*, 2010. 10.1007/s10458-010-9149-y.
- [31] S. D. Ramchurn, T. D. Huynh, and N. R. Jennings. Trust in multiagent systems. *The Knowledge Engineering Review*, 19(1):1–25, 2004.
- [32] Lars Rasmusson and Sverker Jansson. Simulated social control for secure Internet commerce. In *Proceedings of the 1996 Workshop on New Security Paradigms*, NSPW '96, pages 18–25, New York, NY, USA, 1996. ACM.
- [33] Martin Reháč and Michal Pěchouček. Trust modeling with context representation and generalized identities. In Matthias Klusch, Koen Hindriks, Mike Papazoglou, and Leon Sterling, editors, *Cooperative Information Agents XI*, volume 4676 of *Lecture Notes in Computer Science*, pages 298–312. Springer Berlin / Heidelberg, 2007.
- [34] D. H. Ruben. The existence of social entities. *Philosophical Quarterly*, 32:295–310, 1982.
- [35] Jordi Sabater. *Trust and Reputation for Agent Societies*. Number 20 in Monografies de l’Institut d’Investigació en Intel·ligència Artificial. IIIA-CSIC, 2003.
- [36] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24:33–60, September 2005.
- [37] Jordi Sabater-Mir, Mario Paolucci, and Rosaria Conte. Repage: REPUtation and imAGE among limited autonomous partners. *Journal of Artificial Societies and Social Simulation (JASSS)*, 9(2):3, 2006.
- [38] Michael Schillo, Petra Funk, and Michael Rovatsos. Using trust for detecting deceitful agents in artificial societies. *Applied Artificial Intelligence*, 14(8):825–849, 2000.
- [39] Sandip Sen, Anish Biswas, and Sandip Debnath. Believing others: Pros and cons. *Artificial Intelligence*, 142:279–286, 2000.
- [40] Sandip Sen and Neelima Sajja. Robustness of reputation-based trust: Boolean case. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS’02)*, volume 1, pages 288–293. ACM, 2002.
- [41] Carles Sierra and J. Debenham. Information-based agency. In *Twentieth International Joint Conference on AI, IJCAI-07*, pages 1513–1518. AAAI Press, 2007.

- [42] Viviane Torres Silva, Ramón Hermoso, and Roberto Centeno. A hybrid reputation model based on the use of organizations. In Jomi Fred Hübner, Eric Matson, Olivier Boissier, and Virginia Dignum, editors, *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 111–125. Springer-Verlag, Berlin, Heidelberg, 2009.
- [43] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, March 2006.
- [44] Raimo Tuomela. *The Importance of Us: A Philosophical Study of Basic Social Norms*. Stanford University Press, 1995.
- [45] Laurent Vercouter, Sara J. Casare, Jaime S. Sichman, and Anarosa A. F. Brandao. An experience on reputation models interoperability based on a functional ontology. In *20th International Joint Conference on Artificial Intelligence 2007*, Hyderabad, India, January 2007.
- [46] Laurent Vercouter and Guillaume Muller. L.I.A.R.: Achieving social control in open and decentralised multi-agent systems. *Applied Artificial Intelligence*, 24(8):723–768, September 2010.
- [47] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume 1, 2000.
- [48] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.
- [49] Bin Yu and Munindar P. Singh. An evidential model of distributed reputation management. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 294–301. ACM Press, 2002.
- [50] Giorgios Zacharia. Collaborative reputation mechanisms for online communities. Master’s thesis, Massachusetts Institute of Technology, September 1999.

Part IV

Distributed Cognitive Abilities

Chapter 10

Multiagent Learning

Karl Tuyls and Kagan Tumer

1 Introduction

One of the key properties attributed to an agent operating in an unknown environment is its ability to learn from its experiences. For a single-agent system, this generally consists of building a mapping from the agent's inputs (sensor reading and internal state) to an output (action). How that mapping is constructed depends on many factors, and there are many algorithms that have been extensively studied, including learning automata [49], reinforcement learning [72], neuro-evolutionary algorithms [26] and biologically inspired methods [12].

When extending such algorithms to multiagent learning, two new key issues arise: how do agents account for the collective action of other agents in the system, and how do agents select actions that not only provide a direct benefit but also shape the actions of other agents in the future? The first issue addresses an agent's perception, in that it forces an agent to differentiate between the potentially stochastic changes to an environment and the actions of intelligent agents and to exploit this knowledge. The second issue addresses an agent's impact, in that it forces an agent to select actions that lead to desired behavior through its interactions with other agents' actions. These two issues together lead to both theoretical (convergence) and practical (signal to noise in rewards) complications and render the direct application of single-agent learning algorithms problematic.

Yet, multiagent learning must solve these problems because it is a fundamental component of multiagent systems both from scientific and engineering perspectives. From a scientific perspective, studying the interactions among learning

agents provides insights into many social phenomena from game theory to commodities trading to resource allocation problems. From an engineering perspective, learning agents provide a conceptually proven approach to distributed control problems such as load balancing, sensor networks, multirobot coordination, and air traffic management. The benefits of understanding the dynamics of multiagent systems are numerous, in that it would provide desirable system characteristics, including:

- **Robustness:** A multiagent approach removes the single point of failure view of a centralized system; if one or a few components fail, the system still should operate properly;
- **Efficiency:** A multiagent approach allows tasks to be assigned to independent agents, which can then pursue those goals in parallel;
- **Reconfigurability:** A multiagent approach is modular by definition and components can be added or removed as needed;
- **Scalability:** A multiagent system removes the need for full information flow in both the sensory and action directions, and hence multiagent systems generally scale better than centralized systems.

For a multiagent system to realize those potential gains, agents need to interact with each other and quickly adapt to changing environments and the changing strategies of their peers.

As a consequence, machine learning algorithms are crucial to development and deployment of adaptive multiagent approaches [4, 19, 33, 35, 43, 62, 64, 70, 73, 77, 79, 80, 83, 87]. The subfield of multiagent learning studies agent definitions, algorithms, interactions, and reward structures to create adaptive agents that can function in environments where their actions shape and are shaped by the actions of other agents. Though most multiagent learning algorithms are based on traditional single-agent learners (e.g., reinforcement learning), they need to be modified to effectively deal with the challenges stemming from the interaction among multiple independent agents.

In this chapter we introduce the basics of multiagent learning, present its challenges and approaches and provide examples of domains that can benefit from such approaches. The remainder of the chapter is organized as follows.

Section 2 elaborates on the main differences between single and multiagent learning algorithms, and highlights the challenges that multiagent learning algorithms must address. In Section 3 we concisely introduce the necessary background on single-agent reinforcement learning, present the multiagent Markov decision process formulation, and introduce a number of state-of-the-art multiagent

reinforcement learning algorithms. Section 4 presents evolutionary game theory as a multiagent learning paradigm, with an elaborate discussion on the formal link with multiagent reinforcement learning and the evolutionary analysis of several reinforcement learning algorithms. Section 5 presents swarm intelligence as a multiagent learning paradigm. Section 6 presents the use of neuro-evolutionary algorithms as a multiagent learning paradigm, and includes a short description on the link between multiagent reinforcement learning and neuro-evolutionary approaches. Section 7 presents a case study of the successful application of multiagent learning to air traffic control. Finally, Section 8 concludes the chapter.

2 Challenges in Multiagent Learning

The extension of basic learning paradigms from single to multiagent systems poses many challenges. For example, reinforcement learning – one of the key methods for single and multiagent learning – has a well understood framework and theoretical guarantees that are based on a single agent interacting with a (mostly) static environment [36, 72]. Though deviations from this assumption exist, particularly in real-world applications of reinforcement learning, the theory stretches to a breaking point when multiple agents learn in the same environment and where each agent's actions modify the state of other agents in a non-predictable way. The convergence guarantees no longer hold and there exists no general formal theory describing and elucidating the conditions under which algorithms for multiagent learning (MAL) are successful. It is currently an open question how to scale up MAL, i.e., how to efficiently handle many states, many actions, and many agents.

Though treating the actions of all other agents as part of the environment is an acceptable approach in some simple multiagent learning problems, in general, an agent needs to account for those actions. The key reason for this is the new dynamics created by the actions of the other agents in the system, particularly if all the agents are learning as a response to the actions of all the other agents in the system. In this section we first discuss the key characteristics and challenges in multiagent learning, including the state space explosion, the differences in system dynamics in cooperative and competitive multiagent systems, the multiagent credit assignment problems, and the care that must be taken in determining the reward structure of agents operating in a multiagent system. We then revisit the learning problem and provide two simple learning algorithms as starting points for the rest of the chapter.

2.1 State, Action, and Outcome Space Problems

Learning is fundamentally a search problem through possible solutions, and therefore increases in the number of variables (e.g., agents, states, actions, outcome states) exponentially complicate the learning task. In particular, for multiagent learning, three key computational problems stand out: the state space explosion, the joint action space explosion, and the result-state (or outcome state) explosion [56].

First, the state space grows exponentially in the number of agents and environment features, significantly increasing the time required to determine the desirability of each state (e.g., value functions in reinforcement learning). Second, the space of possible actions grows exponentially in the number of agents, making looking ahead for joint actions difficult for a small number of time steps and all but impossible for many real-world sequences. Third, the number of joint action outcomes grows exponentially with the number of agents, making the desirability of the next state difficult to compute exactly. These three issues are referred together as the three “curses of dimensionality” [56]. Though these problems are particularly pronounced in classical reinforcement formulations such as value iteration and policy iteration (see Section 3), they are present in all learning methods since they are caused by the process of searching for solutions in large spaces in stochastic and dynamic environments.

2.2 Multiagent Credit Assignment Problem

The basic problem a learning agent faces is the *credit assignment* problem of determining which of its actions resulted in the feedback received. When a learning agent interacts with its surroundings, it adjusts its internal parameters based on the feedback it receives from the environment. As long as the feedback follows every action, the agent can directly associate each response to its actions, and the credit assignment is trivially solved. However, most problems have *delayed feedback* mechanisms where a sequence of actions needs to be performed before the agent receives feedback. This situation creates the *temporal credit assignment* problem of how to assign a reward received at the end of a sequence to each action in the sequence, a problem which has been extensively studied for single-agent systems [22, 72, 74, 76, 96].

In multiagent learning, a second credit assignment problem appears: the *structural credit assignment* problem of how to assign credit to a particular agent based on the performance of a set of agents. For systems with few agents, this credit assignment problem can be sidestepped, and all agents can use the full system feedback directly. However, when the number of agents in a system increases, this method breaks down and agents need to receive a reward that ac-

counts for their contribution to the system. This problem has also been studied [4, 47, 81, 89, 90, 91, 92, 98].

In most multiagent problems, these two structural problems are confounded and each agent needs to not only determine its contribution to the full system performance, but also determine the relative impact of each of its actions in a sequence. Solving both these credit assignment problems at once is difficult since each reward is a function of actions from different agents over different time steps. Temporal difference methods, such as Q-learning, are an important class of methods for addressing the temporal credit assignment problem for single-agent reinforcement learning. Unfortunately they do not address the structural credit assignment problem present in multiagent problems and can have very low performance when there are many agents.

Because it is essential for designing the proper agent rewards, this credit assignment problem is critical in both cooperative and competitive multiagent systems. In the former, it allows for the design of agent rewards that explicitly seek to maximize a global reward (e.g., win the soccer game in robotic soccer, maximize the information gathered in multirobot exploration). In the latter, it crystalizes the competition for shared resources and allows the creation of appropriate incentive structures to improve system performance (e.g., overall improved traffic congestion).

2.3 Agent Rewards and System Dynamics

As mentioned earlier, the performance of a learning multiagent system greatly depends on the credit assignment structure used to shape its behavior. In competitive settings, agent rewards represent inherent desires of the agents and are generally set. The key then is to determine agent incentives and interactions that lead to desirable outcomes. In cooperative settings, however, the agent rewards can be modified directly and have a large impact on the agent interactions, equilibrium points, and convergence. In this section, we focus on cooperative systems and evaluate key characteristics of agent rewards and their impact on system behavior.¹

Indeed, the impact of the reward structure is particularly pronounced when multiple agents that need to exhibit coordinated behavior are interacting and learning simultaneously [4, 5, 33, 45, 71, 81, 102]. Given a reward that evaluates the performance of the full system, there are many ways in which agent rewards can be selected. In very simple systems, agents may simply use the system reward

¹In this section we will refer to agent “reward” functions, though for the conceptual description in this section, one can replace “reward” with “objective,” “evaluation,” “payoff,” or “utility” and retain the same content.

directly [21]. In more complex systems, agents may need local rewards that increase their ability to learn good policies. Unfortunately this “reward shaping” problem becomes more difficult in domains involving continuous state spaces, and dynamic and stochastic environments. For example, reward structures that have been shown to perform well in static environments do not necessarily lead to good system behavior in dynamic environments.

One way to assess the impact of a reward structure is to have reward characteristics that are independent of the domains and learning algorithms used. Two such characteristics measure reward “alignment” and “sensitivity,” which measure necessary, but often conflicting, properties.

- **Alignment** – formalized as “factoredness” [4, 81] – defines how well two rewards are matched in terms of their assessment of the desirability of particular actions. Intuitively, high values of factoredness mean that actions that optimize one reward will also optimize the other reward. In that sense, having agent rewards have high factoredness with respect to the system reward results in a system where agents that aim to improve their own performance also tend to improve system performance.
- **Sensitivity** – formalized as “learnability” [4, 81] – defines how discernible the impact of an action is on an agent’s reward function. Intuitively, the higher the learnability, the more the agent’s reward depends on its own actions. Therefore, higher learnability means it is easier for an agent to take actions (changing its state) that maximize its reward. When learnability is too low, many other agents are affecting the reward; therefore it is hard for an agent to discern the impact of its actions from the impact of all of the other agents’ actions.

Based on these characteristics, let us analyze three potential agent rewards for a given system, as system performance hinges on balancing the degree of factoredness and learnability for each agent. In general, a reward with high factoredness will have low learnability and a reward with high learnability will have low factoredness [102]. Consider the following three reward functions for an agent:

1. **Full system reward:** Each receives the full system reward. By definition it is fully factored, meaning every action that is good for the agent is good for the system. This reward has been used successfully on multiagent problems with few agents [21]. However, since each agent’s reward depends on the actions of all the other agents, it generally has poor learnability, a problem that gets progressively worse as the size of the system grows. This is because, when an agent’s reward changes, it is difficult for the agent to determine whether that change was caused by its own action or by the actions

of the other agents in the system. It is therefore only suited for problems with a small number of agents.

2. **Local reward:** Each agent receives the local component of the full system reward. In contrast to the global reward, which requires full state information, the local reward is composed of the components of the global reward, which depend on the states of agent i . Because it does not depend on the states of other agents, this reward is “perfectly learnable.” However, depending on the domain, it may have a low degree of factoredness, in that having each agent optimize its own performance may or may not promote coordinated system level behavior.
3. **Difference reward:** Each agent receives a reward that measures its impact. This reward is based on the difference between the system reward and the system reward that would have resulted had the agent performed some “null” action – hence the name [5, 77, 79, 81, 102]. This reward has generally high factoredness as the removed “null” action term does not depend on the agent’s action. As a result, any impact an agent has on this reward comes from its impact on the global reward. Hence, any actions that improve/deteriorate the difference reward also improve/deteriorate the system reward. Furthermore, this reward usually has better learnability than the system reward because subtracting out the null action term removes some of the effects of other agents (i.e., noise) from the agent’s reward. While having good properties, this reward can be impractical to compute because it requires knowledge about the system state. The formal definition and application of this reward is presented in the air traffic case study in Section 7.

The key then for devising good agent rewards is to balance the individual agent’s learning with its coordination with other agents. Since directly using the system reward is impractical in most real-world cases, shaping the agent reward is critical for both local and difference rewards. In both cases, how the missing information is handled (ignored, estimated) determines the trade-off between factoredness and learnability, which in turn directly determines the system performance [4, 79].

2.4 Two Simple Multiagent Learning Paradigms

Having introduced many key challenges in the preceding sections, we now define the multiagent learning problem as the problem of devising learning algorithms for agents that are capable of learning (sub)optimal solutions in the presence of other (learning) agents (or algorithms) – facing the difficulties of incomplete information, large state spaces, credit assignment, cooperative and/or competitive

settings, and reward shaping. We consider these difficulties as the main *complexity factors* behind agents that learn concurrently.

The settings we follow in this book chapter are that of stochastic or Markov games (see Section 3.5), evolutionary game theory (see Section 4), swarm intelligence (see Section 5), and neuro-evolutionary algorithms (see Section 6). In this section we describe two simple learning algorithms, action-value learning and cross learning (a kind of learning automaton [13]), as an introduction to the reinforcement learning approach to multiagent learning presented in Section 3.

2.4.1 Action-Value Learning

First, let us introduce the n -armed bandit problem, where the agent has to repeatedly choose one of n actions, each of which results in a reward. The rewards come from a fixed distribution and the agent's objective is to maximize its long-term reward, by learning the true "value" of each action. This is called the n -armed bandit problem based on its similarity to a slot machine [72]. The learning agent selects an action at each time step and maintains an estimate of the mean reward of each action by averaging the rewards received so far by pulling arm a_i . This estimate for each action a_i can be denoted as follows:

$$Q_t(a_i) = \frac{r_1 + r_2 \dots + r_{k_i}}{k_i} \quad (10.1)$$

where k_i is the total number of times action a_i has been selected. r_i are the rewards received when choosing action a_i over the different trials. t is the current time step and also represents the total number of times an action has been selected so far. In this manner, the agent builds an accurate "model" of its world, having each value represent the potential reward for that action. This simple learning method is called "action-value" learning as it is based on learning the "value" of each action, allowing the agent to select the action that will maximize its long-term reward.

A direct extension of this problem to multiagent learning is when multiple agents aim to learn the same problem. Nominally, this can be treated as each agent learning independently. Some recent work aims to allow information sharing between agents, where either the values can be shared [28], or the agents can learn from the actions of other agents by taking counterfactual actions [79]. A far more interesting situation arises when the probability distributions for the arms are not fixed, but depend on the actions of the other agents. The multnight bar problem provides such an example [80]. In this problem, each agent needs to select one of n nights to attend a bar, where the bar has a capacity c . If too few or too many agents attend any one night the reward drops. The action-value learning in this situation would have each agent associate a value for attending a particular night, but

this value now tracks a non-stationary process. Though theoretically, convergence isn't guaranteed as in the fixed probability case, a simple action-value approach has been shown to work and even track changes in behavior by a large number of agents [80].

2.4.2 Direct Policy Adjustment

Second, let us discuss an alternative learning method based on directly adjusting the agent's policy. The cross learning model is a good example of such an approach, which consists of multiple agents playing the same game (i.e., a normal form game) repeatedly in discrete time. At each point in time, each agent has a probability distribution over its action set, which indicates how likely it is to play any of its actions. That probability distribution constitutes the "policy," that is, a plan to choose actions. The probabilities change over time in response to experience. At each time step (indexed by t), a player chooses one of its actions based on the probabilities that are related to each isolated action. In response to an action, a player receives a reward, expressing how good the chosen action was. For illustration, we consider a multiagent situation with two players to simplify the representation. The first agent can choose from s actions, while the second agent can choose from r actions.

The agents do not know each other's actions and rewards and play the situation repeatedly. After observing the outcome of a round (observing the joint action) and receiving a reward at each stage, they update their probability vector or policy. For the first agent we denote this by vector p : $p(t) = (p_1(t), \dots, p_s(t))$, and for the second agent we denote this by q : $q(t) = (q_1(t), \dots, q_r(t))$. These vectors are updated according to

$$p_i(t+1) = r_{ij}^1 + (1 - r_{ij}^1)p_i(t) \quad (10.2)$$

and

$$p_{i'}(t+1) = (1 - r_{ij}^1)p_{i'}(t) \quad (10.3)$$

where $0 \leq r_{ij} \leq 1$ and r_{ij}^1 is the reward agent 1 receives when it chooses its i -th action and agent 2 chooses its j -th action. Equation 10.2 expresses how the probability of the selected strategy a_i is updated and Equation 10.3 expresses how all the other strategies $a_{i'}$ for which $i' \neq i$, and were not selected, are updated for the first agent. If this player p played action a_i in the t th repetition of the game, and if it received reward r_{ij} , then it updates its policy by taking a weighted average of the old policy, and of the unit vector, which puts all probability on strategy a_i . The probability vector of $q(t)$ is updated in an analogous manner.

This simple learning model easily gets stuck in non-optimal solutions when applied to multiagent coordination problems. For example, returning to the multi-night bar problem, agent strategies can consist of simple rules such as "at time

step t , take the action that agent k took at time step $t - 1$,” or “at time step t do the action you performed at time step $t - 2$,” or “at time step t take the action that yielded the highest reward in the last three time steps.” Each of these is a policy and the agent can update its probability vector as described above. However, as the case with the action-value approach, this method need not converge as the outcome of the policies depend on the actions of other agents. In both cases, the simple learning approaches provide a method for learning to take actions and tracking a non-stationary process, but cannot provide guarantees of convergence.

In the next section, we discuss the extensions of these simple learning concepts to more diverse and complex situations encompassing multiple states, delayed rewards, and the general Markov decision context.

3 Reinforcement Learning for Multiagent Systems

In this section we introduce the basics of single-agent reinforcement learning, the MDP framework, and model-free and model-based approaches. Then we make the transition to multiagent reinforcement learning and introduce Markov games, the elementary framework in which multiagent reinforcement learning typically is studied. We end with a summary of some of the state-of-the-art multiagent reinforcement learning algorithms.

Reinforcement learning (RL) is based on the simple observation that rewarding desirable behavior and discouraging undesirable behavior leads to behavioral change. Figure 10.1 shows the basic concept where at time step t , the agent is in state s_t and takes action a_t . This results in the agent receiving reward r_t and moving to state s_{t+1} [72]. The objective of a reinforcement learner is to discover a policy, i.e., a mapping from states to actions, so as to maximize the reinforcement signal it receives. The reinforcement signal is a scalar value, which is usually negative to express a punishment, and positive to indicate a reward. The two simple learning paradigms presented in Section 2.4 are rudimentary reinforcement learning algorithms.

Unlike supervised learning techniques, reinforcement learning methods do not assume the presence of a teacher who can provide the correct action in a particular situation. Instead the learner finds out what the best actions are by trying them out and by receiving a signal on the consequences of its actions. For this reason, reinforcement learning is considered a “semi-supervised” learning technique. For many problems the consequences of an action are not immediately apparent after performing the action, but only after a number of other actions have been taken. In other words, the selected action may not only affect the immediate reward or punishment the learner receives, but also the reinforcement it might receive in the future.

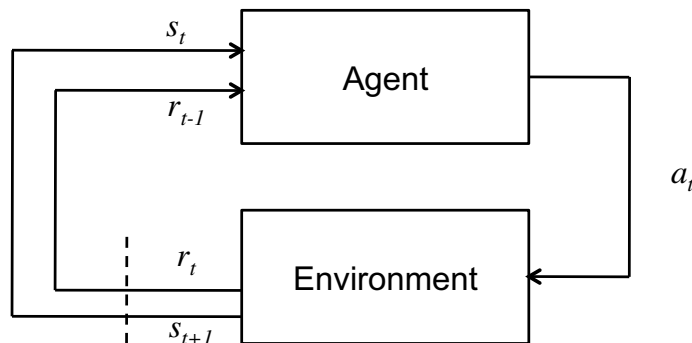


Figure 10.1: Basic reinforcement learning scheme: At time step t , the agent is in state s_t and takes action a_t . This results in the agent receiving reward r_t and moving to state s_{t+1} [72].

3.1 Markov Decision Processes

Most single-agent RL research is based on the framework of Markov decision processes (MDPs) [58]. MDPs are sequential decision-making problems for fully observable worlds. MDPs are defined by a tuple $\langle S, A, T, R \rangle$, where S is a finite set of states and A is a finite set of actions available to the agents. An MDP respects the *Markov property*: the future dynamics, transitions, and rewards fully depend on the current state, i.e., an action a in state $s \in S$ results in state s' based on a transition matrix function $T : S \times A \times S \rightarrow [0, 1]$. The probability of ending up in state s' after doing action a in state s is denoted as $T(s, a, s')$. For all actions a , and all states s and s' , we have that $0 \leq T(s, a, s') \leq 1$, and $\sum_{s' \in S} T(s, a, s') = 1$. The reward function $R : S \rightarrow \mathbb{R}$ returns the reward $R(s, a)$ after taking action a from state s .

The transition function T and reward function R together are often referred to as the model of the environment. The learning task in an MDP is to find a policy $\pi : S \rightarrow A$ for selecting actions with maximal expected (discounted) future reward. The quality of a policy is indicated by a *value function* V^π . The value $V^\pi(s)$ specifies the total amount of reward that an agent may expect to accumulate over the future, starting from state s and then following the policy π . Informally, the value function indicates the long-term desirability of states or state-action pairs after taking into account the states that are likely to follow, and the rewards available in those states. In a discounted infinite horizon MDP, the expected cumulative reward (i.e., the value function) is denoted as:

$$V^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s \right] \quad (10.4)$$

A $\gamma \in [0, 1)$ is introduced to ensure that the rewards returned are bounded (finite) values. The variable γ determines the relevance of future rewards in the update. Setting γ to 0 results in a *myopic* update (i.e., only the immediate reward is optimized), whereas values closer to 1 will increase the contribution of future rewards in the update.

The value for a given policy π , expressed by Equation 10.4, can iteratively be computed by the *Bellman Equation* [9]. One typically starts with arbitrarily chosen value functions, and at each iteration for each state $s \in S$, the value functions are updated based on the immediate reward and the current estimates of V^π :

$$V_{t+1}^\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_t^\pi(s') \quad (10.5)$$

The process of updating state value functions based on current estimates of successor state values is referred to as *bootstrapping*. The depth of successor states considered in the update can be varied, i.e., one can perform a shallow bootstrap where one only looks at immediate successor states or a deep bootstrap where successors of successors are also considered. The value functions of successor states are used to update the value function of the current state. This is called a *backup* operation. Different algorithms use different backup strategies, e.g., sample backups (sample a single successor state) or full backups (sample all successor states).

The goal of an MDP is to find the optimal policy, i.e., the policy that receives the most reward. The optimal policy $\pi^*(s)$ is such that $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in S$ and all policies π .

3.2 Action Selection and Exploration-Exploitation Dilemma

Action selection in reinforcement learning is generally based on a stochastic process. Given estimates of the values of each action, the question becomes how to select future actions. Through exploration the reinforcement learner discovers new actions and their potential value (i.e., rewards and future states) and uses this to improve its policy. An important issue that occurs is the exploration-exploitation dilemma, i.e., when to cease (or slow down) exploration and to start exploiting acquired knowledge. One way to proceed is to select the action with the maximum value so far, i.e., select action a^* for which $Q_t(s, a^*) = \max_i Q_t(s, a_i)$. This means that we under all circumstances choose the action with the highest estimate, i.e., the *greedy* action. This approach does not *explore* actions that are

less good at that moment to check whether they might be better in the long term and so is prone to yield suboptimal solutions.

An alternative is to behave *greedily* most of the time but once in a while select a random action to make sure we do not miss the better actions in the long term. This approach is called the ϵ -*greedy* exploration method in which the greedy option is chosen with high probability $1 - \epsilon$ and with a small probability ϵ a random action is selected. The benefit of this method is that when we play a sufficiently large number of iterations every action will be sufficiently sampled to guarantee that we have learned for all actions a_i its true value $Q_i^*(s, a_i)$. This ensures that an agent learns to play the optimal action in the long term. For a thorough overview, we refer to [101]. Though annealing can be used for ϵ as well, in practice this is often unnecessary.

Another alternative is to use a “softmax” approach, or Boltzmann exploration, where the good actions have an exponentially higher probability of being selected and the degree of exploration is based on a *temperature* parameter τ . An action a_j is chosen with probability:

$$p_j = \frac{e^{\frac{Q(s, a_j)}{\tau}}}{\sum_i e^{\frac{Q(s, a_i)}{\tau}}} \quad (10.6)$$

The selection of the temperature parameter is used to balance exploration and exploitation. Low values of τ increase the probability of selecting the best action, whereas high values of τ make all action probabilities converge. In many cases, starting with high values of τ and reducing this parameter (also called annealing) provides the best results (the agent tends to select actions associated with higher utilities when τ is low). (Note that in the literature τ occurs in the nominator as well as in the denominator.)

3.3 Model-Free and Model-Based Approaches

When the model of the environment is unknown, as it usually is, reinforcement learning can be used to directly map states to actions. Model-free RL does not depend on having a model of state transitions and rewards, but rather collects samples from the environment to estimate that model. In particular when multiple actions need to be taken before a reward is received, the reward received needs to be temporally distributed to the set of actions that preceded it. This is the basis of learning, and Q-learning [95] and SARSA [60] are the most commonly used examples of model-free temporal difference learning algorithms. The algorithms are described in detail in [72]. The *one-step* Q-learning algorithm is summarized in Algorithm 10.1, and the updating of the Q-values of the state action pairs is given by:

```

1 Initialize  $Q(s, a)$  arbitrarily
2 Initialize  $s$  to any starting state
3 for each step do
4   Choose action  $a$  from  $s$  based on values of  $Q$  (e.g.,  $\epsilon$ -greedy)
5   Take action  $a$ , observe reward  $r$  and next state  $s'$ 
6    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a' \in A'(s)} Q(s', a') - Q(s, a) \right]$ 
7    $s \leftarrow s'$ 
8 end

```

Algorithm 10.1: The Q-learning algorithm.

$$Q(s, a) \rightarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right] \quad (10.7)$$

where α is the learning rate, and γ the discount-rate. A similar model-free temporal difference learning method is the SARSA learner. In this case, the update algorithm is given by:

$$Q(s, a) \rightarrow (1 - \alpha)Q(s, a) + \alpha [r + \gamma Q(s', a')] \quad (10.8)$$

where a' denotes the action taken at step s' (this is sometimes denoted by $\pi(s')$ to denote that policy π needs to be followed from state s'). The key difference between SARSA learning and Q-learning is that in SARSA learning, the Q-value update depends on the action selected at s' and hence the policy. Q-learning, on the other hand, is policy independent, since the Q-value update is based on the best possible action from state s' and, thus, does not depend on how the actions are chosen. Both algorithms are proven to converge to optimal policies, given that no abstractions have been applied, though in many practical problems with large state spaces, solving the full MDP is both time-intensive and impractical.

When an environment's model (i.e., transition function T and reward function R) is known or can be learned, that model can be used directly to evaluate the potential outcomes of an agent's actions. Such a "model-based" RL has two key advantages over model-free methods: First, given the existence of a model, the agent needs much less interaction with the environment, making it desirable in situations where the cost of interacting with the environment is high. Second, having a model provides a method for evaluating policies off-line – again minimizing the need for direct interaction with the environment. Adaptive real time dynamic programming (RTDP) is one of the earliest model-based reinforcement learning algorithms and has the simple flow shown in Algorithm 10.2 [8].

```

1 Initialize value function  $v(\cdot)$ 
2 Initialize  $s$  to a starting state
3 for each step do
4   Choose  $a$  from  $s$  using policy derived from  $v$  (e.g.,  $\epsilon$ -greedy)
5   Take action  $a$ , observe  $r, s'$ 
6   Update the model parameters  $r(s, a)$  and  $p(s'|s, a)$ 
7    $v(s) \leftarrow v(s) + \alpha \left[ \max_a \left\{ r(s, a) + \gamma \sum_{s'} p(s'|s, a) v(s') \right\} - v(s) \right]$ 
8    $s \leftarrow s'$ 
9 end

```

Algorithm 10.2: The adaptive RTDP algorithm.

3.4 Multiagent MDP Formulations

The presence of multiple agents creates different interpretation and extensions to the basic MDP formulation. Indeed, whether the agents are independent learners (each with its own MDP), are joint action learners (share the action space), or are coupled through their rewards (share rewards), leads to different MDPs. A non-exhaustive list of potential MDP formulations are listed here:

- Full state, joint action MDP with team reward $\langle S, A, T, \mathcal{R}, \Pi \rangle$: Find single policy π mapping system state $s \in S$ to system action $a \in A$ that maximizes team rewards $r \in \mathcal{R}$.
- Full state, independent action MDP with team reward $\langle S, A_i, T, \mathcal{R}, \Pi_i \rangle$: Agent i finds policy $\pi_i \in \Pi_i$ mapping system state s to agent's action $a_i \in A_i$ that maximizes team rewards r .
- Full state, independent action MDP with local reward $\langle S, A_i, T, \mathcal{R}, \Pi_i \rangle$: Agent i finds policy $\pi_i \in \Pi_i$ mapping system state s to agent's action $a_i \in A_i$ that maximizes agent's local reward $r_i \in R_i$.
- Local state, independent action MDP with team reward $\langle S_i, A_i, T, \mathcal{R}, \Pi_i \rangle$: Agent i finds policy π_i mapping agent's state $s_i \in S_i$ to agent's action a_i that maximizes team r .
- Local state, independent action MDP with local reward $\langle S_i, A_i, T, \mathcal{R}_i, \Pi_i \rangle$: Agent i finds policy π_i mapping agent's state s_i to agent's action a_i that maximizes agent's local reward $r_i \in R_i$.

Each of these formulations is appropriate for particular conditions, and has both benefits and shortcomings, depending on the dynamics of the domain in

which it is used. Below we present Markov games, an approach that aims to directly integrate the interactions among agents into the structure of an MDP.

3.5 Markov Games

Once multiple agents are interacting through their learning processes, the basic MDP model is no longer sufficient, and some situations require a different interpretation than those presented in Section 3.4. Markov or stochastic games generalize both repeated games and MDPs to the more general case of multiple states (repeated games are stateless) and multiple agents (basic MDPs consider only a single agent). In each stage, the game is in a specific state featuring a particular payoff function and an admissible action set for each player. Players take actions simultaneously and thereafter receive an immediate payoff depending on their joint action. A transition function maps the joint action space to a probability distribution over all states, which in turn determines the probabilistic state change. Thus, similar to a Markov decision process, actions influence the state transitions. A formal definition of Markov games is given below.

Definition 10.1 *The game $G = \langle n, S, A, q, \tau, \pi^1 \dots \pi^n \rangle$ is a stochastic game with n players and k states. In each state $s \in S = (s^1, \dots, s^k)$ each player i chooses an action a^i from its admissible action set $A^i(s)$ according to its strategy $\pi^i(s)$.*

The payoff function $\tau(s, a) : \prod_{i=1}^n A^i(s) \mapsto \mathbb{R}^n$ maps the joint action $a = (a^1, \dots, a^n)$ to an immediate payoff value for each player.

The transition function $T(s, a) : \prod_{i=1}^n A^i(s) \mapsto \Delta^{k-1}$ determines the probabilistic state change, where Δ^{k-1} is the $(k-1)$ -simplex and $T_{s'}(s, a)$ is the transition probability from state s to s' under joint action a .

In this chapter we restrict our consideration to the set of games where all states $s \in S$ are in the same ergodic set. The motivation for this restriction is twofold. In the presence of more than one ergodic set one could analyze the corresponding subgames separately. Furthermore, the restriction ensures that the game has no absorbing states. Games with absorbing states are of no particular interest in respect to evolution or learning since any type of exploration will eventually lead to absorption. The formal definition of an ergodic set in stochastic games is given below.

Definition 10.2 *In the context of a stochastic game G , $E \subseteq S$ is an ergodic set if and only if the following conditions hold:*

(a) *For all $s \in E$, if G is in state s at stage t , then at $t+1$:*

$\Pr(G \text{ in some state } s' \in E) = 1$, and

(b) *for all proper subsets $E' \subset E$, (a) does not hold.*

Note that in repeated games player i either tries to maximize the limit of the average of stage rewards

$$\max_{\pi_i} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \tau^i(t) \quad (10.9)$$

or the discounted sum of stage rewards $\sum_{t=1}^T \tau^i(t) \delta^{t-1}$ with $0 < \delta < 1$, where $\tau^i(t)$ is the immediate stage reward for player i at time step t .

3.6 State-of-the-Art Algorithms

In this section we discuss a number of state-of-the-art algorithms. As we cannot discuss all of them and be comprehensive in this chapter, we have chosen to describe three groups of them in more detail. Pointers will be provided to literature that elaborates on these more thoroughly and references to detailed overview articles.

3.6.1 Joint Action Learning

Joint action learning has been introduced in the context of cooperative repeated games, see [19]. A joint action learner (JAL) is an agent that learns Q-values for joint actions in a cooperative repeated game, in contrast to independent learners that learn Q-values only for individual actions. This entails that such an agent stores and adapts Q-values for joint actions \mathbf{a} with \mathbf{a} a vector $\langle a_1, \dots, a_n \rangle \in A_1 \times \dots \times A_n$ composed of the individual actions $a_i \in A_i$ of agent i . This implies that each agent can observe the actions of other agents.

Instead of carrying out Q-learning in the individual action space the JAL agent now learns in the joint action space. Since we consider stateless repeated games, the update rule of Q-learning can be simplified to:

$$Q(a) = Q(a) + \alpha(r - Q(a)) \quad (10.10)$$

In this stateless setting, we assume a Q-value, i.e., $Q(a)$, providing an estimate of the value of taking action a . At each time step a JAL agent i takes an action a_i belonging to joint action a . The sample $\langle a, r \rangle$ is the experience obtained by the agent: joint action a was performed resulting in reward r ; for instance when the agents involved in the game illustrated in Figure 10.2 play joint action $\langle a_0, b_0 \rangle$ they will receive reward r_1 . α is the typical learning rate to control step sizes of the learning process. It is important to realize that a JAL agent is now learning values for all joint actions and no longer individual actions. For instance in the 2-player 2-action game example in Figure 10.2, the joint action learner will learn Q-values for the tuples $\langle a_i, b_j \rangle$ with $i, j \in \{0, 1\}$ instead of for its individual actions a_i as an independent learner does.

$$\begin{array}{cc} & \begin{array}{cc} b_0 & b_1 \end{array} \\ \begin{array}{c} a_0 \\ a_1 \end{array} & \left(\begin{array}{cc} r_1 & r_2 \\ r_3 & r_4 \end{array} \right) \end{array}$$

Figure 10.2: General form of repeated matrix game.

Suppose that agent 1 (or the row player) has Q-values for all four joint actions; then the reward it can expect to accumulate will depend on the strategy adopted by the second (or column) player. Therefore a JAL agent will keep a model of the strategies of other agents i participating in the game such that it can compute the expected value of joint actions in order to select good subsequent actions balancing exploration and exploitation. A JAL then assumes that the other players i will choose actions in accordance with the model it keeps on the strategies of the other players. Such a model can be simply implemented through a fictitious play approach, in which one estimates the probability with which an agent will play a specific action based on the frequencies of their past plays. In such a way expected values can be computed for the actions of a JAL based on the joint actions. For instance in the above example we would have the following expected value EV for the first player's actions:

$$EV(a_i) = \sum_{b_j \in b_0, b_1} Q(b_j \cup \{a_i\}) Pr_{b_j}^1 \quad (10.11)$$

with $Pr_{b_j}^1$ the probability with which player 1 believes the other player will choose actions b_j implemented through a fictitious play approach. Using these EV values, player 1 can now implement, e.g., a Boltzmann exploration strategy for action selection.

3.6.2 Nash-Q Learning

Nash-Q, an algorithm introduced by Hu and Wellman [33, 34], aims to converge to a Nash equilibrium in general-sum stochastic games. In essence the algorithm extends the independent Q-learning algorithm to the multiagent case using the Markov game framework (see section 3.5). The optimal Q-values in this algorithm are the values that constitute a policy or strategy for the different agents that are in Nash equilibrium. The Nash equilibrium serves as the solution concept the agents aim to reach by learning iteratively. To achieve this each Nash-Q learning agent maintains a model of other agents' Q-values and uses that information to update its own Q-values.

The Nash-Q learning algorithm also considers joint actions (such as JAL) but now in the context of stochastic games (containing multiple states). In an n -

agent system, the Q-function for an agent becomes $Q(s, a_1, \dots, a_n)$, rather than the single-agent Q-function, $Q(s, a)$. Given these assumptions Hu and Wellman define a Nash Q-value as the expected sum of discounted rewards when all agents follow specified Nash equilibrium strategies from the next period on. The algorithm uses the Q-values of the next state to update those of the current state. More precisely, the algorithm makes updates with future Nash equilibrium payoffs, whereas single-agent Q-learning updates are based on the agent's own maximum payoff. To be able to learn these Nash equilibrium payoffs, the agent must observe not only its own reward, but those of others as well (as was the case in the JAL algorithm).

The algorithm proceeds as follows. The learning agent, indexed by i , learns its Q-values by starting with arbitrary values at time 0. An option is to let $Q_0^i(s, a_1, \dots, a_n) = 0$ for all $s \in S, a_1 \in A_1, \dots, a_n \in A_n$. At each time t , agent i observes the current state, and takes its action. After that, it observes its own reward, actions taken by all other agents, rewards of others, and the new state s' . Having this information it then computes a Nash equilibrium $\pi^1(s'), \dots, \pi^n(s')$ for the stage game $(Q_t^1(s'), \dots, Q_t^n(s'))$, and updates its Q-values according to:

$$Q_{t+1}^i(s, a_1, \dots, a_n) = (1 - \alpha_t)Q_t^i(s, a_1, \dots, a_n) + \alpha_t[r_t^i + \beta \text{Nash}Q_t^i(s')]$$

where $\text{Nash}Q_t^i(s') = \pi^1(s') \dots \pi^n(s') \cdot Q_t^i(s')$.

$\text{Nash}Q_t^i(s')$ is agent i 's payoff in state s' for the selected equilibrium. In order to calculate the Nash equilibrium $(\pi^1(s'), \dots, \pi^n(s'))$, agent i needs to know $Q_t^1(s'), \dots, Q_t^n(s')$. Since this information about other agents' Q-values is not available, this has to be learned as well. Since i can observe other agents' immediate rewards and previous actions, it can use that information to learn the other agents' Q-functions as well.

The algorithm is guaranteed to converge to Nash equilibrium, given certain technical conditions hold. Littman tackled these restrictive conditions of Nash-Q and introduced *Friend or Foe Q-learning* [44], which converges to Nash-equilibrium with fewer restrictions than Nash-Q. For more details on Nash-Q we refer to [34].

3.6.3 Gradient Ascent Algorithms

Infinitesimal gradient ascent (IGA) [65] is a policy gradient learning algorithm based on the limit of infinitesimal learning rates. It is shown that the average payoff of IGA converges to the pure Nash equilibrium payoff in two-agent, two-action matrix games, although policies may cycle in games with mixed equilibria. Each agent i participating in a game updates its policy π_i such that it follows the gradient of expected payoffs. The IGA algorithm has been generalized into the GIGA

(generalized infinitesimal gradient ascent) algorithm beyond two actions using the regret measure by Zinkevich [103]. Regret measures how much worse an algorithm performs compared to the best static strategy, with the goal of guaranteeing at least zero average regret (i.e., no-regret) in the limit. Since GIGA reduces to IGA in two-player, two-action games, it does not achieve convergence in all types of games. As a response to the fact that the IGA algorithm does not converge in all two-player-two-action games, IGA-WoLF (Win or Learn Fast) was introduced by Bowling [15] in order to improve the convergence properties of IGA. The policies of infinitesimal gradient ascent with WoLF learning rates are proven to converge to the Nash equilibrium policies in two-agent, two-action games [15]. The learning rate is made large if WoLF is losing. Otherwise, the learning rate is kept small as a good strategy has been found. In contrast to other reinforcement learning algorithms, IGA-WoLF assumes that the agents possess a lot of information about the payoff structure. In particular, sometimes agents are not able to compute the gradient of the reward function since that information is not available, which is necessary for this algorithm. Another well-known gradient-ascent type algorithm is policy hill climbing (PHC) explained in [15]. PHC is a simple adaptive strategy based on an agent's own actions and rewards, which performs hill climbing in the space of mixed policies. It maintains a Q-table of values for each of its base actions, and at every time step it adjusts its mixed strategy by a small step towards the greedy policy of its current Q-function. Also the PHC-WoLF algorithm needs prior information about the structure of the game. Related algorithms to infinitesimal gradient ascent have been devised to tackle this issue, such as for instance the WPL (weighted policy learner) algorithm of Abdallah and Lesser; see [1]. The GIGA-WoLF algorithm extended the GIGA algorithm with the WoLF principle [14] improving on its convergence properties. The algorithm basically keeps track of two policies of which one is used for action selection and the other is used for approximating the Nash equilibrium.

3.6.4 Other Approaches

Good overview articles on multiagent reinforcement learning algorithms can be found in [16, 64, 73].

The *extended replicator dynamics* algorithm provides another approach to reaching a Nash equilibrium [82]. Here the authors take a dynamical systems approach in which they first designed the stable differential equations, reaching an asymptotic stable Nash equilibrium in all types of stateless matrix games. After this they constructed the approximating learning algorithm showing the same behavior as the predefined dynamical system, i.e., reaching a stable Nash equilibrium.

Finally, the *AWESOME* algorithm, short for “Adapt When Everybody Is Sta-

tionary, Otherwise Move to Equilibrium,” provides a compromise between best response and precomputed play [20]. This algorithm converges to best response against stationary opponents, and otherwise converges to a precomputed Nash-equilibrium in self-play.

4 Evolutionary Game Theory as a Multiagent Learning Paradigm

Game theory is the field that is mainly concerned with interactive decision making in multiple actor situations. It provides a good basis to study properties of decision making and desirable outcomes of agent interactions. Furthermore, game theory is formally and empirically linked to learning. Basic game theoretic concepts are discussed in detail in Chapter 17. In this section, we only introduce the most relevant concepts from classical game theory and present the basics of evolutionary game theory (EGT). We then formally link evolutionary game theory and reinforcement learning using the replicator equations.

The simple concept at the root of evolutionary game theory is that instead of focusing on a finite, small number of players (such as the most commonly used 2-player, 2-action games), one can consider a population of infinite size. Then, by having that population in constant motion in terms of adapting its strategies, and by selecting the players in the population based on their payoffs, the “evolutionary” aspect is introduced into game theory.

4.1 Matrix Games

In Section 3.6.1 we illustrated already the general concept of a repeated matrix game (see Figure 10.2). Game theory models the interaction between players as a game, in which each player has a set of actions to choose from. All players have to select an action simultaneously, upon which they receive a payoff that depends on the combination of actions played. The goal for each player is to come up with a strategy that maximizes its payoff in the game. The payoffs can be conveniently represented in the bi-matrix (A, B) . A bi-matrix can, just like a normal matrix, contain any number of rows and columns; “bi” just reflects the fact that each cell contains two numbers: the payoffs for both players. Suppose the row player plays action i and the column player plays j , then the bi-matrix (A, B) gives the payoffs A_{ij} to the row player and B_{ij} to the column player. Figure 10.3 presents the payoff bi-matrices of three popular games, i.e., the Prisoner’s Dilemma, the Battle of the Sexes, and the Matching Pennies. Matching Pennies is an example of a competitive game, in which a win for one player is a loss for the other.

	<i>C</i>	<i>D</i>		<i>O</i>	<i>F</i>		<i>H</i>	<i>T</i>
<i>C</i>	$\left(\begin{array}{cc} 3,3 & 0,5 \end{array} \right)$		<i>O</i>	$\left(\begin{array}{cc} 1,\frac{1}{2} & 0,0 \end{array} \right)$		<i>H</i>	$\left(\begin{array}{cc} 1,-1 & -1,1 \end{array} \right)$	
<i>D</i>	$\left(\begin{array}{cc} 5,0 & 1,1 \end{array} \right)$		<i>F</i>	$\left(\begin{array}{cc} 0,0 & \frac{1}{2},1 \end{array} \right)$		<i>T</i>	$\left(\begin{array}{cc} -1,1 & 1,-1 \end{array} \right)$	
Prisoner's Dilemma			Battle of the Sexes			Matching Pennies		

Figure 10.3: Examples of matrix games.

4.2 Solution Concepts

In traditional game theory it is assumed that the players are rational, meaning that every player will choose the action that is best for itself, given its beliefs about the other players' actions. In this chapter, we discuss two solution concepts, the Nash equilibrium and Pareto optimality, which elucidate the behavior of players in games.

The first concept, Nash equilibrium, is defined as a set of strategies in a game where no player can increase its payoff by unilaterally changing its strategy (i.e., while the other players keep their strategies fixed).

Formally, a Nash equilibrium is defined as follows. When two players play the strategy profile $s = (s_i, s_j)$ belonging to the product set $S_1 \times S_2$, then s is a Nash equilibrium if

$$\begin{aligned} P_1(s_i, s_j) &\geq P_1(s_x, s_j) & \forall x \in \{1, \dots, n\} \\ P_2(s_i, s_j) &\geq P_2(s_i, s_y) & \forall y \in \{1, \dots, m\}. \end{aligned}$$

Three distinct classes of 2×2 normal form games are identified in [25]. The first class consists of games with one pure Nash equilibrium, such as the Prisoner's Dilemma (both players play *D* or *Defect*). The second class of games has two pure and one mixed Nash equilibrium. The Battle of the Sexes game belongs to this class (the pure ones are (O, O) and (F, F) , the calculation of the mixed one is left as an exercise). Finally, the third class of games has only one mixed Nash equilibrium; an example is the Matching Pennies game (both players play both actions with probability 0.5).

The second concept, Pareto optimality, is defined as a solution in a game where there is no joint action for all the players that can improve (or leave unchanged) the payoff of all the players. That is, there is no joint action that will not reduce the payoff of at least one player. More formally we have: a strategy combination $s = (s_1, \dots, s_n)$ for n agents in a game is Pareto optimal if there does not exist another strategy combination s' for which each player receives at least the same payoff P_i and at least one player j receives a strictly higher payoff than P_j . As

an example, the Nash equilibrium (D, D) in the Prisoner's Dilemma game is not Pareto optimal. Playing profile (C, C) is Pareto optimal, but not Nash.

4.3 Evolutionary Stable Strategies

The core equilibrium concept of evolutionary game theory is that of an *evolutionarily stable strategy* (ESS), introduced by Maynard Smith and Price in 1973 [46]. One way to understand this idea is to imagine a population of agents playing some game. After a number of games, the agents breed, producing new agents that play the same strategy, where the number of offspring of an agent depends on the payoff it has obtained. Now, in this setting, consider that initially all agents play the same strategy, but then a small number of agents switch to play a second strategy. If the payoff obtained by this new strategy is smaller than the payoff obtained by the original one, the second strategy will be played by fewer agents each generation, and will eventually disappear. In this case we say that the original strategy is evolutionarily stable against this new appearing strategy. More generally, we say a strategy is an ESS if it is robust against evolutionary pressure from any strategy that appears.

Formally an ESS is defined as follows. Suppose that a large population of agents is programmed to play the (mixed) strategy s , and suppose that this population is invaded by a small number of agents playing strategy s' . The population share of agents playing this mutant strategy is $\epsilon \in]0, 1[$. When an individual is playing the game against a random chosen agent, chances that it is playing against a mutant are ϵ and against a non-mutant are $1 - \epsilon$. The payoff for the first player, being a non-mutant is:

$$P(s, (1 - \epsilon)s + \epsilon s')$$

and the payoff for being a mutant is

$$P(s', (1 - \epsilon)s + \epsilon s')$$

Now we can state that a strategy s is an ESS if $\forall s' \neq s$ there exists some $\delta \in]0, 1[$ such that $\forall \epsilon : 0 < \epsilon < \delta$

$$P(s, (1 - \epsilon)s + \epsilon s') > P(s', (1 - \epsilon)s + \epsilon s')$$

holds. The condition $\forall \epsilon : 0 < \epsilon < \delta$ expresses that the share of mutants needs to be sufficiently small.

Note that one can frame the idea of ESS without the need to consider agents breeding. Instead one can consider agents being able to observe the average payoff of agents playing different strategies, and making a decision as to which strategy to adopt based on that payoff. In such a case, the “mutant” strategy is simply

a new strategy adopted by some agents – if other agents observe that they do well, they will switch to this strategy also, while if the new strategy does not do comparatively well, the agents that play it will switch back to the original strategy.

4.4 Replicator Dynamics

Agents playing a strategic game each hold a vector of proportions over possible actions, indicating the proportion of “individuals” or “replicators” in an infinite “population” (the agent) that have adopted a given action. At every time step, the proportions of actions for each agent are changed based on the rewards in the payoff tables as well as on the current probabilities of other agents’ choosing their actions.

An abstraction of an evolutionary process usually combines two basic elements: selection and mutation. Selection favors some population actions over others, while mutation provides variety in the population. The most basic form of replicator dynamics only highlights the role of selection, i.e., how the most fit actions in a population are selected.

In this chapter we consider continuous time replicator equations; for the discrete version we refer to [31]. The equations can be described as follows:

$$\frac{dx_i}{dt} = [(Ax)_i - x \cdot Ax] x_i \quad (10.12)$$

In Equation 10.12, x_i represents the density of action i in the population, and A is the payoff matrix that describes the different payoff values that each individual replicator receives when interacting with other replicators in the population. The state x of the population can be described as a probability vector $x = (x_1, x_2, \dots, x_n)$, which expresses the different densities of all the different types of replicators in the population. Hence $(Ax)_i$ is the payoff that replicator i receives in a population with state x and $x \cdot Ax$ describes the average payoff in the population. The growth rate $\frac{dx_i}{dt} / x_i$ of the proportion of action i in the population equals the difference between the action’s current payoff and the average payoff in the population. [25, 31, 97] detail further information on these issues.

Usually, we are interested in models of multiple agents that evolve and learn concurrently, and therefore we need to consider multiple populations. For simplicity, the discussion focuses on only two such learning agents. As a result, we need two systems of differential equations, one for each agent. This setup corresponds to a replicator dynamics for asymmetric games, where the available actions or strategies of the agents belong to two different populations.

This translates into the following coupled replicator equations for the two populations:

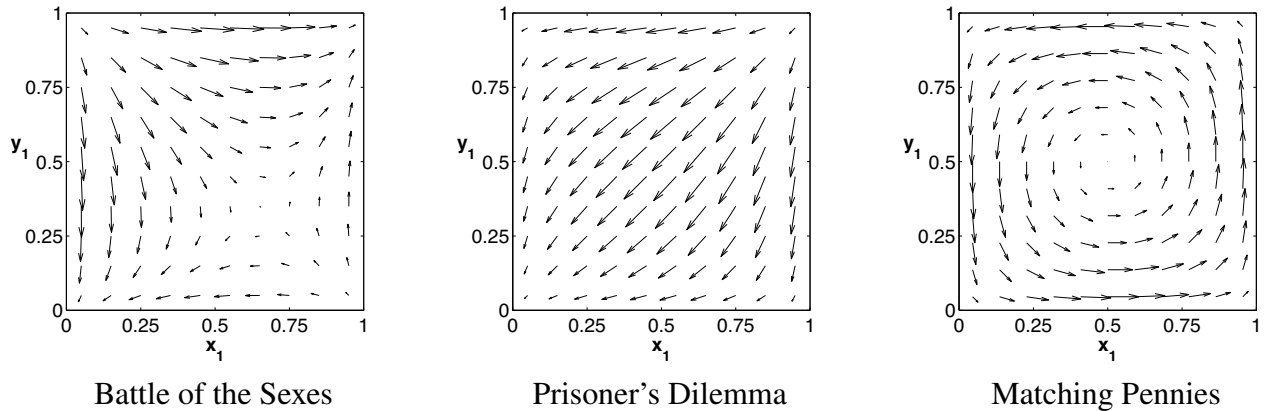


Figure 10.4: Visualization of the replicator dynamics of BoS, PD, and MP games.

$$\frac{dx_i}{dt} = x_i[(Ay)_i - x^T Ay] \quad (10.13)$$

$$\frac{dy_i}{dt} = y_i[(Bx)_i - y^T Bx] \quad (10.14)$$

where x (y) is the frequency distribution for player 1 (2), and A (B) represents its individual payoff matrix.

Equations 10.13 and 10.14 indicate that the growth rate of the types in each population is additionally determined by the composition of the other population, in contrast to the single population (learner) case described by Equation 10.12.

As an example we plot the direction fields of these equations for the three example games, i.e., Battle of the Sexes, Prisoner's Dilemma, and Matching Pennies, in Figure 10.4. It is a graphical representation of the solutions of the differential equations without solving the differential equations analytically. The plots visualize the dynamics qualitatively and show how all possible initial policies of the agents will evolve over time. Moreover they show the attractors and their basins. The x-axis shows the probability with which the first player plays its first action and the y-axis shows the probability with which the second player plays its first action. For instance in the PD game the x-axis shows the probability with which the first player plays *cooperate* – the y-axis shows the same probability for the second player.

4.5 The Role of Game Theory for Multiagent Learning

In the last decades, the focus of computer science has shifted from standalone systems towards distributed components that interact with one another. The more

these components exhibit properties that are usually assigned to intelligent entities, the more the theory of interactive decision making, i.e., game theory, becomes relevant to understand, model, and steer the interactions of these intelligent components. Agent technology, which has become an important research field within computer science, exactly studies computer systems that are capable of autonomous action taking in order to optimize some given performance measure. For an overview of the application of game theory to multiagent systems we refer to [53] and to Chapter 17. Here we lay out the use of evolutionary game theory for the purpose of evolution and learning in multiagent systems.

Building models of agents that evolve and behave optimally requires insight into the type and form of these agents' interactions with the environment and other agents in the system. In much work on multiagent adaptation and learning, this modeling is very similar to that used in a standard game theoretical model: players are assumed to have complete knowledge of the environment, are hyper-rational, and optimize their individual payoff disregarding what this means for the utility of the entire population. A more recent approach relaxes the strong assumptions of classical game theory and follows the evolutionary approach. The basic properties of multiagent systems seem to correspond well with those of evolutionary game theory. First of all, a multiagent system is a distributed dynamic system, which makes it hard to model using a rather static theory like traditional game theory. Secondly, a multiagent system consists of actions and interactions between two or more independent agents, who each try to accomplish a certain, possibly cooperative or conflicting, goal. No agent has the guarantee to be completely informed about the other agents' intentions or goals, nor has it the guarantee to be completely informed about the complete state of the environment. Furthermore, evolutionary game theory offers us a solid basis to understand dynamic iterative situations in the context of strategic interactions and this fits well with the dynamic nature of a typical multiagent system. Not only do the fundamental assumptions of evolutionary game theory and multiagent systems seem to fit each other rather well, but there is also a formal relationship between the replicator equations of evolutionary game theory and reinforcement learning. This relation will allow for studying the theoretical properties of multiagent learning in greater detail.

4.6 Evolutionary Game Theory as a Theoretical Framework

Learning in multiagent systems is a complex and cumbersome task. The theoretical foundation of single-agent learning implies that as long as the environment an agent experiences is stationary, and the agent can experiment sufficiently, RL guarantees convergence to the optimal strategy [72]. This is no longer valid in the multiagent case because there are now multiple agents learning in the same

environment, facing unobservable actions and rewards of other agents and non-stationarity of the environment. All these complicating properties of multiagent systems make it hard to engineer learning algorithms capable of finding optimal solutions.

Recent debate in the MAL community gave direction to a new research agenda for the field [64]. An important problem of MAL that stands out is the lack of a theoretical framework such as exists for the single-agent case. For this purpose we employ an evolutionary game theoretic approach by formally linking and analyzing the relation between RL and replicator dynamics (RD). More precisely, in [13, 52, 84, 86] the authors derived a formal link between the replicator equations of evolutionary game theory and such reinforcement learning techniques as Q-learning and learning automata. In particular this link showed that in the limit these learning algorithms converge to a certain form of the RD. This makes it possible to establish equilibria using the RD that tell us what states a given learning system will settle into over time and what intermediate states it will go through.

There are a number of benefits to exploiting this link: one, the model predicts desired parameters to achieving Nash equilibriums with high utility; two, the intuitions behind a specific learning algorithm can be theoretically analyzed and supported by using the basins of attraction; three, it was shown how the framework can easily be adapted and used to analyze new MAL algorithms, such as, for instance, lenient Q-learning, regret minimization, etc. [38, 52].

Extension of the framework to multiple states (e.g., switching dynamics) and continuous strategy spaces have been explored as well and can be found in, e.g., [30, 88, 93]. In this chapter we limit ourselves to describing the link between stateless Q-learning (and its two variants *FAQ* and *LFAQ*) and the basic replicator equations. More precisely we present the dynamical system of Q-learning. These equations are derived by constructing a continuous time limit of the Q-learning model, where Q-values are interpreted as Boltzmann probabilities for the action selection. For reasons of simplicity we consider games between two players. The equations for the first player are

$$\frac{dx_i}{dt} = x_i \alpha \tau((A\mathbf{y})_i - \mathbf{x} \cdot A\mathbf{y}) + x_i \alpha \sum_j x_j \ln\left(\frac{x_j}{x_i}\right) \quad (10.15)$$

and analogously for the second player, we have

$$\frac{dy_i}{dt} = y_i \alpha \tau((B\mathbf{x})_i - \mathbf{y} \cdot B\mathbf{x}) + y_i \alpha \sum_j y_j \ln\left(\frac{y_j}{y_i}\right) \quad (10.16)$$

Equations 10.15 and 10.16 express the dynamics of both Q-learners in terms of Boltzmann probabilities. Each agent (or player) has a probability vector over

its action set, more precisely x_1, \dots, x_n over action set a_1, \dots, a_n for the first player and y_1, \dots, y_m over b_1, \dots, b_m for the second player.

For a complete mathematical derivation and discussion of these equations we refer to [84, 85]. Comparing (10.15) or (10.16) with the RD in (10.12), we see that the first term of (10.15) or (10.16) is exactly the RD and thus takes care of the selection mechanism (see [97]). The mutation mechanism for Q-learning is therefore left in the second term, and can be rewritten as:

$$x_i \alpha \sum_j x_j \ln(x_j) - \ln(x_i) \quad (10.17)$$

In equation (10.17) we recognize two entropy terms, one over the entire probability distribution x , and one over strategy x_i .

Other RL methods for which the dynamics have been derived are the following:

Frequency Adjusted Q-learning (FAQ) [37] is a variation of the value-based Q-learning method, which modulates the learning step size to be inversely proportional to the action selection probability. This modulation leads to more rational behavior that is less susceptible to initial overestimation of the action values. The update rule for FAQ-learning is $Q_i(t+1) \leftarrow Q_i(t) + \min\left(\frac{\beta}{x_i}, 1\right) \alpha [r(t+1) + \gamma \max_j Q_j(t) - Q_i(t)]$, where α and β are learning step size parameters, and γ is the discount factor. The Boltzmann action-selection mechanism is used with a temperature τ : $x_i = \frac{e^{Q_i \cdot \tau^{-1}}}{\sum_j e^{Q_j \cdot \tau^{-1}}}$.

Lenient FAQ-learning (LFAQ) is a variation of FAQ-learning. Leniency has been shown to improve convergence to the optimal solution in coordination games [52]. Leniency is introduced by having the FAQ method collect κ rewards for an action, before updating this action's Q-value based on the highest perceived reward.

Finite Action-set Learning Automata (FALA) [49] is a policy-based learning method. [49] considers the linear reward-inaction update scheme. It updates its action selection probability based on a fraction α of the reward received. The probability is increased for the selected action, and decreased for all other actions. The update rules for FALA are $x_i(t+1) \leftarrow x_i(t) + \alpha r_i(t+1)(1 - x_i(t))$ if i is the action taken at time t , and $x_j(t+1) \leftarrow x_j(t) - \alpha r_i(t+1)x_j(t)$ for all actions $j \neq i$.

Regret Minimization (RM) [38] is another policy-based learning method. It updates its policy based on the loss (regret) incurred for playing that policy, with respect to some other policy. [38] studies the polynomial weights method, which calculates the loss with respect to the optimal policy in hindsight. Again, a learning step size parameter α controls the update process. The method updates the weight of the actions by $w_i(t+1) \leftarrow w_i(t)(1 - \alpha l_i(t+1))$. Normalization of these weights gives the action selection probabilities.

Method	Evolutionary model
FAQ	$\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau} [(Ay)_i - x^T Ay] + x_i \alpha \sum_j x_j \ln\left(\frac{x_j}{x_i}\right)$
LFAQ	$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k: A_{ik} \leq A_{ij}} y_k \right)^\kappa - \left(\sum_{k: A_{ik} < A_{ij}} y_k \right)^\kappa \right]}{\sum_{k: A_{ik} = A_{ij}} y_k}$ $\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau} (u_i - x^T u) + x_i \alpha \sum_j x_j \ln\left(\frac{x_j}{x_i}\right)$
FALA	$\frac{dx_i}{dt} = \alpha x_i [(Ay)_i - x^T Ay]$
RM	$\frac{dx_i}{dt} = \frac{\lambda x_i [(Ay)_i - x^T Ay]}{1 - \lambda [\max_k (Ay)_k - x^T Ay]}$

Figure 10.5: Overview of the evolutionary dynamics of the studied learning methods. Only the dynamics of the first player are given; the dynamics of the second player can be found by substituting B for A , swapping x and y , and swapping the matrix indexes in the u_i rule of LFAQ.

The evolutionary dynamics of these models are presented in Figure 10.5. As an example of these evolutionary models we visualize the dynamics of FAQ and LFAQ in directional field plots, see Figure 10.6. Self-play is the standard form of learning experiments, in which each competing player implements the same learning method. We describe the behavior of the learning methods in self-play. All plots use the same parameter settings: step size $\alpha = 0.001$; for (L)FAQ, $\beta = 0.01$, $\tau = 0.01$, and $\gamma = 0$. The behavior of a learner over time can be visualized using a trajectory plot or by plotting the directional field of the corresponding replicator dynamics. Here, a combination of both is used to show how the individual learning trajectories relate to their evolutionary prediction. All trajectory plots show the average trajectory over 10 runs of 50,000 iterations each (100,000 for the Prisoner's Dilemma). The plots show that FAQ and LFAQ indeed behave as predicted by their evolutionary models. In the Prisoner's Dilemma all trajectories converge to the game's Nash equilibrium (D,D), which in the plot lies at (0,0). The directional field shows that indeed all possible initial policies will eventually converge to this equilibrium.

5 Swarm Intelligence as a Multiagent Learning Paradigm

Swarm intelligence is a bio-inspired machine learning technique, largely based on the behavior of social insects (e.g., ants and honeybees), which is concerned

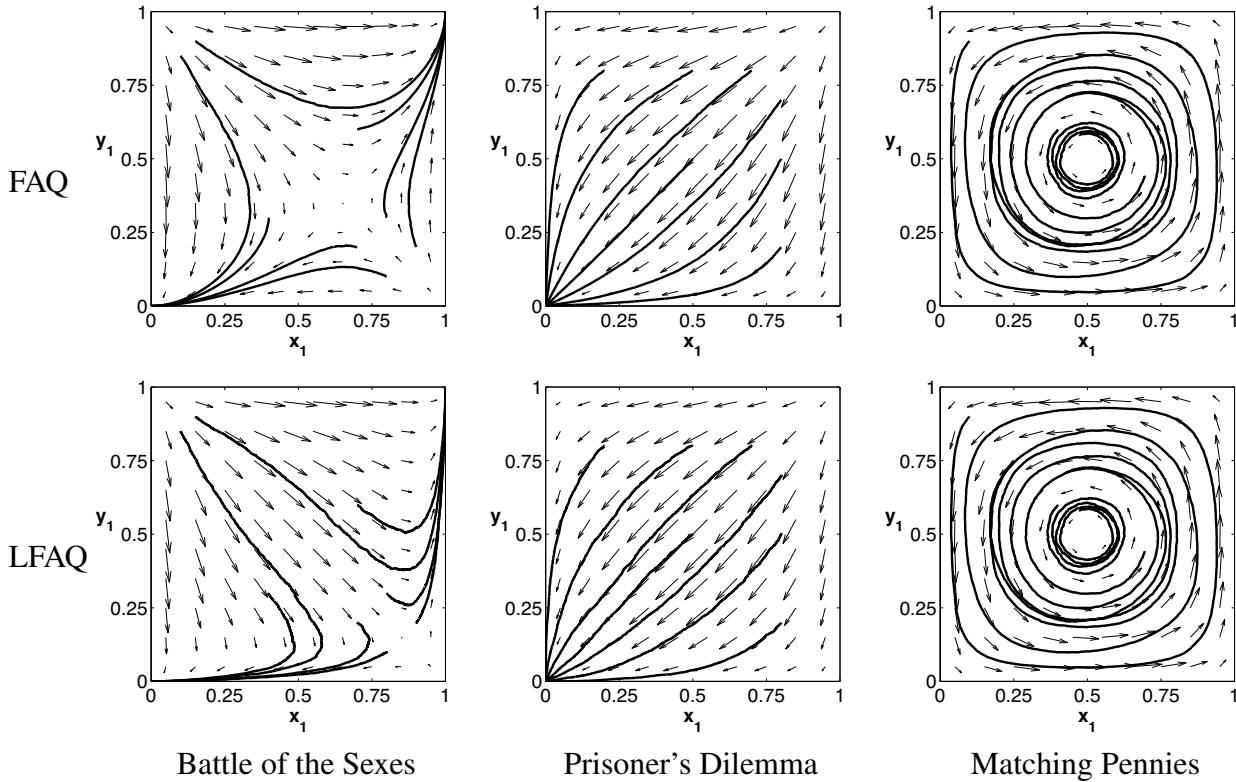


Figure 10.6: Policy trajectories of FAQ and LFAQ in three different games.

with developing self-organized and decentralized adaptive algorithms. The type and form of learning in a swarm intelligence is characterized by a large population of cognition limited agents that locally interact. Rather than developing complex behaviors for single individuals, as is done in reinforcement learning, swarm intelligence investigates the emerging (intelligent) behavior of a group of simple individuals that achieve complex behavior through their interactions with one another. Consequently, swarm intelligence can be considered as a cooperative multiagent learning approach in that the behavior of the full set of agents is determined by the actions of and interactions among the individuals.

In a swarm intelligence, each individual in the group follows simple rules without central control structures. By interacting locally, a global behavior emerges, yet the individual has no knowledge of this “big picture” behavior. Examples of such systems are ant foraging, bird flocking, fish schooling, and animal herding. Though based on different principles, swarm intelligence and reinforcement learning are closely related, as both techniques comprise iterative learning algorithms based on trial and error, and use a “reinforcement signal” (reward or

fitness) to find optimal solutions. The key difference though is how the reinforcement signal is used to modify an individual's behavior. In this chapter, we do not delve into the details of this relationship.

Currently the most well-known swarm intelligence algorithms are pheromone-based (stigmergic), such as ant colony optimization. For an overview, we refer to [12, 23]. Recently, interest has grown in non-pheromone-based approaches, mainly inspired by the foraging behavior of honeybees. For an overview we refer to Lemmens et al. [41, 42]. Below we concisely explain the basics of both approaches.

5.1 Ant Colony Optimization

Ant colony optimization (ACO) is a class name for ant-inspired algorithms solving combinatorial optimization problems. Algorithms belonging to it are stochastic search procedures in which the central component is the pheromone model. Pheromone-based algorithms are inspired by the behavior of ants and are the most well-known swarm intelligence algorithm. The algorithms are based on the fact that ants deposit a pheromone trail on the path they take during travel. Using this trail, they are able to navigate toward their nest or food. Ants employ an indirect recruitment strategy by accumulating pheromone trails in the environment. The ants literally communicate indirectly via the environment, in the subject called *stigmergy*. When a trail is strong enough, other ants are attracted to it and, with high probability, will follow this trail toward a destination. More precisely, the more ants follow a trail, the more that trail becomes attractive for being followed. This is known as an autocatalytic process. Since long paths take more time to traverse, it will require more ants to sustain a long path. As a consequence, short paths will eventually prevail. Figure 10.7 illustrates this concept.

Optimization problems best suited to being solved by ant colony optimization are those that can be cast as computational problems on a graph, implying that optimal solutions will correspond to specific paths in such a graph. Successful examples of such problems include the traveling salesman problem [23], various routing problems [17], group shop scheduling [11], and “covering problems” with robots [94].

The basic ant system algorithm works as follows [12]. A swarm of m ants solve the optimization problem iteratively by traversing the graph, which represents the problem at hand. An example of such a graph is the traveling salesman problem in which the purpose is to find the shortest route through a number of cities, visiting each of them exactly once. Each of these m ants searches for candidate solutions individually. While doing this, they influence each other's solution indirectly through pheromones deposited in the environment in which they search.

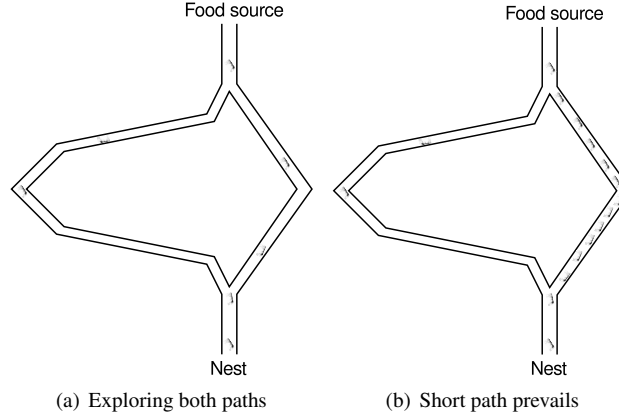


Figure 10.7: Ant foraging. Individual ants explore both paths. Then the shorter path on the right gets reinforced more and becomes the dominant path.

Nodes of such a graph are called states, and in one iteration t an ant produces a solution to the given problem (e.g., a route between cities in the traveling salesman problem). At the start all edges are initialized with a certain amount of pheromone τ_0 . The probability for an ant k to move from state s_i to state s_j at iteration t is given by

$$p_{i,j}^k = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta} \quad (10.18)$$

with $\tau_{ij}(t)$ the amount of pheromone along the edge from state s_i to state s_j at iteration t . The parameter α controls the weight given to the pheromone part of the equation. η_{ij} expresses the desirability of moving to state s_j ; for instance in the traveling salesman problem it would correspond to the visibility of the next city, formally described by the inverse of the distance between cities (states) s_i and s_j , i.e., $\frac{1}{d(i,j)}$. β controls the weight given to the visibility part of the equation. N_i^k is the set of unvisited nodes of ant k that are reachable from state s_i . The intuition behind the rule is that ants prefer to move to nodes or states connected to the current state by short edges with high pheromone presence.

Once all ants have found their own solution or route, pheromones are updated on all edges according to the following equation:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k(t) \quad (10.19)$$

where ρ controls evaporation of pheromone and $\Delta\tau_{ij}^k(t)$ is the amount per unit of length of trail pheromone laid on edge (i, j) by the k^{th} ant, i.e.,

$$\Delta\tau_{ij}^k(t) = \frac{1}{L_k(t)} \quad (10.20)$$

where edge (i, j) belongs to the solution generated by ant k . $L_k(t)$ is the length of the solution found by ant k at iteration t . As with the biological counterpart, shorter solutions or routes are preferred.

5.2 Bee Colony Optimization

Recently, non-pheromone-based algorithms have also been proposed, see, e.g., Lemmens et al. and Teodorovic et al. [41, 75]. Such algorithms are inspired by the foraging and nest-site selection behavior of bees. In contrast to ants, bees do not use pheromones to navigate through unfamiliar worlds. Instead, for navigation, they use a strategy named path integration (PI). Bees are able to compute their present location from their past trajectory continuously and, as a consequence, can return to their starting point by choosing the direct route rather than retracing their outbound trajectory. For recruitment, bees communicate directly by dancing in the nest. The dance is a representation of a PI vector and therefore communicates distance and direction toward a destination (see Figure 10.9). Dance strength (i.e., its duration) indicates the “fitness” of a solution. More precisely, bees know which destination they are traveling to. Depending upon the strength of a dance, the dance attracts other bees which may follow the PI vector toward a destination. High evaluated solutions result in stronger bee dances. The more bees follow a danced PI vector, the more that PI vector will be danced for and the more it attracts other bees. Eventually, the best solution prevails. Although we mainly refer to bees when we speak about PI, we have to note that PI can also be found in other insects such as desert ants. For an illustration see Figure 10.8. A full description of bee-inspired algorithms is beyond the scope of this chapter. We refer the interested reader to [41, 42]. Note that non-pheromone-based algorithms can be applied to the same problems as pheromone-based algorithms.

For a detailed comparison between ACO and bee colony optimization (BCO) we refer to [42]. A key difference though between ACO and BCO is that ACO largely depends on indirect communication between individual agents through the environment, whereas BCO depends on direct communication between individuals in the hive or nest. Generally speaking, social-insect swarm intelligence de-

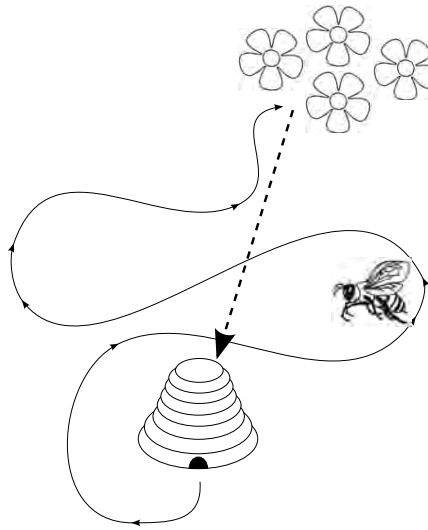


Figure 10.8: Path integration. By tracking all angles steered and all distances covered (i.e., solid arrow), bees have an up-to-date vector indicating their nest at all times (i.e., dashed arrow).

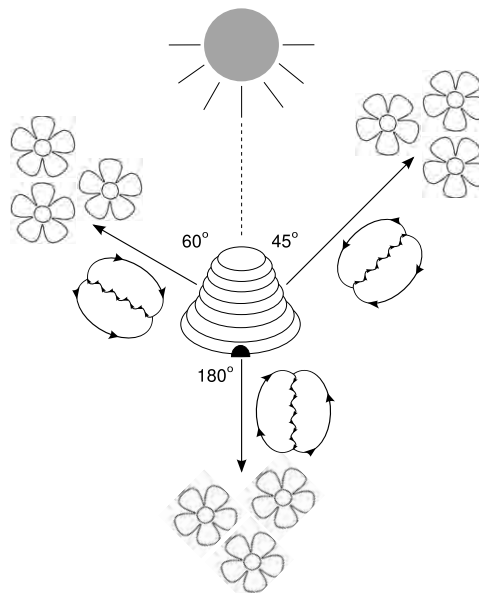


Figure 10.9: Distance and direction by waggle dance. Waggle straight up on the vertical comb indicates a food source that is located at an azimuthal angle of 0 degrees while waggle straight down indicates a food source located at an azimuthal angle of 180 degrees.

depends on four principles. First, individuals in a swarm only have local awareness. Their view of the environment is limited to their direct surroundings and they do not take their previous actions into account for future actions. Second, swarms are decentralized. Although central control is possible over short distances with a small number of individuals, the performance of such control deteriorates with distance or increasing number of agents. Therefore, decentralized control is a necessity. Third, to implement control, social insects rely on local interaction. Local interaction is used to solve a subproblem of the task at hand. For example, find the shortest path in a segment in the total search space. Fourth, by using local interaction the swarm displays self-organization on a global level.

6 Neuro-Evolution as a Multiagent Learning Paradigm

In addition to reinforcement learning and swarm intelligence, a third type of learning approach for autonomous agents consists of using evolutionary algorithms to train a policy to perform the state-to-action mapping. In this approach, rather than update the parameters of a single agent interacting with the environment as is done in reinforcement learning, one searches through a population of policies to find one that is appropriate for the task. This type of “policy search” approach is best suited to domains with continuous states and actions where traditional reinforcement learning approaches generally encounter difficulties.

The most commonly used policy in conjunction with evolutionary algorithms is a feed-forward neural network with non-linear activation functions [29]. The aim of the neural network is to perform a mapping between its inputs (states) and its outputs (actions) that satisfies the agent’s task. For example, a mobile robot using a neural network to navigate can map the sensory inputs it receives to direction and velocity. The key then is to find the correct parameters for the neural network that will provide the desired behavior. If the desired behavior is known in advance, traditional training algorithms (e.g., gradient descent) can be used. For example, an autonomous vehicle can learn to drive by recording the actions of a human driver, using those actions as the “correct” actions [55, 59] and modifying its parameters to minimize the error between its actions and those of the driver. In most cases, however, the correct actions for each particular state are not known in advance. For such cases, the use of evolutionary algorithms provides an effective search method through the possible policies, resulting in the set of approaches known as neuro-evolutionary algorithms [18, 24, 26, 40, 48, 69].

6.1 Evolutionary Algorithm Basics

Evolutionary computation has a long history of success in single-agent control problems, and has been extensively applied to a multitude of domains [27, 32, 61, 63, 99]. At its most fundamental, an evolutionary algorithm consists of five steps: representation, selection, generation of new individuals (crossover and mutation), evaluation, and replacement.

- *Representation* refers to how the problem is abstracted into a set of parameters that capture the key aspects of the policy. For example, a neural network, as described above, is a representation of a policy.
- *Selection* refers to how an individual policy is selected from a population of policies. For example, the best policy can be selected, or policies can be selected with a probability proportional to their success.
- *Generation of new individuals* is the core of the search algorithm. It may consist of simply mutating the selected individual by injecting some randomness into the policy, or by a combination of crossover and mutation. The impact of the crossover operation where two individual policies are merged to yield a new policy is greatly dependent on the representation. For most representations, crossover simply introduces random noise and can be skipped. In a few cases, where the representation captures partial solution concepts, crossover may be beneficial.
- *Evaluation* refers to the assignment of a success/failure metric to the new policy. It is sometimes referred to as the “fitness” function for an individual.
- *Replacement* refers to the final step where the new individual is reinserted into the population, usually at the expense of removing another individual. For example, the worst individual can be removed to make room for the new individual, or an individual may be removed based on a probability that depends on its fitness.

Algorithm 10.3 shows a simple evolutionary algorithm, with specific choices for these steps. Though all five of these components affect the performance of an evolutionary algorithm, two are particularly critical in the complex domains of interest to autonomous agents. Indeed, the selection, generation of new individuals, and replacement can follow simple principles and still provide good performance.

Though all steps are important, the two steps that are particularly critical are representation and evaluation. The representation step is a key step in that it allows the abstraction that guides the search for good policies. If the policy is represented

```

1 Initialization Phase:
2 At  $t = 0$ 
3 for Each policy  $\pi_k$  where  $k \in (1, N_{pop})$  do
4   | Use policy  $\pi_k$  for a fixed number of steps
5   | Evaluate performance of  $\pi_k$ 
6 end
7 Training Phase:
8 for  $t < t_{max}$  do
9   | Select a policy  $\pi_i$  from population of policies:
10  |   with probability  $\epsilon$ :  $\pi_{current} \leftarrow \pi_i$ 
11  |   with probability  $1 - \epsilon$ :  $\pi_{current} \leftarrow \pi_{best}$ 
12  | Modify the parameters of  $\pi_{current}$  to produce  $\pi'$  (mutation)
13  | Use policy  $\pi'$  for a fixed number of steps
14  | Evaluate performance of  $\pi'$ 
15  | Replace  $\pi_{worst}$  with  $\pi'$ 
16  |  $t \leftarrow t+1$ 
17 end

```

Algorithm 10.3: An evolutionary search algorithm to determine the parameters of a policy. This general algorithm uses a population of N_{pop} , selects the best policy with probability $1 - \epsilon$ at each step, generates a new solution by perturbing the parameters of the selected policy, and deterministically replaces the worst policy in the population at the end of each step.

in a manner that captures its salient features, the search will be far more effective. In addition, it is important to select a representation where similar policies will be “close” in parameter space, and where small changes to the parameters will not lead to significantly different policies. Indeed, the effectiveness of the new individual generation step is directly related to the effectiveness of the representation.

The evaluation step, on the other hand, applies the selective pressure to guide the search. Having the appropriate evaluation function is critical to finding good solutions, and in ensuring that policies that offer a new and useful advancement are identified and kept in the population (this is another instance of the fundamental credit assignment problem discussed in Section 2.2). The impact of the evaluation function is even more critical in large coevolutionary systems where agents operate in collaboration or competition with one another. Indeed, the co-evolution step brings the same concerns of a search in a non-stationary domain. Coevolution has been successfully applied in many domains, including multi-rover coordination where robots are not only evolved to learn good policies, but also to cooperate with one another [39, 50, 51].

6.2 Linking Multiagent Reinforcement Learning to the Neuro-Evolutionary Approach

The coevolutionary approach has also been linked to the replicator equations of section 4, and as such also fits into the evolutionary game theory framework (see [100]). In [52] the replicator dynamics models for cooperative coevolutionary algorithms and for traditional multiagent Q-learning have been compared and extended to account for lenient learners: agents that forgive possible mismatched teammate actions that resulted in low rewards. These extended formal models have been used to study the convergence guarantees for these algorithms, and also to visualize the basins of attraction to optimal and suboptimal solutions in benchmark coordination problems.

7 Case Study: Air Traffic Control

Air traffic control provides an ideal case study for demonstrating both the need for and the effectiveness of multiagent learning approaches. Air traffic is a naturally distributed problem where the complex interaction among the aircraft, airports, and traffic controllers renders a predetermined centralized solution severely sub-optimal at the first deviation from the expected plan. Though a truly distributed and adaptive solution (e.g., free flight where aircraft can choose almost any path) offers the most potential in terms of optimizing flow [54, 66, 67], it also provides the most radical departure from the current system. In this section, we provide an agent-based system that can be implemented readily, one based on assigning an agent to a “fix,” which is a specific location in 2D [2, 6, 77]. This representation allows localized fixes (or agents) to have direct impact on the flow of air traffic, and makes a good case study for showing the impact of multiagent learning in general and the structural credit assignment problem described in Section 2.2.

7.1 Motivation

The efficient, safe, and reliable management of the ever increasing air traffic is one of the fundamental challenges facing the aerospace industry today. On a typical day, more than 40,000 commercial flights operate within the US airspace [68], and the scheduling allows for very little room to accommodate deviations from the expected behavior of the system. As a consequence, the system is slow to respond to developing weather or airport conditions, leading potentially minor local delays to cascade into large regional congestions. Current air traffic management

relies on a centralized, hierarchical routing strategy that performs flow projections ranging from one to six hours. Therefore, the system is not only slow to respond to changes, but is also at the limit of its capacity. Unlike many other flow problems where the increasing traffic is to some extent absorbed by improved hardware (e.g., more servers with larger memories and faster CPUs for Internet routing), the air traffic domain needs to find mainly algorithmic solutions, as the infrastructure (e.g., number of airports) will not change significantly to impact the flow problem. There is therefore a strong need to explore multiagent solutions based on learning agents to the air flow control problem.

With over 40,000 flights operating within the United States airspace on an average day, the management of traffic flow is a complex and demanding problem. Not only are there concerns for the efficiency of the system, but also for fairness (e.g., different airlines), adaptability (e.g., developing weather patterns), and reliability and safety (e.g., airport management). In order to address such issues, the management of this traffic flow occurs over four hierarchical levels:

1. Separation assurance (2–30 minute decisions);
2. Regional flow (20 minutes to 2 hours);
3. National flow (1–8 hours); and
4. Dynamic airspace configuration (6 hours to 1 year).

Of these, the regional and national flow are ideally suited for algorithmic improvements as they fit between long-term planning by the FAA and the very short-term decisions by air traffic controllers.

7.2 Simulation and System Performance

Simulating air traffic to allow the evaluation of agent-based decision making is a key step. Two different types of simulators can be used for this, depending on the aims of the research. Physics-based simulators (such as FACET [10, 68]) provide an approach based on propagating the trajectories of proposed flights forward in time. FACET simulates air traffic based on flight plans and through a graphical user interface allows the user to analyze congestion patterns of different sectors and centers (Figure 10.10). FACET also allows the user to change the flow patterns of the aircraft through a number of mechanisms, including metering aircraft through fixes. The user can then observe the effects of these changes to congestion. Though accurate and well-accepted in industry, physics-based simulators are also slow, making them difficult for learning-based approaches where a (very) large number of runs may be required.

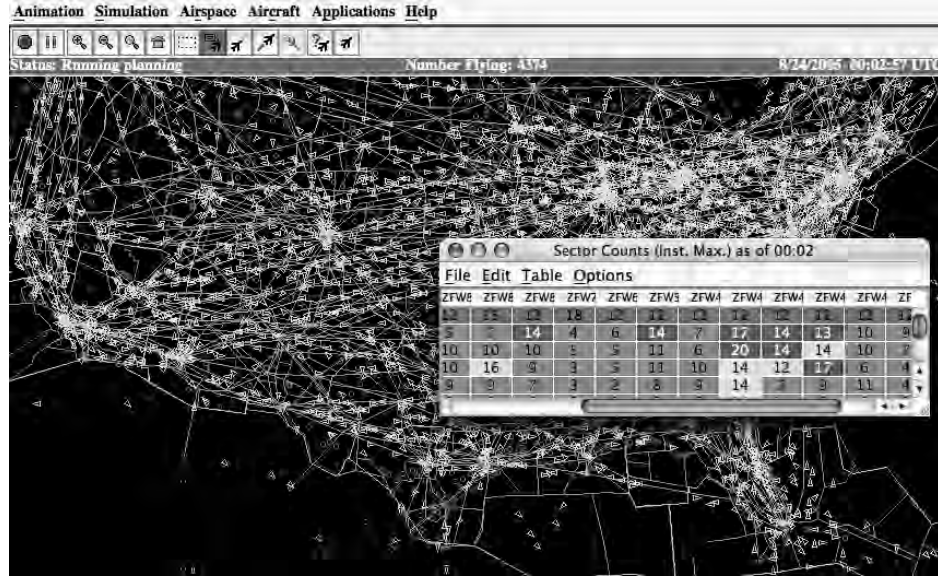


Figure 10.10: Graphical user interface of the FACET simulator.

Event-based simulators (such as FEATS) provide an alternative that allows one to quickly simulate thousands of aircraft of different characteristics taking off from airports, navigating via waypoints and airways to their destination airport, and landing. Such simulators are designed for being used with learning algorithms and can simulate 26,000 flights/second (on a high-end PC). Individual simulations take a fraction of a second to complete, which allows users to efficiently experiment with machine learning techniques [57].

A good system performance evaluation function focuses on delay and congestion, and also potentially the “fairness” on different commercial entities. Both defining and optimizing fairness requires a great deal of domain specific information. As a consequence, in this chapter we present results based on delay and congestion alone to demonstrate the impact of multiagent learning. The linear combination of these two terms gives the full system evaluation function, $G(z)$, as a function of the full system state z . More precisely, we have

$$G(z) = -(B(z) + \alpha C(z)) , \quad (10.21)$$

where $B(z)$ is the total delay penalty for all aircraft in the system, and $C(z)$ is the total congestion penalty. The relative importance of these two penalties is determined by the value of α , a congestion cost.

The total delay, B , is a sum of delays over the set of aircraft A and is given by:

$$B(z) = \sum_{a \in A} B_a(z) \quad (10.22)$$

where $B_a(z)$ is the delay of each aircraft caused by the agents' controls. For controls that delay aircraft (discussed in Section 7.3), the $B_a(z)$ is simply the amount of delay applied to that aircraft. For controls involving rerouting (Section 7.3), the delay is the amount of additional time it takes an aircraft to go on its new route instead of its scheduled route.

The total congestion penalty is a sum over the congestion penalties over the sectors of observation, S :

$$C(z) = \sum_{s \in S} C_s(z) \quad (10.23)$$

where

$$C_s(z) = \sum_t \Theta(k_{s,t} - c_s)(k_{s,t} - c_s)^2, \quad (10.24)$$

where c_s is the capacity of sector s as defined by the FAA, and $\Theta(\cdot)$ is the step function that equals 1 when its argument is greater or equal to zero, and has a value of zero otherwise. Intuitively, $C_s(z)$ penalizes a system state where the number of aircraft in a sector exceeds the FAA's official sector capacity. Each sector capacity is computed using various metrics, which include the number of air traffic controllers available. The quadratic penalty is intended to provide strong feedback to return the number of aircraft in a sector to below the FAA-mandated capacities.

7.3 Agent-Based Air Traffic

Four key decisions are needed to implement a full agent-based system. First, the agents need to be defined in a manner that allows agents to have access to relevant information and be in a position to influence the system dynamics. Second, the agent action set has to be defined in a manner that allows agents to have impact on the system performance defined above. Third, the agent's learning algorithm needs to be selected. Finally, the agent reward structure has to be defined in a manner that allows agents to determine what a "good" action is given the system condition.

Selecting the aircraft as agents is perhaps the most obvious choice for defining an agent. That selection has the advantage that agent actions can be intuitive (e.g., change of flight plan, increase or decrease in speed and altitude) and offer a high level of granularity, in that each agent can have its own policy. However, there are several problems with that approach. First, there are in excess of 40,000 aircrafts in a given day, leading to a massively large multiagent system. Second, as the agents would not be able to sample their state space sufficiently, learning would be prohibitively slow.

In this chapter, we present results based on assigning agents to individual "fix" ground locations throughout the airspace. Each agent is then responsible for any aircraft going through its fix. Fixes offer many advantages as agents:

1. Their number can vary depending on need. The system can have as many agents as required for a given situation (e.g., agents coming “live” around an area with developing weather conditions).
2. Because fixes are stationary, collecting data and matching behavior to reward is easier.
3. Because aircraft flight plans consist of fixes, agents will have the ability to affect traffic flow patterns.
4. They can be deployed within the current air traffic routing procedures, and can be used as tools to help air traffic controllers rather than compete with or replace them.

The second issue that needs to be addressed is determining the action set of the agents. Again, an obvious choice may be for fixes to “bid” on aircraft, affecting their flight plans. Though appealing from a free flight perspective, that approach makes the flight plans too unreliable and significantly complicates the scheduling problem (e.g., arrival at airports and the subsequent gate assignment process). Three key actions can be selected:

1. **Miles in Trail (MIT):** Agents control the distance aircraft have to keep from each other while approaching a fix. With a higher MIT value, fewer aircraft will be able to go through a particular fix during congested periods, because aircraft will be slowing down to keep their spacing. Therefore setting high MIT values can be used to reduce congestion downstream of a fix.
2. **Ground Delays:** An agent controls how long aircraft that will eventually go through a fix should wait on the ground. Imposing a ground delay will cause aircraft to arrive at a fix later. With this action, congestion can be reduced if some agents choose ground delays and others do not, as this will spread out the congestion. However, note that if all the agents choose the same ground delay, then the congestion will simply happen at a later moment in time.
3. **Rerouting:** An agent controls the routes of aircraft going through its fix, by diverting them to take other routes that will (in principle) avoid the congestion.

The third issue that needs to be addressed is what type of learning algorithm each agent will use. The selection of the agent and action space (as well as the state space) directly impacts this choice. Indeed, each agent aims to select the action that leads to the best system performance, G (given in Equation 10.21). For delayed-reward problems, sophisticated reinforcement learning systems such as

temporal difference may have to be used. However, due to our agent selection and agent action set, the air traffic congestion domain modeled in this paper only needs to utilize immediate rewards. As a consequence, a simple table-based immediate reward reinforcement learner is used. Our reinforcement learner is equivalent to an ϵ -greedy action-value learner [72]. At every episode an agent takes an action and then receives a reward evaluating that action. After taking action a and receiving reward R an agent updates its value for action a , $V(a)$ (which is its estimate of the value for taking that action [72]) as follows:

$$V(a) \leftarrow (1 - \lambda)V(a) + (\lambda)R, \quad (10.25)$$

where λ is the learning rate. At every time step, the agent chooses the action with the highest table value with probability $1 - \epsilon$ and chooses a random action with probability ϵ . In the experiments described in this chapter, λ is equal to 0.5 and ϵ is equal to 0.25. The parameters were chosen experimentally, though system performance was not overly sensitive to these parameters.

The final issue that needs to be addressed is selecting the reward structure for the learning agents. The first and most direct approach is to let each agent receive the system performance as its reward. However, in many domains such a reward structure leads to slow learning. We will therefore also set up a second set of reward structures based on agent-specific rewards. Given that agents aim to maximize their own rewards, a critical task is to create “good” agent rewards, or rewards that when pursued by the agents lead to good overall system performance. In this work we focus on “difference rewards,” which aim to provide a reward that is both sensitive to that agent’s actions and aligned with the overall system reward [3, 80, 102]. This difference reward is of the form [5, 78, 80, 102]

$$D_i(a) \equiv G(a) - G(a_{-i} + c_i), \quad (10.26)$$

where a_{-i} denotes the action of all agents other than i , and c_i is a constant “action” that replaces agent i ’s actual action.²

In this domain, D_i cannot be directly computed, as G cannot be expressed in analytical form, and depends on sector counts obtained from a simulator. This is a key issue with difference objectives, and various estimates have been proposed to overcome this issue [6, 57, 77]. For example, precomputed difference rewards or estimates based on the functional form of G have provided good results [6, 77]. Similarly, modeling G and using that model to estimate D has shown promise in newer applications to air traffic [57]. Below, we will briefly provide results from the first set of results.

²This notation uses zero padding and vector addition rather than concatenation to form full state vectors from partial state vectors.

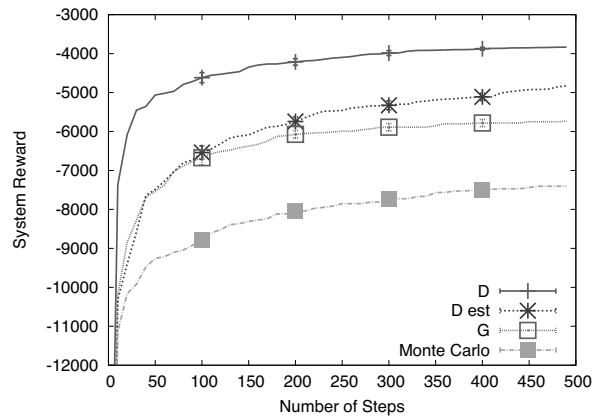


Figure 10.11: Performance of agents controlling miles in trail on the congestion problem, with 1,000 aircraft, 20 agents, and $\alpha = 5$.

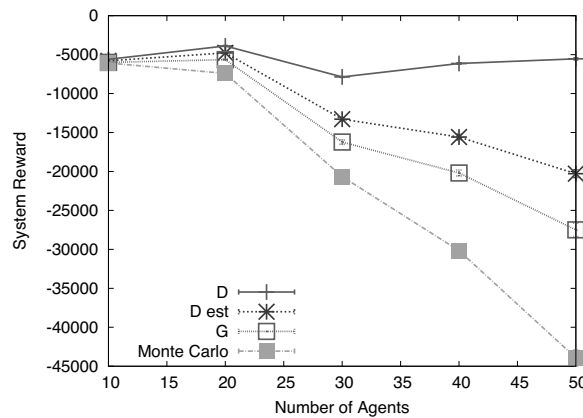


Figure 10.12: Scaling properties of agents controlling miles in trail on the congestion problem, with $\alpha = 5$. Number of aircraft is proportional to the number of agents, ranging from 500 – 2,500 aircraft (note that sector capacities have been also scaled according to the number of aircraft).

7.4 Multiagent Air Traffic Results

We present the performance of the multiagent approach in two types of congestion (see Figures 10.11 and 10.12). The first one consists of 1,000 aircraft, where 600 of the aircraft are going through an area of high congestion, while 400 aircraft are going through an area of moderate congestion. The second scenario consists of real-world historical data in the Chicago and New York areas. All experiments are for an eight hour window of traffic.

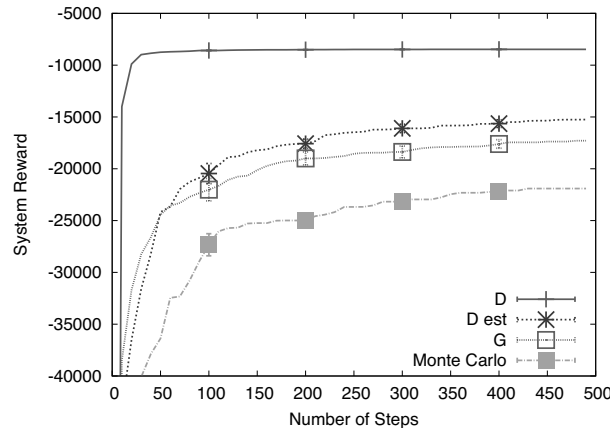


Figure 10.13: Performance of agents controlling miles in trail on historical data from New York area. Agents control 1,577 aircraft.

In all experiments the goal of the system is to maximize the system performance given by $G(z)$ with the parameters, $\alpha = 5$. In all experiments to make the agent results comparable to the Monte Carlo estimation, the best policies chosen by the agents are used in the results. All results are an average of thirty independent trials with the differences in the mean (σ/\sqrt{n}) shown as error bars, though in most cases the error bars are too small to see.

We also performed experiments with real dates from the Chicago and New York areas. We present the results from the New York area where congestion peaks at certain times during the day. This is a difficult case, since in many situations slowing down aircraft to avoid a current congestion just adds to aircraft in a future congestion. The results show (Figure 10.13) that agents using D were able to overcome this challenge and significantly outperform agents using the other rewards. In addition agents using the estimate of D were able to perform better than agents using G .

7.5 Summary

In this section we presented the applicability of the discussed methods to a difficult real-world domain. Indeed, the efficient, safe, and reliable management of air traffic flow is a complex problem, requiring solutions that integrate control policies with time horizons ranging from minutes up to multiple days. The presented results show that a multiagent learning approach provides significant benefits in this difficult problem. The keys to success were in defining the agents (agents as fixes), their actions (miles in trail), their learning algorithm (action-value learners), and their reward structure (difference rewards).

8 Conclusions

Multiagent learning is a young and exciting field that has produced many interesting research results and has seen a number of important developments in a relatively short period of time. This chapter has introduced the basics, challenges, and state-of-the-art of multiagent learning. Foundations of different multiagent learning paradigms have been introduced, such as reinforcement learning, evolutionary game theory, swarm intelligence, and neuro-evolution. Moreover, examples and pointers to the relevant literature of the different paradigms have been provided. On top of this a real-world case study on multiagent learning, from the air traffic control domain, has been extensively described.

Over the past years multiagent learning has seen great progress at the intersection of game theory and reinforcement learning due to its strong focus on this intersection. Recently, the field also started to take a broader and more interdisciplinary approach to MAL, which is an important step toward efficient multiagent learning in complex applications. As an example we have introduced and discussed the potential of evolutionary game theory, swarm intelligence, and neuro-evolutionary approaches for MAL. Specifically, we believe that in order to be successful in complex systems, explicit connections should be drawn between the different paradigms.

9 Exercises

1. **Level 1** Create your own MDP for a domain that you consider interesting. You should clearly specify the states, actions, transition probabilities, and rewards. There should be at least three states and at least three actions. Try to make it a relatively “interesting” MDP, i.e., make it so that there are not too many deterministic transitions, and that it is not immediately obvious what the optimal policy is. Draw your MDP.
2. **Level 1** Calculate the value $V(i)$ for each state i in the table shown on page 469, given a discount factor of 0.9. Hint: draw a diagram first.
3. **Level 1-2** The subsidy game can be described by the following situation. There are two new standards that enable communication via different protocols. The consumers and suppliers can be described by probability vectors that show which standard is supported by which fractions. One protocol is 20% more energy efficient, hence the government wants to support that standard. Usually, the profit of the consumers and suppliers are directly proportional to the fraction of the corresponding type that supports its standard. However, the government decides to subsidize early adopters of the better

State	Reward (state)	Action consequence	Probability that consequence occurs	Reward (action)
1	0	Go to 2	0.8	-2
		Go to 4	0.2	+1
2	0	Go to 1	0.2	-1
		Go to 3	0.8	-2
3	0	Go to 4	0.3	-2
		Go to 5	0.7	10
4	0	Go to 2	0.5	-2
		Go to 3	0.5	+3
5	10	-	-	-

protocol. Such subsidies are expensive and the government only wants to spend as much as necessary. They have no market research information and consider any distribution of supporters on both sides equally likely. Furthermore, they know that the supporters are rational and their fractions will change according to the replicator dynamics.

This game is a variation of the pure coordination game. A subsidy parameter $s \in \{0, 11\}$ is added, which can be used to make one action dominant. Figure 10.14 illustrates the game in its general form.

$$\begin{matrix} & s_1 & s_2 \\ \begin{matrix} s_1 \\ s_2 \end{matrix} & \left(\begin{array}{cc} 10, 10 & 0, s \\ s, 0 & 12, 12 \end{array} \right) \end{matrix}$$

Figure 10.14: The subsidy matrix game.

Applying the replicator dynamics (without mutation) to the two instances of the subsidy game, shown in Figure 10.15, yields the direction field plots shown in Figure 10.16.

- Identify the Nash equilibria in the game; is (are) there Pareto optimal solution(s)?
- Which one(s) is (are) evolutionary stable? Why? What can you say about the basins of attraction?
- What is the effect of adding the subsidy in the second instance of the game?

$$\begin{array}{cc}
 \begin{array}{c} s_1 \\ s_2 \end{array} \begin{array}{cc} s_1 & s_2 \\ \left(\begin{array}{cc} 10, 10 & 0, 0 \\ 0, 0 & 12, 12 \end{array} \right) \end{array} &
 \begin{array}{c} s_1 \\ s_2 \end{array} \begin{array}{cc} s_1 & s_2 \\ \left(\begin{array}{cc} 10, 10 & 0, 11 \\ 11, 0 & 12, 12 \end{array} \right) \end{array} \\
 \text{Instance 1} & \text{Instance 2}
 \end{array}$$

Figure 10.15: Subsidy game instances.

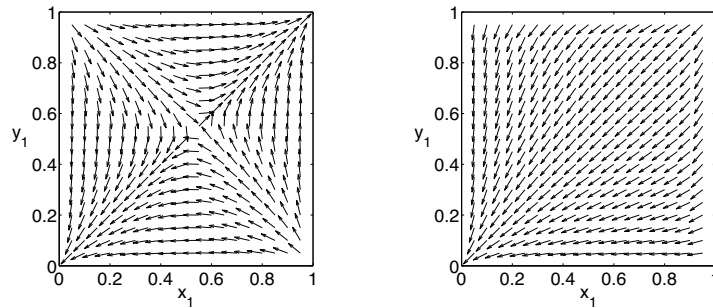


Figure 10.16: The dynamics of the game without subsidy (left) and with subsidy (right).

4. **Level 2-3** In this exercise perform an EGT comparison of Q-learning, FAQ-learning, LFAQ-learning, and regret minimization in self-play. For this purpose do the following:
- Implement the evolutionary dynamics of Q-learning and FAQ-learning for the 2×2 matrix games Prisoner's Dilemma, Battle of the Sexes, and Matching Pennies (e.g., in Matlab). This means that you should be able to visualize the direction fields of these algorithms for different parameter settings. Test your code by plotting the direction fields for FAQ-learning and compare these to the ones provided in Section 4.
 - Implement Q-learning and FAQ-learning for these games (e.g., in Matlab). Again test your code by comparing the plots of the learning traces to the ones provided in Section 4.
 - Now also implement LFAQ-learning and regret minimization and the evolutionary dynamics for these games. Compare your findings for LFAQ with those provided in Section 4.
 - Conduct an extensive comparison of the four algorithms by varying the parameter settings of the evolutionary dynamics. Draw conclusions on the performance of the algorithms.

5. **Level 4** Repeat the previous exercise (same games), but now perform the comparison in mixed play. In the mixed play experiments, games are played by heterogeneous pairs of players, meaning that both players implement different learning methods. Plot and compare the dynamics for the following cases:
- Q-learning vs. FAQ-learning
 - RM vs. FAQ-learning
 - FAQ-learning vs. LFAQ-learning
 - RM vs. LFAQ-learning
6. **Level 2-3** Implement the joint action learning algorithm for stateless games and compare its convergence behavior with (independent) Q-learning on the matrix games shown in Figures 10.17 and 10.18 (draw clear conclusions).

$$\begin{array}{c} a_0 \\ a_1 \end{array} \left(\begin{array}{cc} b_0 & b_1 \\ 10 & 0 \\ 0 & 10 \end{array} \right)$$

Figure 10.17: Repeated matrix game (general form).

	a_0	a_1	a_2
b_0	11	-30	0
b_1	-30	7	6
b_2	0	0	5

Figure 10.18: The climbing game.

7. **Level 1-2** This question concerns the swarm intelligence paradigm. Foraging is the task of locating and acquiring resources. Typically, this task has to be performed in an unknown and possibly dynamic environment. The problem consists of two phases. First, leaving the starting location (e.g., the nest) in search for food. Second, returning to the starting location loaded with food. In its simplest form (e.g., an open-field-like environment), this is a problem domain that can be solved by a single agent or multiple, independent agents which act in parallel. If agents want to solve the problem by cooperation, getting to the solution of the problem becomes more complex.

Performance can be measured in time used before all items are collected and the number of items collected in a certain time span. Foraging can be seen as an abstract problem with regard to more complex real-world problems such as network routing, information retrieval, transportation, and patrolling.

- Explain the essential similarities and differences, and advantages and disadvantages between ant and bee self-organization. Relate this to the foraging problem.
- Would it be possible to combine the best of both worlds, i.e., of bee and ant systems, into one new hybrid algorithm for the task of foraging? If yes, describe how; if no, motivate your answer.

8. **Level 2** Consider a congestion problem where each agent takes one of N_k actions. Each action has a parameter b_k and the number of agents that select action k is given by x_k . The total number of agents selecting the same action as agent i is denoted by x_{k_i} . The system-level objective function is:

$$G(z) = \sum_{k=1}^{N_k} (x_k - b_k)^2 \quad (10.27)$$

Now, someone gives you the following agent objective functions for agent i . Discuss each and why you would expect them to work or not work (use factoredness and/or learnability to make your point if necessary).

- (a) $g_i = (x_{k_i} - b_k)^2 - (x_{k_i})^2$
- (b) $g_i = (x_{k_i} - b_k)^2 - (x_{k_i} - b_k - 1)^2$
- (c) $g_i = (x_{k_i} - b_k)^2 - b_k^2$
- (d) $g_i = \sum_k^{n_k} (x_k - b_k)^2 - \sum_k^{N_k} b_k^2$

9. **Level 2** Repeat the previous exercise for the case where $b_k = b \ \forall k$. (That is, the action parameters are the same for all actions.) How do your answers change?
10. **Level 2** Consider a multiagent system with N agents and the following system-level objective function:

$$G(z) = \frac{|\sum_{i=1}^N a_i e_i|}{\sum_{k=1}^N a_k}, \quad (10.28)$$

where $a_i \in \{0,1\}$ is agent i 's binary action (e.g., agent i is active or not), and where $\{e_i\}$ is a specific value associated with each agent (the

state vector z in this case consists of the set of all the agents' actions, or $z = \{a_1, a_2, \dots, a_i, \dots, a_N\}$.

- (a) Derive the difference objective for agent i given $c_i = 0$ (i.e., the agent removes itself from the system). Simplify your answer.
 - (b) Derive the difference objective for agent i given $c_i = 0.5$ (i.e., the agent takes half an action, an average over its possible choices). Simplify your answer.
 - (c) Both difference objectives are factored by definition. What can you deduce about their learnability? Why?
11. **Level 2** Consider a 5 x 10 gridworld. The agent starts at a random location and has four actions (moves in each of the four directions). There is a reward of 100 to catch T1, a target that starts at the bottom right square and moves randomly by one square at each time step. In addition, there is a reward of -2 for every time step the agent is in this gridworld.
 - (a) Devise a reinforcement algorithm to catch T1. Clearly state all system parameters (inputs, states, outputs, etc.)
 - (b) Now, the target uses a different algorithm. Instead of moving randomly, it moves in a direction opposite the agent in each time step, if possible, and randomly if not. Can the agent still catch the target? Explain the behavior of the system.
 - (c) Now, have two agents start at the same location and move in this grid. What behavior do you observe now? Do the agents get a benefit from each other? If so, is that direct or incidental? Do they cooperate?
 - (d) Suggest one simple modification to the learning that will make agents cooperate more explicitly.
12. **Level 3** Repeat the previous exercise when the agent uses a neural network to map its states to actions directly and uses a neuro-evolutionary algorithm to train this neural network. Can you suggest a method to coevolve the agents so that they directly cooperate?
13. **Level 3** Arthur's El Farol bar problem [7] is a perfect example of a congestion game. In this problem each agent i decides whether to attend a bar by predicting, based on its previous experience, whether the bar will be too crowded to be "rewarding" at that time, as quantified by a system reward G . The congestion game structure means that if most agents think the attendance will be low (and therefore choose to attend), the attendance will

actually be high, and vice versa. Give a modified version of this problem where the N agents pick one out of K nights to attend the bar every week. The system reward in any particular week is [4, 102]:

$$G(z) \equiv \sum_{k=1}^K x_k(z) e^{\frac{-x_k(z)}{b}}, \quad (10.29)$$

where $x_k(z)$ is the total attendance on night k ; and b is a real-valued parameter.

- (a) Derive a simple “local” reward for each agent in this problem (e.g., the agent reward should depend on information easily available to the agent). Discuss the factoredness and learnability of the reward you derived.
- (b) Derive a difference reward for each agent. What is a good “fixed vector” c_i for this case? For at least two values of c_i , discuss the locality of the information and the factoredness and learnability of the resulting difference rewards.
- (c) Perform a simulation for this problem with the following parameters: (i) capacity of each night is 4 ($b = 4$), $k = 6$, and there are 30 agents in the system; and (ii) capacity of each night is 4 ($b = 4$), $k = 5$, and there are 50 agents in the system.

Plot the performance of three agent rewards (G , difference, and local) and a histogram of sample attendance profiles. Discuss the simulation results.

14. **[Level 3]** The degree of *factoredness* of an agent reward function is defined by [4, 80]:

$$\mathcal{F}_{g_i} = \frac{\sum_{z'} u[(g_i(z) - g_i(z'))(G(z) - G(z'))]}{\sum_{z'} 1} \quad (10.30)$$

where the states z and z' only differ in the states of agent i , and $u[x]$ is the unit step function, equal to 1 if $x > 0$. The numerator counts the number of states where $g_i(z) - g_i(z')$ and $G(z) - G(z')$ have the same sign, and the denominator counts the total number of states. Intuitively, the degree of factoredness gives the fraction of states in which a change in the state of agent i has the same impact on g_i and G . A high degree of factoredness means that the agent reward g_i moves in the same direction (up or down) as the global reward G based on a change to the system state. A system in which all the agent rewards equal G has a degree of factoredness of 1.

Now, this definition of alignment does not take into account by how much g_i and G change, nor does it take into account which parts of the state space are likely to be visited. Provide two improvements to the concept of factoredness to allow better prediction of whether an agent reward will lead to good system behavior:

- (a) Define a new concept of reward alignment that is maximized when *large* improvements to g_i lead to *large* improvements of G .
 - (b) Define a new concept of reward alignment where states that are more likely to be visited have higher weight in the computation of alignment, and having g_i and G misaligned on states unlikely to be visited do not impact the computation of alignment.
15. **Level 4** Based on your new definition quantifying the degree of alignment between a system reward and an agent reward, derive a new agent reward that will lead to good system behavior. (Hint: the difference reward presented in Section 2 leads to a fully factored system; derive an agent reward that will lead to a fully “aligned” system with your new definition of alignment.)

References

- [1] S. Abdallah and V. R. Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research (JAIR)*, 33:521–549, 2008.
- [2] A. Agogino and K. Tumer. Learning indirect actions in complex domains: Action suggestions for air traffic control. *Advances in Complex Systems*, 12:493–512, 2009.
- [3] A. Agogino, K. Tumer, and R. Miikulainen. Efficient credit assignment through evaluation function decomposition. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.
- [4] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.
- [5] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.
- [6] A. K. Agogino and K. Tumer. A multiagent approach to managing air traffic flow. *Journal of Autonomous Agents and Multi-Agent Systems*, 2011. DOI: 10.1007/s10458-010-9142-5.

- [7] W. B. Arthur. Complexity in economic theory: Inductive reasoning and bounded rationality. *The American Economic Review*, 84(2):406–411, May 1994.
- [8] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1–2):81–138, 1995.
- [9] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.
- [10] K. D. Bilimoria, B. Sridhar, G. B. Chatterji, K. S. Shethand, and S. R. Grabbe. FACET: Future ATM concepts evaluation tool. *Air Traffic Control Quarterly*, 9(1), 2001.
- [11] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.
- [12] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.
- [13] T. Borgers and R. Sarin. Learning through reinforcement and replicator dynamics. *Journal of Economic Theory*, 77:115–153, 1997.
- [14] M. Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems, volume 17, pages 209–216*, 2004.
- [15] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.
- [16] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, March 2008.
- [17] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Swarm intelligence for routing in mobile ad hoc networks. In *Swarm Intelligence Symposium (SIS), Proceedings*, 2005.
- [18] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, pages 1471–1496, September 1999.
- [19] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multi-agent systems. In *Proceedings of the 15th International Conference on Artificial Intelligence, pages 746–752*, 1998.
- [20] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *20th International Conference on Machine Learning (ICML)*, pages 83–90, 2003.

- [21] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.
- [22] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence*, 13:227–303, 2000.
- [23] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.
- [24] J. A. Fernandez-Leon, G. G. Acosta, and M. A. Mayosky. Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation. *Robotics and Autonomous Systems*, pages 411–419, 2008.
- [25] H. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*. Princeton University Press, 2001.
- [26] F. Gomez and R. Miikkulainen. Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1356–1361, Stockholm, Sweden, 1999.
- [27] F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.
- [28] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artif. Intell. Res. (JAIR)*, 19:399–468, 2003.
- [29] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.
- [30] D. Hennes, K. Tuyls, and M. Rauterberg. State-coupled replicator dynamics. In *AAMAS (2)*, pages 789–796, 2009.
- [31] J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, 1998.
- [32] F. Hoffmann, T.-J. Koo, and O. Shakernia. Evolutionary design of a helicopter autopilot. In *Advances in Soft Computing - Engineering Design and Manufacturing, Part 3: Intelligent Control*, pages 201–214, 1999.
- [33] J. Hu and M. P. Wellman. Experimental results on Q-learning for general-sum stochastic games. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 407–414. Morgan Kaufmann Publishers Inc., 2000.
- [34] J. Hu and M. P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

- [35] C. Jafari, A. Greenwald, D. Gondek, and G. Ercal. On no-regret learning, fictitious play, and Nash equilibrium. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 223–226, 2001.
- [36] L. P. Kaelbling, M. L. Littman, and A.W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res. (JAIR)*, 4:237–285, 1996.
- [37] M. Kaisers and K. Tuyls. Frequency adjusted multi-agent Q-learning. In *AAMAS*, pages 309–316, 2010.
- [38] T. Klos, G. J. van Ahee, and K. Tuyls. Evolutionary dynamics of regret minimization. In *ECML/PKDD (2)*, pages 82–96, 2010.
- [39] M. Knudson and K. Tumer. Coevolution of heterogeneous multi-robot teams. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 127–134, Portland, WA, July 2010.
- [40] M. Knudson and K. Tumer. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems*, 59:410–420, 2011.
- [41] N. Lemmens and K. Tuyls. Stigmergic landmark foraging. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 497–504. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
- [42] N. Lemmens and K. Tuyls. Stigmergic landmark optimization. In *Advances in Complex Systems*, 2012, to appear.
- [43] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.
- [44] M. L. Littman. Friend-or-foe Q-learning in general-sum games. In *ICML*, pages 322–328, 2001.
- [45] M. J. Mataric. Coordination and learning in multi-robot systems. In *IEEE Intelligent Systems*, pages 6–8, March 1998.
- [46] J. Maynard-Smith and J. Price. The logic of animal conflict. *Nature*, 146:15–18, 1973.
- [47] M. McGlohon and S. Sen. Learning to cooperate in multi-agent systems by combining Q-learning and evolutionary strategy. *International Journal on Lateral Computing*, 1(2):58–64, 2005.
- [48] D. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.

- [49] K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall International, Inc, 1989.
- [50] L. Panait. Improving coevolutionary search for optimal multiagent behaviors. In *International Joint Conference on Artificial Intelligence*, pages 653–658. Morgan Kaufmann, 2003.
- [51] L. Panait, S. Luke, and R. P. Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.
- [52] L. Panait, K. Tuyls, and S. Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research (JMLR)*, 2008.
- [53] S. Parsons and M. Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(12):243–254, 2002.
- [54] M. Pechoucek, D. Sislak, D. Pavlicek, and M. Uller. Autonomous agents for air-traffic deconfliction. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Hakodate, Japan, May 2006.
- [55] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.
- [56] W. B. Powell and B. van Roy. Approximate dynamic programming for high-dimensional dynamic resource allocation problems. In J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, editors, *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, Hoboken, NJ, 2004.
- [57] S. Proper and K. Tumer. Modeling difference rewards for multiagent learning (extended abstract). In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, June 2012.
- [58] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 1994.
- [59] R. Roberts, C. Pippin, and T. Balch. Learning outdoor mobile robot behaviors by example. *Journal of Field Robotics*, 26(2):176–195, 2009.
- [60] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. In *Technical report*, 1994.
- [61] M. Salichon and K. Tumer. A neuro-evolutionary approach to micro aerial vehicle control. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1123–1130, Portland, WA, July 2010.

- [62] S. Sen, S. Airiau, and R. Mukherjee. Towards a Pareto-optimal solution in general-sum games. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–160, 2003.
- [63] J. Shepherd III and K. Tumer. Robust neuro-control for a micro quadrotor. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1131–1138, Portland, WA, July 2010.
- [64] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artif. Intell.*, 171(7):365–377, 2007.
- [65] S. P. Singh, M. J. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *UAI*, pages 541–548, 2000.
- [66] D. Sislak, M. Pechoucek, P. Volf, D. Pavlicek, J. Samek, V. Mařík, and P. Losiewicz. AGENTFLY: Towards multi-agent technology in free flight air traffic control. In Michal Pechoucek, Simon Thompson, and Holger Voss, editors, *Defense Industry Applications of Autonomous Agents and Multi-Agent Systems*, chapter 7, pages 73–97. Birkhauser Verlag, 2008.
- [67] D. Sislak, J. Samek, and M. Pechoucek. Decentralized algorithms for collision avoidance in airspace. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 543–550, Estoril, Portugal, May 2008.
- [68] B. Sridhar, T. Soni, K. Sheth, and G. B. Chatterji. Aggregate flow model for air-traffic management. *Journal of Guidance, Control, and Dynamics*, 29(4):992–997, 2006.
- [69] K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002.
- [70] P. Stone. *Layered Learning in Multiagent Systems*. MIT Press, 2000.
- [71] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.
- [72] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
- [73] P. J. ’t Hoen, K. Tuyls, L. Panait, S. Luke, and H. la Poutré. An overview of cooperative and competitive multiagent learning. In *Learning and Adaptation in Multi-Agent Systems*, pages 1–50. Springer LNAI 3898, 2006.

- [74] M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods for reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, Seattle, WA, July 2006.
- [75] D. Teodorovic and M. Dell’Orco. Bee Colony Optimization: A cooperative learning approach to complex transportation problems. In *Proceedings of the 16th Mini - EURO Conference and 10th Meeting of EWGT*, 2006.
- [76] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:33–53, 1992.
- [77] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 330–337, Honolulu, HI, May 2007.
- [78] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 378–385, Bologna, Italy, July 2002.
- [79] K. Tumer and N. Khani. Learning from actions not taken in multiagent systems. *Advances in Complex Systems*, 12:455–473, 2009.
- [80] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, 2004.
- [81] K. Tumer and D. Wolpert. A survey of collectives. In *Collectives and the Design of Complex Systems*, pages 1–42. Springer, 2004.
- [82] K. Tuyls, D. Heytens, A. Nowe, and B. Manderick. Extended replicator dynamics as a key to reinforcement learning in multi-agent systems. In *14th European Conference on Machine Learning, volume 2837 of Lecture Notes in Computer Science*, pages 421–431, 2003.
- [83] K. Tuyls and S. Parsons. What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, 171(7):406–416, 2007.
- [84] K. Tuyls, P. J. ’t Hoen, and B. Vanschoenwinkel. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Journal of Autonomous Agents and Multi-Agent Systems*, 12:115–153, 2006.
- [85] K. Tuyls, K. Verbeeck, and T. Lenaerts. A selection-mutation model for Q-learning in multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems. ACM Press, Melbourne, Australia*, 2003.

- [86] K. Tuyls, K. Verbeeck, and T. Lenaerts. A selection-mutation model for Q-learning in multi-agent systems. In *AAMAS*, pages 693–700, 2003.
- [87] K. Tuyls and G. Weiss. Multiagent learning: Basics, challenges, and prospects. In *AI Magazine*, 2012, to appear.
- [88] K. Tuyls and R. Westra. Replicator dynamics in discrete and continuous strategy spaces. In A. M. Uhrmacher and D. Weyns, editors, *Agents, Simulation and Applications*. Taylor and Francis, 2008.
- [89] K. Verbeeck, A. Nowe, M. Peeters, and K. Tuyls. Multi-agent reinforcement learning in stochastic single and multi-stage games. In *Adaptive Agents and Multi-Agent Systems II*, pages 275–294. Springer LNAI 3394, 2005.
- [90] K. Verbeeck, A. Nowe, and K. Tuyls. Coordinated exploration in multi-agent reinforcement learning: An application to load-balancing. In *Proceedings of the Fifth Joint Conference on Autonomous Agents and Multi-Agent Systems, Utrecht, The Netherlands*, 2005.
- [91] J. M. Vidal. Multiagent coordination using a distributed combinatorial auction. In *AAAI Workshop on Auction Mechanism for Robot Coordination*, July 2006.
- [92] J. M. Vidal and E. H. Durfee. The moving target function problem in multi-agent learning. In *Proceedings of the Third International Conference on Multi-Agent Systems*, pages 317–324. AAAI/MIT press, July 1998.
- [93] P. Vrancx, K. Tuyls, R. Westra, and A. Nowe. Switching dynamics of multi-agent learning. In *Proceedings of the Seventh Joint Conference on Autonomous Agents and Multi-Agent Systems, Estoril, Portugal*, 2008.
- [94] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15:918–933, 1999.
- [95] C. Watkins. *Learning with Delayed Rewards*. PhD thesis, Cambridge University, 1989.
- [96] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.
- [97] J. W. Weibull. *Evolutionary Game Theory*. MIT Press, 1996.
- [98] S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection for reinforcement learning. *Adaptive Behavior*, 15(1), 2007.
- [99] D. Whitley, F. Gruau, and L. Pyeatt. Cellular encoding applied to neurocontrol. In *International Conference on Genetic Algorithms*, 1995.

-
- [100] R. Paul Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, Fairfax, Virginia, 2004.
 - [101] M. Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, Universiteit van Amsterdam, 1999.
 - [102] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.
 - [103] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936, 2003.

Chapter 11

Multiagent Planning, Control, and Execution

Ed Durfee and Shlomo Zilberstein

1 Introduction

Planning is important to an agent because its current and upcoming choices of actions can intentionally establish, or accidentally undo, the conditions that later actions depend upon to reach desirable states of the world. Hence, planning in single-agent systems is concerned with how an agent can efficiently model and select from alternative sequences of actions, preferably without considering every possible sequence. In a multiagent setting, the added complication is that decisions an agent makes about near-term actions can impact the future actions that *other agents* can (or cannot) take. Similarly, knowing what actions other agents plan to take in the future could impact an agent's current action choices. And, unlike single agents, multiple agents can act concurrently. Therefore an agent's choice of action at any given time can impact and be impacted by the action choices of other agents at the same time. Because the space of possible joint courses of action the agents could take grows exponentially with the number of agents (as we will detail later), planning in a multiagent world is inherently intractable, a problem that is compounded in dynamic, partially-observable, and/or non-deterministic environments. Yet, when agents are cooperative, as we will assume in this chapter, then they should strive to make decisions that collectively over time achieve their joint objectives as effectively as possible.

The above paragraph captures in a simplistic way the themes of this chapter. By *multiagent control*, we refer to how agents in a multiagent system can be provided with and utilize information to make better decisions about what to do *now* so that their joint actions can further the achievement of joint objectives. *Multiagent planning* focuses not just on current decisions, but also on sequences of decisions, allowing an agent to “look ahead” so as to establish conditions for another agent that allow it to achieve desired shared goals. From the cooperative perspective, an agent should be willing to incur local cost if by doing so it enables other agents to achieve benefits that more than offset that cost. *Multiagent execution* extends and in some senses combines multiagent planning and control, where agents both proactively plan their (inter)actions to guide the evolution of their shared environment, but also reactively control their behaviors in response to emergent or unlikely events.

In single-agent planning, a key to getting traction on solving hard problems is to exploit structure, typically involving notions of locality and composition. An agent’s *state* is typically composed of *features* (for example, propositions about what facts are true in the agent’s environment), and an action the agent can take typically involves only a small subset (*locality*) of features. By focusing only on features of interest and the actions that involve them, an agent can focus its search only on a much smaller space of relevant plans. Similarly, as we will see in this chapter, solving multiagent planning and control problems can depend critically on exploiting problem structure, where that structure extends to locality of interacting agents (e.g., an agent’s action choices only directly affect a few other agents), and locality of involved features (e.g., few agents can directly affect any particular feature, and/or only a few of the features an agent cares about can be affected by others). Agents can exploit such structure to formulate their joint plans through *composition*: the solutions for the different localities can be combined (relatively) straightforwardly into a comprehensive multiagent solution.

This chapter builds on the topics of the previous chapters to describe the concepts and algorithms that comprise the foundations of multiagent control, planning, and execution. We assume that the reader is already familiar with protocols of interaction; here those protocols are used in the context of coordinating cooperative multiagent action. We also assume the reader is familiar with traditional AI search techniques, planning algorithms and representations, and models for reasoning under uncertainty. We make liberal use of the relevant concepts as we delve into their multiagent analogues.

2 Characterizing Multiagent Planning and Control

As with many topics in multiagent systems (and artificial intelligence, and computer science ...), a phrase like “multiagent planning” or “multiagent control” can mean different things to different people. We do not claim that our characterization here is necessarily the consensus opinion of the community, but it should give the reader of this chapter a sense of the problems that are (and are not) within the space of problems considered here.

Multiagent planning is something of an ambiguous term, because it is unclear exactly what is “multiagent.” It could be that the operative issue is that, as a consequence of possibly centralized planning, a plan is formulated that can be distributed across and acted upon by a set of agent systems. Alternatively, the operative issue could be that the planning process should itself be multiagent, whether or not the resulting plan(s) are. Or perhaps both issues are of interest.

In this chapter, we consider both multiagent plans and multiagent plan formation as requirements. The case where neither holds is simply traditional single-agent planning. The case where multiple agents cooperatively generate a single-agent plan (such as where a set of planning specialists can contribute to the formation of a plan for the manufacture of an artifact [34], is effectively an instance of cooperative problem solving, where the problem being solved is the construction of a plan. Depending on the nature of the plan being devised, techniques such as distributed constraint satisfaction [67], distributed constraint optimization (Chapter 12), or distributed search [62] can be employed to jointly construct such a plan. Finally, consider the case where a centralized planner builds a detailed collection of plans to distribute among the agents, such that the behavior of each agent is precisely dictated. While beautifully coordinated behavior can ensue, the agents are stripped of any autonomy, and are thus arguably not agents in any interesting sense any more. In essence, this is multi-effector planning, rather than multiagent planning.

If the centralized planner provides less detailed guidance, however, then agents can exercise their “agent” attributes to utilize local awareness of the world, along with local preferences, knowledge, and capabilities, to more autonomously and individually decide on current actions, and even to plan future actions. When this occurs, then the multiagent plan and the plan formation process are both inherently distributed among agents: no single agent forms or even might be aware of the entire joint plan, since different agents may have made their own local plan elaborations and refinements.

What happens if there is no centralized planner, and hence no centralized guidance, at all? This is an interesting question. If we say that the agents could cooperatively converge on an effective (distributed) joint plan without a centralized planner, where did the guidance come from to do so? Typically, some central-

izing entity (an agent, a human system designer, a group of people comprising a standards body) will have devised and disseminated some guidelines, such as interaction plans (aka protocols) and the rules for using them, which the agents count upon to communicate and cooperate with each other. How and whether the environment can itself provide the structure to allow dissimilar agents to converge on cooperative plans for non-trivial problems, or can engender the unguided emergence of languages and protocols that enable cooperation, is beyond the scope of this chapter.

The preceding thus sets the table for our exploration of multiagent planning and control techniques in this chapter. We begin in Section 3 with looking at the process of creating centralized guidelines that push agents to make good control decisions (about current actions) and/or planning decisions (about sequences of actions). As we shall see, in some cases these guidelines can guarantee that the control decisions or plans that each agent makes in adherence to the guidelines *must* be jointly coordinated. In such cases, coordination precedes planning.

We then turn to the opposite (Section 4), when planning precedes coordination, where the guidelines are weaker (typically, more general-purpose) and hence do not constrain the space of joint plans much. Instead, each agent can elaborate its own plan to achieve its assigned goals, and then these plans are coordinated by, for example, adjusting the timing of agents' activities to preclude interference.

Unfortunately, for a variety of interesting problems, including problems where unexpected events can occur at runtime, (local) planning of individual actions and (multiagent) coordination of interactions need to be done together, in an interleaved manner. In Section 5, we look at how such multiagent sequential decision-making problems can be formulated as decentralized (partially-observable) Markov decision processes, and describe techniques for finding optimal and approximately-optimal solutions (joint plans) in such problems.

Finally, finding good joint plans is only useful if agents can successfully execute them. Since planning is done using a model of the environment, and that model might not correctly represent the actual environment at the time the plan is executed, agents should monitor the progress of their plans against expectations, and repair their joint plans in response to deviations. These ideas are familiar (though still challenging) in the single-agent planning world; we conclude this chapter (Section 6) describing strategies to handle similar problems in the multi-agent setting.

3 Coordination Prior to Local Planning

Developers of distributed systems typically anticipate how entities within a system might need to interact, and predefine interaction plan templates for the en-

tities to fill in and follow. Examples of such interaction plan templates abound in this book. These templates can take the form of interagent protocols, defining the possible sequences of communicative acts between agents, where the content of these acts can be domain dependent. For example, agents solving a distributed constraint satisfaction problem follow protocols for exchanging information about tentative assignments of values to variables, or of no-good assignments that collectively violate constraints [67]. As another example, agents solving a resource allocation problem can work within auction mechanisms that have been designed to cause information exchanges to converge on efficient allocations (Chapter 7).

3.1 Social Laws and Conventions

We begin with a simple strategy to ensure sufficient coordination of agents' actions, a strategy that has been characterized as imposing *social laws* on agents [55, 56]. The idea is to identify joint states that should not be allowed to arise, and to impose restrictions on agents' action choices to prevent them.

A canonical application domain for social laws is in coordinating mobile robots. Collisions between robots leads to system degradation (robots become disabled) and cost (robots need repairs), and thus should be avoided. If space is discretized, such as modeling it as a grid, then states where two or more robots are in the same grid coordinate should be prevented. Thus, one law to impose on the robots is that a robot should never move into a neighboring location that is occupied.

A moment's reflection reveals that such a law is insufficient, because it fails to prevent two or more robots from simultaneously entering the same empty location from different directions. One way to strengthen the social law is to prohibit agents from entering a location from more than one direction. If each location in the grid is to be reachable from every other location, this stronger law effectively defines a directed cycle through the grid locations such that each location is visited exactly once.

The stronger social law leads to agents moving through the locations in a sort of "conga line," where each can move to its next location when that location is empty. Agents do not need to coordinate their action choices as they decide where they want to go, because so long as agents obey the law, collisions cannot arise. However, such prebuilt coordination generally comes at a price. An agent might take a very circuitous route to get to a desired destination because of the social law, when it could have potentially gotten where it wanted much more directly and safely because other agents were far away. In human terms, going the wrong way on a one-way street might be more efficient late at night when the odds of encountering oncoming cars is negligible. But deciding when it is safe to break such laws requires agents to reason about (and often communicate with) each other. A

purpose of social laws is to relieve agents of the burden of explicitly coordinating, potentially at the price of some degree of inefficiency in joint behaviors.

The flip side of social laws that tell agents what they are prohibited from doing in certain circumstances is the notion of *conventions*, which tell agents what they should (or must) do. The conceptual framework for conventions is the same as for social laws, which is to identify undesirable joint situations and to constrain agents to actions that avoid them. A canonical application domain for conventions is when agents share joint intentions [33], such that they have committed to work together on achieving some mutually-desired goal. If, in the midst of pursuing this joint goal, an agent comes to believe that the goal is unachievable, then it would be irrational for the agent to continue pursuing it. However, a state in which some agents are continuing to pursue a joint goal while others have dropped it as unachievable is arguably an undesirable state, since the former agents are taking futile actions. Hence, agents in the joint intentions framework follow a convention that they must notify each other if they come to believe the joint goal cannot be achieved.

Other flavors of these concepts have been introduced, such as that agents should return shared resources to their default state after usage (e.g., putting a tool back where it belongs when finished using it) or even go slightly out of their way to make a shared environment more conducive to goal achievement for other agents (e.g., widening a path while following it to make its traversal easier for later agents) [29]. The algorithmic model shared by them all is:

1. Identify joint states that should be avoided (or sought).
2. Work backward through agents' joint actions to identify possible precursor states to these states.
3. Impose constraints on agents' action choices in the precursor states to prevent (or require) joint actions accordingly.

Note that this process can recurse. If a precursor of a state to avoid leads inexorably to the undesirable state, then the precursor state should be added to the states to be avoided, and the algorithm should work backwards from it too. Similarly, if there is a way to go assuredly to a sought state from its precursor, the precursor can be added to the set of sought states.

3.2 Organizational Structuring

While social laws and conventions apply equally to all agents, cooperation in some types of problems can be better achieved if agents are differentially biased in the actions they choose to, or choose not to, take. Organizational structures

are a familiar example of this in human institutions. An organizational structure defines, for example, a set of different organizational roles with identified responsibilities, and connections between roles to direct exchanges of information and to dictate authority relationships. A good human organizational structure is one that provides the people occupying each of the roles with guidance about how to prioritize their tasks and direct their communications such that their complementary actions dovetail together into an effective whole.

As described in Chapter 2, similar ideas can and have been incorporated in multiagent systems. Organizational structuring has been exploited in a variety of application domains for multiagent systems, such as disaster response and sensor networks. Since the sensor network application has been a mainstay of organizational structuring research for decades, we use it for illustrative purposes in this section.

The most obvious roles for agents in a distributed sensor network correspond with geographical regions: different sensor agents will be responsible for monitoring events near where they are located (or where they are now tasked with relocating to). Where sensor coverages overlap, responsibility for the overlapping region should be assigned, though perhaps not exclusively. That is, just as in human organizations where overlap between roles allows whomever is least burdened in the current situation to take on more of the shared responsibility, role overlap in multiagent systems also enables some degree of dynamic load balancing, and even fault tolerance.

Other forms of task decomposition within the distributed sensor network domain can lead to further refinement of roles. An agent with access to a particular sensory apparatus (e.g., acoustic instead of visual) might be given greater responsibility for monitoring for particular events. Agents might balance computational load by assigning responsibility for different kinds of phenomena among themselves. Some agents might be given greater responsibility for integrating interpretations from others rather than forming interpretations from raw data themselves.

3.2.1 Organizational Design

While the preceding says something about what an organizational structure does, the question remains about where it comes from. In general, the space of possible organizational designs for a non-trivial multiagent (including human) enterprise is vast, and the ability to predict organizational performance (particularly in human settings) is limited. Hence, while computational techniques have been used to study and extend organizational theory [10], no consensus strategy for forming organizations for systems of computational agents has emerged.

As outlined in Chapter 2, organizational designs can arise from the bottom up, by adopting and codifying emergent patterns of interactions between agents,

or can be constructed from the top down, by tasking one or more organizational designers with forming an organizational structure that the collection of agents then adopts. Both cases involve a search over (part of) the space of designs.

To make the design process more concrete, we here summarize one approach developed by Sims, Corkill, and Lesser [57]. A core idea is to view the design of an organization much like the creation of a hierarchical plan: given goals and environmental conditions, decompose the goals into component subgoals, identify agent roles whose preconditions are met by the environment and whose expected effects match the subgoals, and compose an organization out of the resultant roles. Then, match agents to the roles to instantiate the organization.

More precisely, the ORGANIZATIONSEARCH algorithm takes a sorted list of candidate partial organizations, and steps through the list until the following procedure returns:

1. Generate expansions of the candidate partial organization by finding a goal leaf in the decomposition hierarchy that has not been fully bound, and replacing it with either a role-goal binding indicating how it could be achieved, or with a subgoal tree that further decomposes it into subgoals.
2. Repeat step 1 until all leaves have associated roles.
3. Then use information about agent capabilities to assign agents to the roles.
4. If all roles can be assigned, return the organizational design, else return failure, triggering the search to continue with other candidate decompositions.

The design search begins with the candidate partial plan corresponding to the organization's top-level goal(s), and terminates as soon as a completely instantiated organizational design has been found. Because the candidate list is kept sorted using a heuristic, the first successful returned design is adopted; even though a better design might be possible, the costs of an exhaustive search for it argues for heuristic termination.

The above algorithmic outline ignores a variety of details about how knowledge about decompositions and agent capabilities are collected, stored, and retrieved, and about complications that arise from, for example, assigning a single role to multiple agents that then themselves need to coordinate. Yet, it is fundamentally like a planning algorithm. As has been pointed out elsewhere, the distinction between an organizational role and an abstract plan step is blurry [21]. In both cases, the agent is expected to dynamically elaborate the specification given its current circumstances, with the expectation that any suitable elaboration will fulfill the responsibilities to the rest of the organization/plan.

3.2.2 Organizational Execution and Functionally-Accurate Cooperation

Agents working in a distributed sensor network lack global awareness of the problem, and thus cooperate by forming local interpretations based on their local sensor data and the tentative partial interpretations received from others, and then sharing their own tentative partial interpretations. As a result, these agents need to cooperate to solve their subtasks, and might formulate tentative results along the way that turn out to be unnecessary. This style of collective problem solving has been termed functionally accurate (it gets the answer eventually, but with possibly many false starts) and cooperative (it requires iterative exchange) [40].

Functionally-accurate cooperation has been used extensively in distributed problem solving for tasks such as interpretation and design, where agents only discover the details of how their subproblem results interrelate through tentative formulation and iterative exchange. For this method to work well, participating agents need to treat the partial results they have formulated and received as tentative, and therefore might have to entertain and contrast several competing partial hypotheses at once. A variety of agent architectures can support this need; in particular, blackboard architectures [15] have often been employed as semi-structured repositories for storing multiple competing hypotheses.

Exchanging tentative partial solutions can impact completeness, precision, and confidence. When agents can synthesize partial solutions into larger (possibly still partial) solutions, more of the overall problem is covered by the solution. When an agent uses a result from another to refine its own solutions, precision is increased. And when an agent combines confidence measures of two (corroborating or competing) partial solutions, the confidence it has in the solutions changes. In general, most distributed problem-solving systems assume similar representations of partial solutions (and their certainty measures), which makes combining them straightforward, although some researchers have considered challenges in crossing between representations, such as combining different uncertainty measurements [68].

In functionally-accurate cooperation, the iterative exchange of partial results is expected to lead, eventually, to some agent having enough information to keep moving the overall problem solving forward. Given enough information exchange, therefore, the overall problem will be solved. Of course, without being tempered by some control decisions, this style of cooperative problem solving could incur dramatic amounts of communication overhead and wasted computation. For example, if agents share too many results, a phenomenon called *distraction* can arise: it turns out that they can begin to all gravitate toward doing the same problem-solving actions (synthesizing the same partial results into more complete solutions). That is, they all begin exploring the same part of the search space. For this reason, limiting communication is usually a good idea, as is giving

agents some degree of skepticism in how they assimilate and react to information from others. We address these issues next.

Organizational structuring can provide the basis for making good decisions about where agents should direct their attention and apply their communication resources. The organization defines control and communication protocols between agents by providing messaging templates and patterns to agents that trigger appropriate information exchange. As a simple example, the organization can provide an agent with simple communication rules, such that if the agent creates a local hypothesis that matches the rule pattern (e.g., characterizes an event near a boundary with other agents), then the agent should send that hypothesis to the specified agents. Similarly, if an agent receives a hypothesis from another, the organizational structure can dictate the degree to which it should believe and act on (versus being skeptical about) the hypothesis.

Organization structures thus provide static guidelines about who is generally interested in what results. But this ignores timing issues. When deciding whether to send a result, an agent really wants to know whether the potential recipient is likely to be interested in the result now (or soon). Sending a result that is potentially useful but that turns out not to be at best clutters up the memory of the recipient, and at worst can distract the recipient away from the useful work that it otherwise would have done. On the other hand, refraining from sending a result for fear of these negative consequences can lead to delays in the pursuit of worthwhile results and even to the failure of the system to converge on reasonable solutions at all because some links in the solution chain were broken.

When cluttering memory is not terrible and when distracting garden paths are short, then the communication strategy can simply be to send all partial results. On the other hand, when it is likely that an exchange of a partial result will distract a subset of agents into redundant exploration of a part of the solution space, it is better to refrain, and only send a partial result when the agent that generated it has completed everything that it can do with it locally. For example, in a distributed theorem-proving problem, an agent might work forward through a number of resolutions toward the sentence to prove, and might transmit the final resolvent that it has formed when it could progress no further.

Between the extremes of sending everything and sending only locally-complete results are a variety of gradations [22], including sending a small partial result early on (to potentially spur the recipient into pursuing useful related results earlier). For example, in a distributed vehicle monitoring problem, sensing agents in neighboring regions need their maps to agree on how vehicles move from one region to the other. Rather than waiting until it forms its own local map before telling its neighbor, an agent can send a preliminary piece of its map near the boundary early on, to stimulate its neighbor into forming a complementary map

(or determining that no such map is possible and that the first agent is pursuing a dubious interpretation path).

So far, we have concentrated on how agents decide when and with whom to voluntarily share results. But the decision could clearly be reversed: agents could only send results when requested. When the space of results formed is large and only a few are really needed by others, then sending requests (or more generally, goals) to others makes more sense. This strategy has been explored in distributed vehicle monitoring [17], as well as in distributed theorem proving [25, 42].

It is also important to consider the delays in iterative exchange compared to a blind inundation of information. A request followed by a reply incurs two communication delays, compared to the voluntary sharing of an unrequested result. But sharing too many unrequested results can introduce substantial overhead. Clearly, there is a trade-off between reducing information exchanged by iterative messaging versus reducing delay in having the needed information reach its destination by sending many messages at the same time. Sen, for example, has looked at this in the context of distributed meeting scheduling [52]. Our experience as human meeting schedulers tells us that finding a meeting time could involve a series of proposals of specific times until one is acceptable, or it could involve having the participants send all of their available times at the outset. Most typically, however, practical considerations leave us somewhere between these extremes, sending several well-chosen options at each iteration.

Finally, the communication strategies outlined have assumed that messages are assured of getting through. If messages get lost, then results (or requests for results) will not get through. But since agents do not necessarily expect messages from each other, a potential recipient will be unable to determine whether or not messages have been lost. One solution to this is to require that messages be acknowledged, and that an agent sending a message will periodically repeat the message (sometimes called “murmuring”) until it gets an acknowledgment [41]. Or, a less obtrusive but more uncertain method is for the sending agent to predict how the message will affect the recipient, and to assume the message made it through when the predicted change of behavior is observed.

3.3 The Contract-Net Protocol and Role Assignment

The third and final category of techniques we discuss where coordination is done prior to making local planning and control decisions is the use of predefined protocols. An entire chapter of this book is dedicated to communication and protocols, and so we will not go into all of the gory details. But the important point from the perspective of this chapter is that protocols typically amount to predefined multiagent plan templates. These plan templates are intended to bring about a new joint state of the world, where the new state can include agents now knowing

things that they previously did not know, forging relationships and dependencies that mutually benefit them, making commitments that allow each to pursue some tasks with confidence that others will pursue related tasks, etc.

Formulating protocols is thus akin to formulating social laws or organizational structures: given properties of desired states of the world, predefine patterns of actions that if jointly followed will bring them about. In fact, the creation of choreographed service-oriented computing frameworks can treat the problem of composing and sequencing services as a planning problem (Chapter 3).

For the remainder of this section, however, we focus not on where protocols come from, but rather on how they can serve to control the interactions of agents toward a particular outcome. We will illustrate the ideas by examining one of the very first multiagent protocols, the contract-net protocol, and one of its first applications, which is to establish a distributed sensor network [18]. In distributed sensor network establishment (DSNE), roles (areas of sensing responsibility, responsibilities for integrating partial interpretations into more complete ones) need to be assigned to agents, where the population of agents might be initially unknown or dynamically changing. Thus, the purpose of the protocol is to exchange information in a structured way to converge on assignments of roles to particular agents.

At the outset, it is assumed that a particular agent is given the task of monitoring a wide geographic area. This agent has expertise in how to perform the overall task, but is incapable of sensing all of the area from its own locality. Therefore, the first step is that an agent recognizes that to perform its task better (or at all) it should enlist the help of other agents. As a consequence, it then needs to create subtasks to offload to other agents. In the DSNE problem, it can use its representation of the structure of the task to identify that it needs sensing done (and sensed data returned) from remote areas. Given this decomposition, it then uses the protocol to match these sensing subtasks with available agents.

The agent announces a request for bids for subtask. The important aspects of the announcement for our purposes here are the eligibility specification, the task abstraction, and the bid specification. (Attributes of message structures are described more fully in Chapter 3.) To be eligible for this task requires that the bidding agent have a sensor position within the required sensing area and that it have the desired sensing capabilities. Agents that meet these requirements can then analyze the task abstraction (what, at an abstract level, is the task being asked of the bidders?) and can determine the degree to which it is willing and able to perform the task. An eligible agent can then bid on the task, where the content of a bid is dictated by the bid specification.

The agent with the task receives back zero or more bids. If it gets no bids, then it can give up, try again (since the population of agents might be changing),

broaden the eligibility requirements to increase the pool of potential bidders, or decompose the task differently to target a different pool of bidders. Even if it gets back bids, it could be that none are acceptable to it, and it is as if it got none back. If one or more is acceptable, then it can award the sensing subtask to one (or possibly several) of the bidding agents. Note that, because the agent with the task has a choice over what it announces and what bids it accepts, and an eligible agent has a choice over whether it wants to bid and what content to put into its bid, no agent is forced to be part of a contract. The agents engage in a rudimentary form of negotiation, and form teams through *mutual selection*.

4 Local Planning Prior to Coordination

In some problem domains, predicting and pre-arranging the resolution of all possible interactions can be difficult and costly. For example, consider an environment where agents might pursue a wide variety of goals largely independently, but where aspects of the environment are shared such that how one agent affects the environment can impact how (and even whether) another agent can achieve its goals. Anticipating and planning for every possible interaction might be overkill. Instead, coordination should depend on the actual plans, and hence emergent interactions, of the agents in the current circumstances.

This viewpoint is appealing from the perspective of “divide and conquer” problem solving. The notion is to divide the problem up such that agents initially treat their own local problems as being independent, and thus each agent can formulate its own plan concurrently with the planning of other agents. After formulating their separate plans, then, the agents need to coordinate their plans to resolve their unintended interactions. We will refer to the problem of resolving interactions between separately-formed agent plans as the *multiagent plan coordination problem (MPCP)*.

As has been noted by a variety of people (dating back to Conry et al. [13], and recently by Nassim et al. [46]), this view of multiagent planning is compatible with a distributed constraint satisfaction formulation, where the variables are the agents’ plans, and the constraints enforce that the plans dovetail together suitably. In distributed constraint satisfaction approaches [67], each agent is responsible for some set of variables, where each variable has an associated (typically finite) domain of values, and whose value assignment can be constrained depending on the assignments of other variables (possibly belonging to other agents). Traditionally, distributed constraint satisfaction algorithms involve asynchronous exchanges of tentative value assignments to (some of) the variables, and then information about constraint violations that trigger one or more agents to revise their tentative assignments. Parallel search can speed up finding a satisfying assignment, but care

must be taken to ensure the process is complete and terminates.

The MPCP differs from typical distributed constraint satisfaction problems in several important ways. First, the domain of possible “values” for an agent’s plan “variable” is usually large (even infinite), and expensive to construct. Hence, there is a desire to generate as few elements of the variable domains as possible before converging on a joint plan. Second, the “constraints” between agents’ variables are complex. It is non-trivial to assess whether the plans of two agents are compatible and will lead to some desired outcome state (or avoid an undesired state) if executed asynchronously.

Hence, the MPCP is generally solved in a sequential manner, possibly with backtracking, without assurances of finding an optimal joint plan. The basic outline of the algorithm is:

1. Each agent builds its local plan as if it were alone in the environment.
2. Agents directly, or through a more centralized intermediary, identify potential problems that can arise during joint execution.
3. For problems that can arise, agents inject additional constraints into their plans (typically, over the timing of their actions relative to each other) to prevent such problems.
4. If all problems are prevented, then the agents are done. Otherwise, if some problems cannot be prevented, then one or more agents develops an alternative local plan, and the coordination problem repeats with the new portfolio of agent plans.

4.1 State-Space Techniques

The preceding algorithm sketch obviously leaves underspecified exactly how agents identify and rectify potential problems. One approach for doing so, reminiscent of GRAPHPLAN concepts [50, section 11.4], is to forward simulate the execution of agents’ plans, step by step, and to detect whether an inconsistent state of the world arises, such as where a condition is simultaneously both established and undone, or when a single resource is assumed to be claimed by more than one agent at the same time. When such a state arises, the combination of actions that led to the inconsistency is analyzed, and one or more of those actions are prohibited to prevent the inconsistency. In essence, the agent(s) performing those action(s) are required to pause in the execution of their plan(s) at this step. Then, from the new resulting consistent state, again all agents (including those who paused) propose their next actions, and the process continues. If one or more agents is blocked from ever completing its plan, then the process can backtrack to

either pause different combinations of actions, or even to request some agents to form different local plans. It is also possible for some agent actions to be entirely skipped over, if serendipitously some other agents had established the conditions that those actions were intended to establish.

Ephrati and Rosenschein [23, 24] have formulated a more robust version of this approach. Their *plan combination search* approach to multiagent plan coordination begins with each agent constructing a set of possible local plans, rather than just one specific local plan which might be incompatible with local plans of others. During the search, the agents' sets of plans gets refined to converge on a nearly optimal combination. The search process avoids commitment to sequences of actions by specifying sets of *propositions* that hold as a result of action sequences instead of fully grounded states. The agents search by proposing, given a particular set of propositions about the world, the changes to that set that they each can make with a single action from their plans. These are all considered so as to generate candidate next sets of propositions about the world, and these candidates are ranked using an A* heuristic. Specifically, the cost of reaching the candidate set of propositions is summed with the total of each agent's estimated cost to achieve its goal based on its plans. The best candidate is chosen and the process repeats, until no agent wants to propose any changes (each has accomplished its goal).

Ephrati and Rosenschein illustrate this approach using a simple problem of agents cooperatively constructing an arch by separately planning the construction of each upright as well as the lintel [23]. Note that, depending on the more global movement of the plan, an agent will be narrowing down the plan it expects to use to accomplish its own private goals. Thus, agents are simultaneously searching for which local plan to use as well as for synchronization constraints on their actions, since in many cases the optimal step forward in the set of achieved propositions might omit the possible contributions of an agent, meaning that the agent should not perform an action at that time.

4.2 Plan-Space Techniques

The plan-space formulation of the MPCP below follows from Russell and Norvig's presentation of partial-order planning [50]. We begin with a brief refresher on single-agent partial-order planning, and then provide definitions and algorithms that extend it to the multiagent case.

4.2.1 Single-Agent Plans

A single-agent plan is a total or partial ordering of steps that will advance an agent from its initial state I to a state that satisfies its goal conditions G . A plan

step is a fully grounded (or variable free) instance of an operator from the agent's set of operators. An operator a in this representation has a set of preconditions ($pre(a)$) and postconditions ($post(a)$), where each condition $c \in pre(a) \cup post(a)$ is a positive or negative (negated) first-order literal. The set $pre(a)$ represents the set of preconditions that must hold for the agent to carry out operator a , and the set $post(a)$ represents the postconditions, or effects, of executing the operator on an agent's world state.

A standard formulation of a single-agent plan is a partial-order, causal-link (POCL) plan. POCL plans capture temporal and causal relations between steps in the partial-order plan. The definition of a POCL plan here is based on Bäckström [2], though it follows common conventions in the POCL planning community [63] to include special steps representing the initial and goal states of the plan.

Definition 11.1 A **POCL plan** is a tuple $P = \langle S, \prec_T, \prec_C \rangle$ where S is a set of plan steps (operator instances), \prec_T and \prec_C are (respectively) the temporal and causal partial orders on S , where $e \in \prec_T$ is a tuple $\langle s_i, s_j \rangle$ with $s_i, s_j \in S$, and $e \in \prec_C$ is a tuple $\langle s_i, s_j, c \rangle$ with $s_i, s_j \in S$ and where c is a condition. A POCL plan models the agent's initial state using an *init* step, $init \in S$, and the agent's goal using a *goal* step, $goal \in S$, where $post(init) = I$ (the initial state conditions), and $pre(goal) = G$ (the goal conditions).

Elements of \prec_T are commonly called **ordering constraints** on the steps in the plan. A partial-order plan has the following properties:

- \prec_T is *irreflexive* (if $s_i \prec_T s_j$ then $s_j \not\prec_T s_i$).
- \prec_T is *transitive* (if $s_i \prec_T s_j$ and $s_j \prec_T s_k$ then $s_i \prec_T s_k$).

Elements of \prec_C are the causal links, representing causal relations between steps, where causal link $\langle s_i, s_j, c \rangle$ represents the fact that step s_i achieves condition c for step s_j . The presence of a causal link in a plan implies the presence of an ordering constraint.

The *single-agent planning problem* can be seen as the problem of transforming an inconsistent POCL plan into a consistent POCL plan. This is done by searching through the space of possible POCL plans, identifying the consistency flaws in the current POCL plan under consideration, and iteratively repairing them to produce a state representing a consistent POCL plan. Flaws include causal-link conflicts and open preconditions. The presence of a causal-link conflict in a plan indicates that, for some causal link $\langle s_i, s_j, c \rangle$, there exist executions (linearizations) of the partial-order plan where a step $s_k \in S$ negates condition c after it is produced by s_i but before it can be utilized by step s_j . Given a conflict between a step s_k and a causal link $\langle s_i, s_j, c \rangle$, the standard method to resolve it is to add either $\langle s_k, s_i \rangle$ or

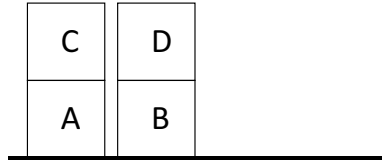


Figure 11.1: Initial state for simple blocks world problem.

$\langle s_j, s_k \rangle$ to \prec_T . That is, order the threatening step either before or after the link. An open precondition c of a plan step $s_j \in S$ can be satisfied by adding a causal link $\langle s_i, s_j, c \rangle$ where step $s_i \in S$ establishes the needed condition (and s_i is not ordered after s_j). If this requires that a new step s_i is added to S , then its preconditions become new open preconditions in the plan.

As a simple example, consider the blocks world situation portrayed in Figure 11.1. Say Agent1 has a goal of achieving a state where block A is on block B. Using the POCL algorithm, it creates the plan shown in Figure 11.2, where actions and their parameters are given in the squares, their preconditions (postconditions) are given to the left (right) of the square, causal links are the narrow black arrows, and ordering constraints are the wide gray arrows. Notice that the plan is partially ordered, in that before block A can be stacked on B, both blocks must be cleared, but they can be cleared in either order. Finally, in Figure 11.3 is a plan for Agent2 to stack block B on C. Neither agent cares where block D ends up.

4.2.2 Multiagent Plans

To extend the notation and definitions of POCL plans to the multiagent case, we first address the issue of action concurrency. A single-agent usually takes only one action at a time, and thus a POCL plan such as in Figure 11.2 will be linearized before or during execution. If an agent can execute multiple actions in parallel, unordered actions that are eligible for execution are assumed to be executed concurrently. The POCL plan representation cannot express that *some* unordered steps to be executed in parallel but not others. To make this distinction, we extend from Bäckström [2] the idea of a *parallel* plan, to define a *parallel POCL plan*.

Definition 11.2 A *parallel POCL plan* is a tuple $P = \langle S, \prec_T, \prec_C, \#, = \rangle$ where $\langle S, \prec_T, \prec_C \rangle$ is the embedded POCL plan, and “#” and “=” are symmetric non-concurrency and concurrency relations over the steps in S , respectively.

The relation $\langle s_i, s_j \rangle \in =$ means that s_i and s_j are required to be executed simultaneously. For example, if a plan has multiple goal steps and is intended to reach a state where all goals are satisfied simultaneously, then all pairs of goal steps would be elements of $=$. The relation $\langle s_i, s_j \rangle \in \#$ is equivalent to the statement

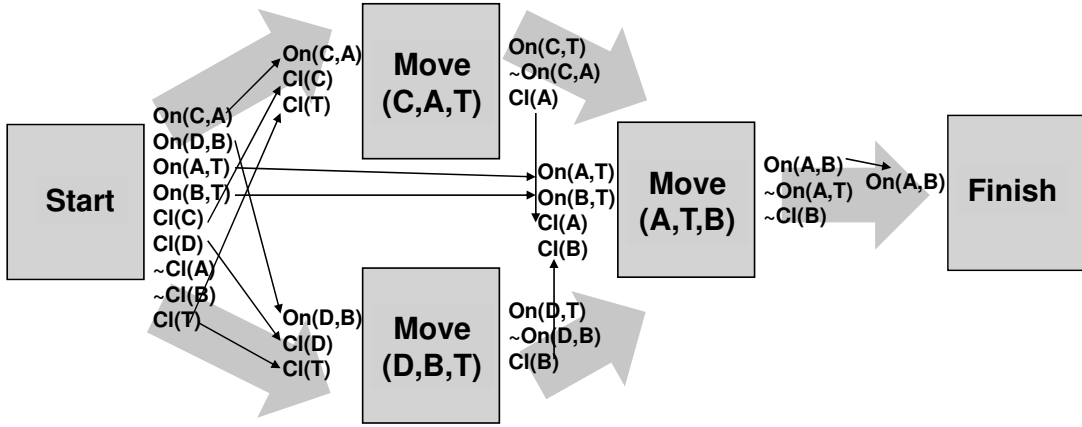


Figure 11.2: Single POCL plan for stacking block A on block B.

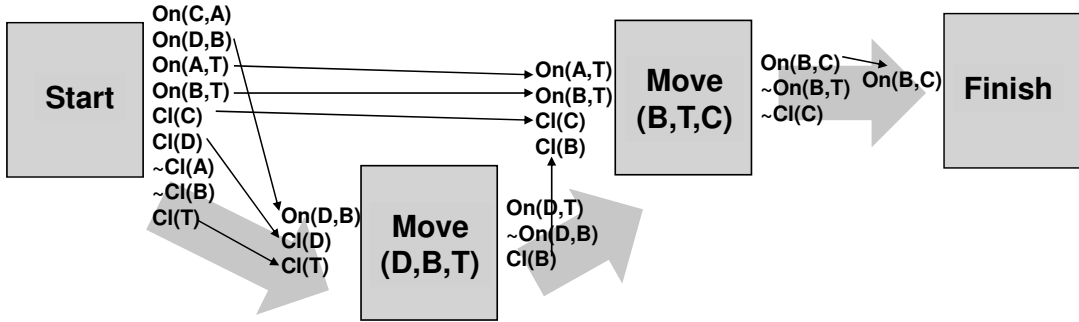


Figure 11.3: Single POCL plan for stacking block B on block C.

$(s_j \prec_T s_i) \vee (s_i \prec_T s_j)$. The $\#$ relation is needed because we make the assumption that parallel plans obey the *post-exclusion principle* [2], which states that actions cannot take place simultaneously when their postconditions are not consistent. The $\#$ and $=$ are disjoint sets, as two steps cannot be required to be concurrent and non-concurrent.

Given this definition of a parallel plan, it is clear that a partial-order (POCL) plan P is a specialization of a parallel (POCL) plan P^* in which either all pairs of steps in P^* are in $\#$ (if it is assumed an agent can do only one action at a time) or none of them are (if all unordered actions are assumed to be concurrently executable). Likewise, a POCL plan implicitly requires that $=$ be empty (unless a

single agent can execute multiple steps concurrently).

Because of the post-exclusion principle, parallel plans have an additional source of plan flaws:

Definition 11.3 A *parallel-step conflict* exists in a parallel plan when there are steps s_j and s_i where $\text{post}(s_i)$ is inconsistent with $\text{post}(s_j)$, $s_j \not\prec_T s_i$, $s_i \not\prec_T s_j$ and $\langle s_i, s_j \rangle \notin \#$.

However, unlike open conditions and causal-link conflicts, parallel-step conflicts can always be resolved, no matter what other flaw resolution choices are made. Recall that to repair a parallel-step conflict between steps s_i and s_j , we need only ensure that the steps are non-concurrent, either by adding $s_i \prec_T s_j$ or $s_j \prec_T s_i$ to the plan. Given an acyclic plan P , there will always be at least one way of ordering every pair of steps in the plan such that the plan P remains acyclic. This can be trivially shown by considering the four possible existing orderings of any pair of steps s_i and s_j in plan P . First, s_i and s_j could be unordered. In this case, we can add either $s_i \prec_T s_j$ or $s_j \prec_T s_i$ to the plan without introducing cycles in the network of steps. Second, $s_i \prec_T s_j$ is in the plan, in which case the parallel-step conflict has already been resolved. The same is true when $s_j \prec_T s_i$ is in the plan. Finally, $s_i \prec_T s_j$ and $s_j \prec_T s_i$ could both hold in the plan, but in this case the plan already has a cycle, and so repairing the parallel-step conflict becomes moot.

The parallel plan model captures the idea of concurrency, but it is not rich enough to describe the characteristics of a multiagent plan, in which we also need to represent the agents involved, and to which actions they are assigned. To do so, we extend the definition of a parallel plan to a multiagent parallel POCL plan.

Definition 11.4 A *multiagent parallel POCL plan* is a tuple $M = \langle A, S, \prec_T, \prec_C, \#, =, X \rangle$ where $\langle S, \prec_T, \prec_C, \#, = \rangle$ is the embedded parallel POCL plan, A is the set of agents, and X is a set of tuples of form $\langle s, a \rangle$, representing that the agent $a \in A$ is assigned to execute step s . A multiagent plan models the agents' initial states using *init* steps, $\text{init}_i \in S$, and the goals of the agents using a set of goal steps, $\text{goal}_i \in S$, where the preconditions of the goal steps represent the conjunctive goal that the plan achieves, and the postconditions of the *init* steps represent features of the agents' initial states before any of them take any actions.

Figure 11.4 shows the (inconsistent!) multiagent parallel POCL plan composed of the two agents' individual plans from Figures 11.2 and 11.3. The dashed rectangles around the *init* steps and *goal* steps indicate that these pairs of steps are in the $=$ relation. That is, all *init* steps happen simultaneously (there is a single initial state whose conditions are the union of the *init* step postconditions), and all *goal* steps happen simultaneously (there is a final goal state for the multiagent system whose conditions are the union of the *goal* step preconditions).

Now, just as in the single-agent case, the *multiagent planning problem* can be solved by making an inconsistent multiagent parallel POCL plan consistent, where each plan step is assigned to an agent capable of executing the step. More formally, the multiagent plan coordination problem is the problem, given a set of agents A and the set of their associated POCL plans \mathbf{P} , of finding a consistent and optimal multiagent parallel POCL plan M *composed entirely of steps drawn from \mathbf{P}* (in which agents are only assigned to steps that originate from their own individual plans) that results in the establishment of all agents' goals, given the collective initial state of the agents. The MPCP can thus be seen as a restricted form of the more general multiagent planning problem in which new actions are not allowed to be added to any agent's plan.

This definition of the MPCP imposes a set of restrictions on the kinds of multiagent plan coordination problems that can be represented. Because an agent can only be assigned steps that originated in its individual plan, this definition does not model coordination problems where agents would have to reallocate their activities. Further, because only individually-planned steps are considered, the definition does not capture problems where additional action choices are available if agents work together; that is, an agent when planning individually will not consider an action that requires participation of one or more other agents. Finally, in keeping within the "classical" planning realm, the definition inherits its associated limitations, such as assuming a closed world with deterministic actions where the initial state is fully observable.

For any given multiagent parallel plan, there may be many possible *consistent* plans one could create by repairing the various plan flaws. However, not all *consistent* plans will be *optimal*. Based on the assumptions outlined previously concerning the nature of the multiagent plan coordination problem (namely, that the final plan must be assembled solely from the original agents' plans), an optimal multiagent plan will be one that minimizes the total cost of the multiagent plan:

Definition 11.5 *Total step cost measures the cost of a multiagent parallel plan by the aggregate costs of the steps in the plan.*

This simple, global optimality definition is not the only one that could be used for the MPCP, but correlates to the most widely-adopted single-agent optimality criterion. Other relevant definitions include ones minimizing the time the agents take to execute their plans (exploiting parallelism), maximizing the load balance of the activities of the agents, or some weighted combination of various factors.

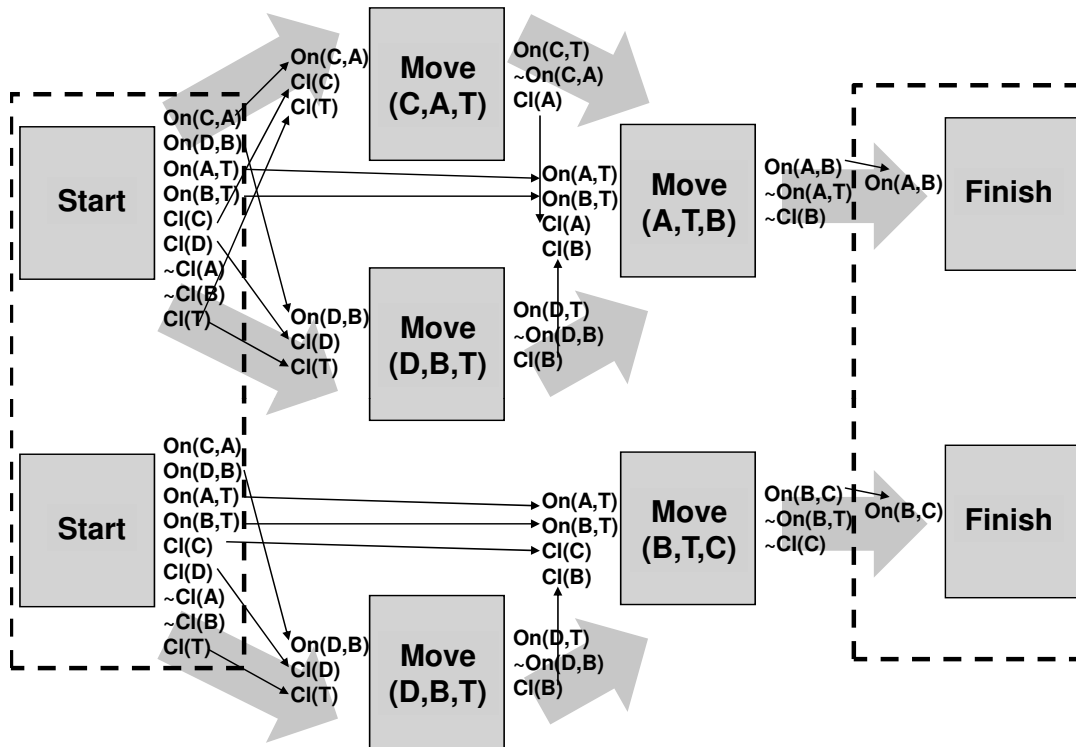


Figure 11.4: Initial (inconsistent) multiagent parallel POCL plan.

4.2.3 Multiagent Plan Coordination by Plan Modification

As illustrated in Figure 11.4, an initial multiagent parallel plan can simply be the union of the individual plan structures of the agents, and thus might contain flaws due to potential interactions between the individual plans. The initial multiagent plan can then be incrementally modified as needed (by both asserting new coordination decisions and retracting the individual planning decisions of the agents) to resolve the flaws. We call this approach coordination by *plan modification*.

From the initial (as yet uncoordinated) multiagent plan, plan coordination takes place by repairing any flaws due to interactions between the plans. The types of flaws are exactly the same as in parallel POCL plans: open preconditions, causal-link threats, and parallel-step flaws. Assuming each of the individual plans are consistent, there should be no open precondition flaws to resolve, at least to begin with. Causal link threats within each agent's plan should not exist, but new threats arise when an action in one agent's plan threatens a link in another

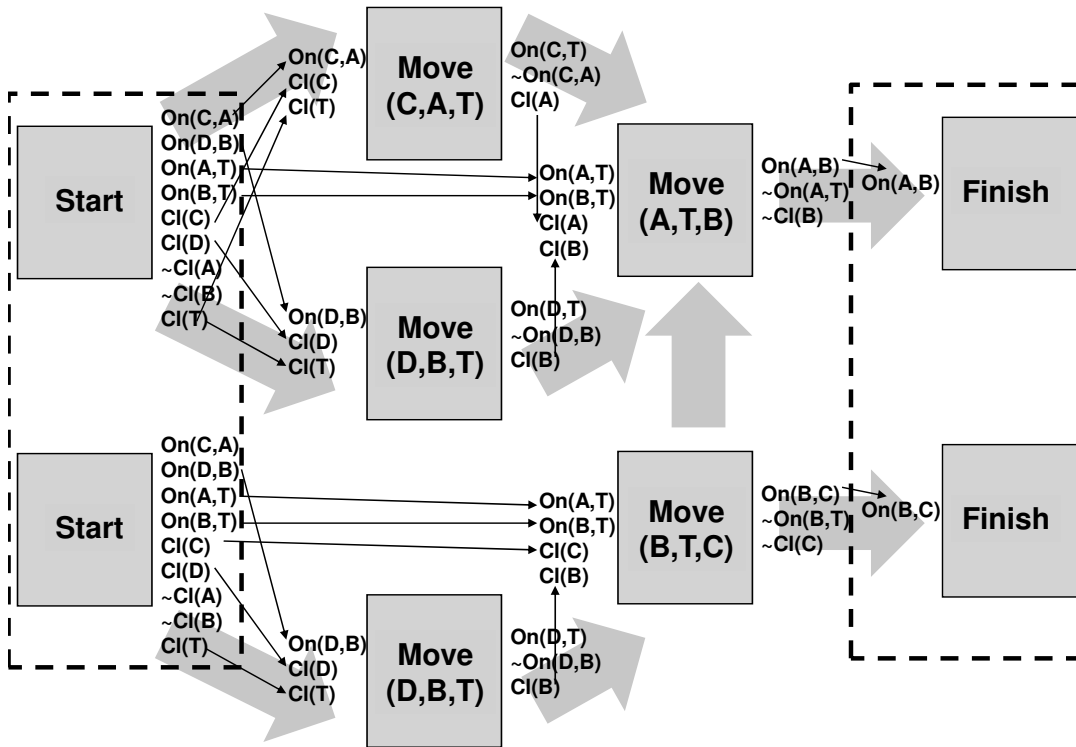


Figure 11.5: Ordering constraint added to resolve causal-link threat.

agent's plan. In the running example (Figure 11.4), Agent1's $Move(A,T,B)$ step results in block B no longer being clear ($\neg Cl(B)$), which threatens the causal link between Agent2's $Move(D,B,T)$ and $Move(B,T,C)$ steps. The flaw can be resolved by adding to the ordering constraints that $Move(A,T,B)$ come after $Move(B,T,C)$, as shown in Figure 11.5. Similarly, as before, parallel-step flaws can also be handled by adding in ordering constraints, though the running example has no such flaws.

The multiagent parallel POCL plan in Figure 11.5 could still be considered flawed, however, because Agent1 and Agent2 both are planning on moving block D from block B to the table. In the best case, one of these agents would execute the action, and then the other before attempting its action would recognize that it can simply skip the action because the desired effects are done. However, some plan execution systems would treat the situation as a deviation from expectations and attempt to repair the plan by inserting actions to (re)establish the conditions that the step expected. In other words, the second agent to execute the $Move(D,B,T)$

action might put block D back onto block B just so that it can move it off. This is obviously wasteful of time and energy. And, even worse, if the two agents were to attempt their $\text{Move}(D,B,T)$ actions at about the same time, their effectors (grippers) might collide, and the agents might disable themselves!

The MPCP thus introduces a new type of flaw that affects the correctness, or at least the optimality, of the multiagent plan. Specifically, a step from some agent's plan could be *redundant* given the presence of steps in others' plans. Note that redundancy does not require that the agents seek the same effect. For example, if Agent1 had included action $\text{Move}(D,B,T)$ to achieve $\text{On}(D,T)$, while Agent2 had planned that action to achieve $\text{Cl}(B)$, the action taken by one agent has the side effect of satisfying the other. In such cases, redundant steps may be able to be removed without introducing new open precondition flaws.

Definition 11.6 *A plan step s is **redundant** in a multiagent parallel POCL plan M with steps S when there exists a set of replacing steps R , where $R \subseteq S$, such that for each causal link of form $\langle s, s'', c \rangle$, it is also the case that $\exists s' \in R$ s.t. $c \in \text{post}(s')$.*

Redundancies can be discovered by altering the causal structure of the multiagent plan, by retracting some causal-link instantiation decisions and then asserting others to replace them (so as to prevent the introduction of open precondition flaws). To perform an adjustment of a single causal-link $l = \langle s_i, s_j, c \rangle$, we simply identify another step s_k that also achieves condition c , and then change l such that $l = \langle s_k, s_j, c \rangle$. If such an adjustment leaves s_i with no outgoing causal links, then s_i can be removed from the plan.

Note that the removal of a single redundant step may require many causal links to be adjusted as each outgoing causal link of a redundant step must be adjusted so that the redundant step is no longer causally necessary in the plan. Thus, a set of steps can collectively replace a single redundant step in a plan. Also note that removing plan steps provides an alternative way to resolve causal-link and parallel-step flaws: rather than adjust orderings for a step that threatens a link or introduces a potential inconsistency with another step, the step can be removed and the flaws go away. Finally, note that just because a redundancy exists, it does not mean that it can necessarily be exploited. As in a single-agent plan, the same action might need to be taken multiple times because its effects are necessarily undone by intervening actions. (The Towers of Hanoi puzzle is a familiar example of this.) During plan modification, if removal of such a step is attempted, the process of link adjustment would result in causal link threats that have no valid resolutions.

The plan modification algorithm (PMA) is shown in Algorithm 11.1. The PMA uses a best-first search in order to find the optimal solution. The search algorithm begins by initializing the search queue with the starting multiagent par-

Input : an (inconsistent) multiagent parallel plan
Output: an optimal and consistent multiagent parallel plan or null plan

```

1 Initialize Solution to null;
2 Add input plan to search queue;
3 while queue not empty do
4   Select and remove multiagent plan M from queue;
5   if M not bounded by Solution then
6     if (M passes Solution Test) and (steps in M < steps in Solution) then
7       Solution = M;
8     end
9     Select and adjust a non-flagged causal-link in M;
10    For each refinement, remove unnecessary steps in plan;
11    Enqueue all plan refinements in search queue;
12  end
13 end
14 repair parallel-step conflicts in Solution;
15 return Solution;

```

Algorithm 11.1: Multiagent plan coordination by plan modification.

allel plan, and by initializing the current best solution, *Solution*, to *null*. Then, while the queue is not empty, it selects a multiagent plan *M* with the lowest total step cost from the queue.

A *bounding test* is applied to *M* to determine whether it is possible for *M* to have a lower total step cost than the best *Solution* found so far (if any). A lower bound on total step cost is computed for plan *M* by working backwards from the goal steps to find all plan steps that contribute *flagged* causal links (as will shortly be explained) to achieving the goal. If the lower bound of *M* is below the cost of *Solution*, the algorithm proceeds.

PMA next applies a *SolutionTest* to *M*, to derive a consistent solution from *M*, where any flaws in *M* other than step redundancy flaws are iteratively resolved by adding ordering constraints. The *SolutionTest* thus conducts a depth-first search through the space of flaw resolutions to find a consistent solution, heuristically ordering the search to prioritize flaws for which there are fewer alternative resolutions, which is a minimum remaining values (MRV) heuristic [50]. If the consistent solution from *M* has a lower total step cost than *Solution*, it replaces *Solution*.

The PMA algorithm then selects and adjusts a non-flagged causal link in *M*. In Figure 11.5, consider the causal link into condition Cl(B) for Agent1's Move(A,T,B) step. That causal link could originate from either Agent1's Move(D,T,B) step (as it does in the figure) or from Agent2's Move(D,T,B) step. The PMA makes copies of *M* that differ only in this refinement of the source of

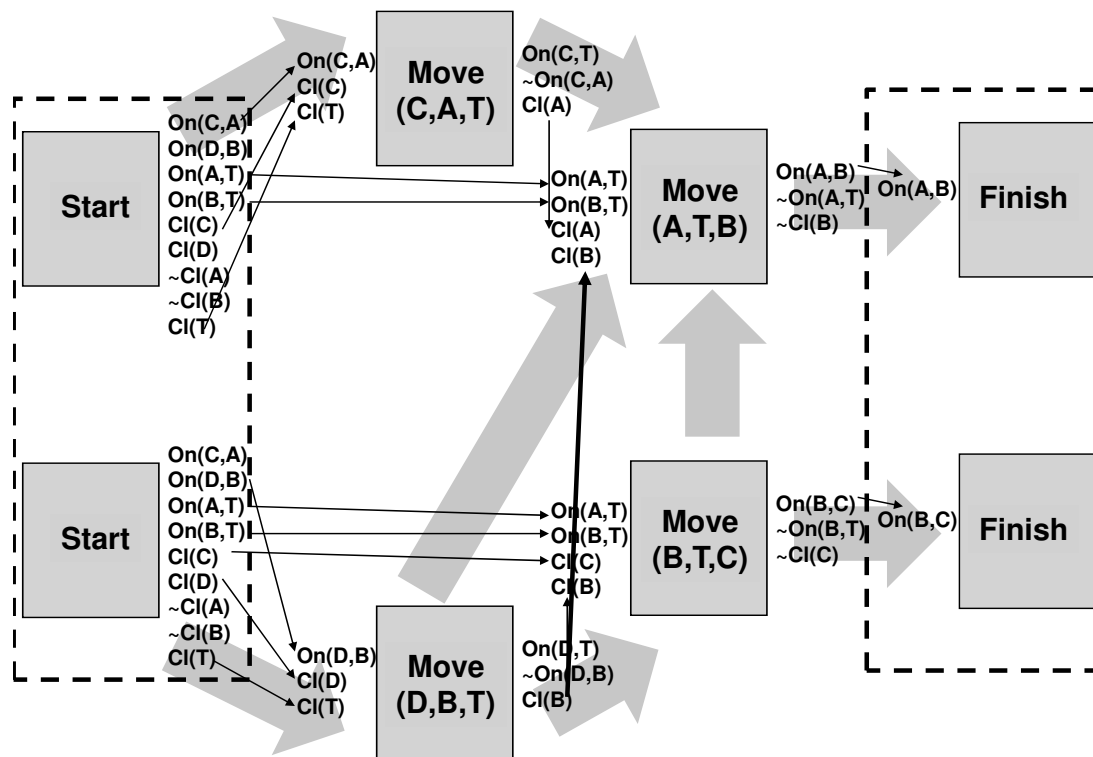


Figure 11.6: Solution multiagent parallel POCL plan.

the causal link, and in each “flags” the causal link so that it is not branched on again, because each of the refinements is added to the queue and might later be further modified. If in a refinement of M the redirection of causal links results in a plan step having no outgoing causal links, then that plan step is removed. When that refinement is enqueued, it will move forward in the queue since its total step cost will be lower.

The process of dequeuing, testing, and refining continues until the queue is empty, at which point the current value of *Solution* represents the lowest cost multiagent parallel plan derivable from the input plans. PMA then resolves any parallel-step conflicts in *Solution* (which, as was previously noted, must be resolvable), and returns *Solution*. In our simple example, the input from Figure 11.4 will lead to two possible solutions, one of which is shown in Figure 11.6 and the other where the Move(D,B,T) action is instead done by Agent1. These both have the same number of steps. If the total step cost function also considers parallel activity, the solution shown would be preferred.

4.3 Hierarchical Multiagent Plan Coordination

Hierarchical task network (HTN) planning is a method with algorithmic similarities to POCL planning. As per the summary in [50], single-agent HTN planning is a plan-space search method where the process focuses on *refining* abstract plans, rather than fixing flawed plans as in POCL planning. The idea is that the planning agent is endowed with a library of plans at various levels of abstraction, and the planning process involves refining a plan by iteratively replacing abstract steps with sequences of steps that are closer to being operational.

A familiar example of (human) HTN planning is planning a trip to attend a distant meeting. Given the distances involved, I might decide that my plan is to “fly” to the meeting. But I cannot execute such a plan without first refining it into steps of getting to a nearby airport, taking a flight to an airport near the meeting site, and then getting from that airport to the meeting. Each of these steps in turn needs to be refined further, until all the steps have been reduced to executable (primitive) actions. Given the same space of primitive actions, a state-space or POCL planner could in principle find the same plan, but the knowledge encoded in entries of the plan library (e.g., that flying involves going from where you are to an airport, taking a flight, and getting from the resulting airport to your final destination) can greatly streamline the planning process.

The same can be said for streamlining the multiagent planning process. The specification of hierarchical plans for teams of agents is part of a process that has been called *team-oriented programming* [49]. Such plans not only describe how high-level tasks are broken down into subtasks, but also how collaborative activities are broken down into different *roles*, similarly to relationships between agents in organizational structures (Section 3.2). Plan refinements associate with a subtask one or more roles that are responsible for that subtask, and ultimately the refinement process assigns an agent to each role. (Recovery from agent failure thus can be done by simply reassigning a role rather than replanning from scratch.) An agent can further refine an assigned subtask for its role using standard HTN planning. Timing and other relationships between different agents’ tasks are specified in the team-oriented programming framework, but generic mechanisms for enforcing them can be automatically instantiated by a team-oriented programming infrastructure such as STEAM [59].

HTN planning representations have also been exploited as a means for flexibly solving multiagent plan coordination problems (Section 4). The insight is that, while an agent can only execute a plan at the level of primitive actions, multiagent coordination actions can (and often should) be based on more abstract levels of the hierarchy [11]. For example, two robots performing tasks in the rooms of a shared environment might avoid collisions by synchronizing to avoid ever occupying the same room, rather than incurring the expense of reasoning about colli-

1. Initialize the current abstraction level to the most abstract level.
2. Agents exchange descriptions of the plans and goals of interest at the current level.
3. Remove plans with no potential conflicts. If the set is empty, then done; otherwise determine whether to resolve conflicts at the current level or at a deeper level.
4. If conflicts are to be resolved at a deeper level, set the current level to the next deeper level and set the plans/goals of interest to the refinements of the plans with potential conflicts. Go to step 2.
5. If conflicts are to be resolved at this level:
 - (a) Agents form a total order. Top agent is the current superior.
 - (b) Current superior sends down its plan to the others.
 - (c) Other agents change their plans to work properly with those of the current superior. Before confirming with the current superior, an agent also double-checks that its plan changes do not conflict with previous superiors.
 - (d) Once no further changes are needed among the plans of the inferior agents, the current superior becomes a previous superior and the next agent in the total order becomes the superior. Return to step (b). If there is no next agent, then the protocol terminates and the agents have coordinated their plans.

Algorithm 11.2: Hierarchical behavior-space search.

sions at the primitive level. Thus, agents might communicate and coordinate at an abstract planning level. This not only can have computational benefits (fewer combinations of joint steps to reason about), but also can have flexibility benefits at execution time. For instance, in our example of robots in a shared workspace, if robots only coordinate at the level of entering and leaving rooms, then each robot retains flexibility to change its planned movements within a room without needing to renegotiate with the other. However, coordinating at an abstract level tends to lead to less efficient joint plans (e.g., a robot idling waiting for another to exit a room rather than carefully jointly working within the room). Further, to anticipate potential primitive interactions at abstract levels means that agents need to summarize for abstract steps the repercussions of the alternative refinements that might be made [11]. Algorithm 11.2 summarizes a simple algorithm for solving the multiagent plan coordination problem for agents that have hierarchical plans.

5 Decision-Theoretic Multiagent Planning

Decision-theoretic planning is aimed at choosing actions under uncertainty by maximizing the expected value of some performance measure called *utility*. Planning in this case explicitly factors the uncertainty about the outcomes of actions and the state of the domain, aiming to *optimize* utility rather than provably *satisfy* certain goals. For example, a decision-theoretic plan for a space exploration rover could maximize the *scientific return*, measured by the amount or the value of the collected data in a given mission, in the face of uncertainty about the pace of progress of the rover and amount of power left in its battery. When applied to a multiagent system, decision-theoretic planning optimizes simultaneously the local plan as well as coordination decisions. The value associated with each action is based on its impact on the domain, the information it transmits to other agents, and the information it obtains from the domain or other agents. Thus, a single planning process optimizes the comprehensive value, thereby optimizing both domain actions and coordination.

A standard framework to tackle planning under uncertainty is the *Markov decision process* (MDP) [47]. The model represents the domain using a set of states. It is designed for a *single* decision maker whose actions lead to stochastic transitions to new states and a reward that can depend on the action and outcome. The *partially observable MDP* (POMDP) is a generalization of the basic model that accounts for imperfect observations. In a POMDP, the decision maker receives *partial* information about the state of the world after taking each action. In that case, the agent can maintain a belief state (probability distribution over domain states), and must act without knowing the exact state of the world. One of the key observations that made it possible to solve single-agent POMDPs is that any POMDP can be viewed as a *belief-state* MDP – an MDP whose domain states are probability distributions over real-world states. Unfortunately, the same is not true in the multiagent case, making planning substantially more complicated. A range of exact and approximate dynamic programming algorithms have been developed for solving MDPs and POMDPs. These algorithms have been used in many practical applications.

A more general planning problem arises when two or more agents have to coordinate their actions. Imagine for example two space exploration rovers that conduct experiments as part of an overall mission. The value of the data collected by one rover may depend on the experiments performed by the other rover. Planning in this case becomes particularly complicated when each agent receives *different* observations and has different partial knowledge of the overall situation. We refer to such problems as *decentralized control problems*, indicating that all the agents control a single process in a collaborative manner, but must each act in a decentralized manner based on their own observations. Such decentralized multiagent

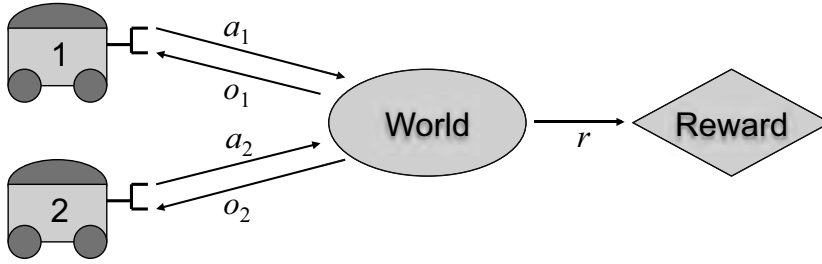


Figure 11.7: Illustration of a two-agent DEC-POMDP.

control problems are ubiquitous. Examples include coordination of mobile robots, load balancing for decentralized queues, target tracking in sensor networks, and monitoring of hazardous weather phenomena. This section describes models for representing such planning problems and algorithms for solving them.

5.1 Models for Decision-Theoretic Multiagent Planning

Natural extensions of MDPs and POMDPs to multiagent settings have been proposed and extensively studied since the late 1990s. We focus in this chapter on decentralized POMDPs (DEC-POMDPs) [7]. Figure 11.7 illustrates a DEC-POMDP with two agents. In each step, each agent takes an action, the joint set of actions causes a stochastic change in the state of the world, and a reward is generated based on the actions and their outcome. Then, each agent receives its own private observation and the cycle repeats.

Definition 11.7 (DEC-POMDP) A decentralized partially observable Markov decision process (DEC-POMDP) is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T \rangle$ where

- I is a finite set of agents indexed $1, \dots, n$.
- S is a finite set of states, with distinguished initial state s_0 or belief state b_0 .
- A_i is a finite set of actions available to agent i and $\vec{A} = \otimes_{i \in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, \dots, a_n \rangle$ denotes a joint action.
- $P : S \times \vec{A} \rightarrow \Delta S$ is a Markovian transition function. $P(s' | s, \vec{a})$ denotes the probability of a transition to state s' after taking joint action \vec{a} in state s .
- Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ is the set of joint observations, where $\vec{o} = \langle o_1, \dots, o_n \rangle$ denotes a joint observation.
- $O : \vec{A} \times S \rightarrow \Delta \vec{\Omega}$ is an observation function. $O(\vec{o} | \vec{a}, s')$ denotes the probability of observing joint observation \vec{o} given that joint action \vec{a} was taken and led to state s' . Here $s' \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.

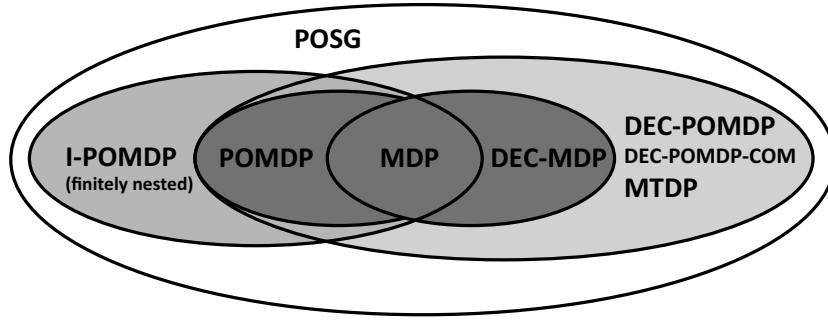


Figure 11.8: Relationship among several models for multiagent planning.

- $R : \vec{A} \times S \rightarrow \Re$ is a reward function. $R(\vec{a}, s')$ denotes the reward obtained after joint action \vec{a} was taken and a state transition to s' occurred.

The goal is to maximize the cumulative (discounted) reward over some finite horizon T or over an infinite horizon. The model includes only one reward function, indicating that the agents operate collaboratively toward one objective. A special case of DEC-POMDP, called DEC-MDP, models situations in which the *combined* observations of all the agents provide perfect information about the underlying world state. DEC-MDPs extend *multiagent MDPs* (MMDPs), where each agent has full knowledge of the underlying world state [9]. Notice that a DEC-POMDP model is equivalent to a single-agent POMDP when $n = 1$.

Communication between agents can be modeled by a DEC-POMDP either implicitly or explicitly. Implicit communication occurs whenever one agent's actions affect another agent's observations. Explicit communication – exchanging messages between agents – can be represented by making the message a component of each observation. Each action in this case can be divided into two parts: a domain action that affects the state of the environment, and a communication action that affects the messages received by other agents.

A model equivalent to DEC-POMDP called *multiagent team decision problem* (MTDP) was introduced in 2002 [48]. DEC-POMDPs and MTDPs are a special case of *partially-observable stochastic games* (POSGs), which allow each agent to have a different objective encoded by a private reward function. Another related model that explicitly represents beliefs about other agents, called *interactive POMDP* (I-POMDP), was introduced in 2005 [28]. The relationships between the different models is illustrated in Figure 11.8, largely based on the analysis in [54].

To illustrate the problems and solution methods, we will use a simple toy problem called the *multiagent tiger problem*. This domain includes two agents, two states, three actions, and two observations, and was introduced by Nair et al. [44]. In this problem, the two agents are initially situated in a room with two

doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (e.g., both open the same door), a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent listens, a small penalty is given and an observation is seen that is a noisy indication of which door the tiger is behind. Once a door is opened, the game resumes from its initial state with the tiger, and treasure's locations randomly reshuffled.

This class of problems has raised several questions about the feasibility of decision-theoretic planning in multiagent settings: Are DEC-POMDPs significantly harder to solve than POMDPs? What features of the problem domain affect the complexity, and how? Is optimal dynamic programming possible? Can dynamic programming be made practical? Can locality of agent interaction be exploited to improve algorithm scalability? Research in recent years has significantly increased the understanding of these issues and produced a solid foundation for multiagent planning in stochastic environments. We describe below some of the key results and lessons learned.

5.1.1 Solution Representation and Evaluation

Solutions for decentralized control problems involve a set of *policies* – one per agent – that determine how each agent should act so as to maximize the overall reward. How can these policies be represented? In the case of finite-horizon problems, one can use *policy trees*. Each policy tree is a decision tree where each node is labeled with an action and branches are labeled with observations. Starting with the root node, at each step, each agent performs the action of the current node and then branches to a subtree based on the observation it receives. Sample optimal policy trees for the multiagent tiger problem with horizons 1–4 are shown in Figure 11.9. The actions are L (listen), OL (open left), and OR (open right). The observations are hl (hear tiger on left) and hr (hear tiger on right). Note that while the value generally grows with the horizon, there are some fluctuations when the added time allows for costly listening actions, but is not sufficient for establishing a reliable belief about the tiger's location.

Figure 11.10 shows a horizon 5 optimal solution (same tree for both agents). As this example illustrates, the size of policy trees grows exponentially with the horizon of the problem, making it hard to keep complete policy trees in large problems. And when the problem has an infinite horizon, policy trees can no longer be used to represent solutions. A common approach in that case is for each agent to summarize what it knows using finite memory and to represent policies using *finite-state controllers*. Each controller state represents an intermediate internal

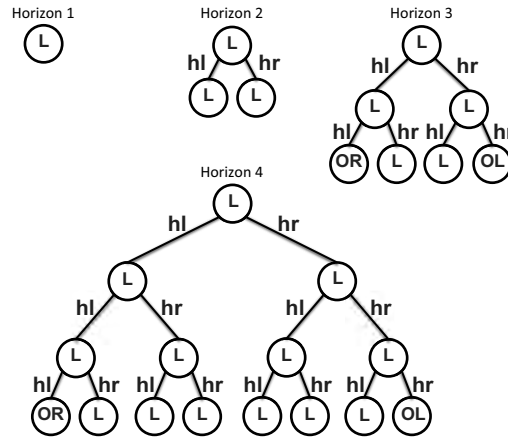


Figure 11.9: Optimal policy trees for the multiagent tiger problem with horizons 1–4. The policy trees of both agents are the same in this case. The expected values as a function of the horizon are: $V_1 = -2$, $V_2 = -4$, $V_3 = 5.19$, $V_4 = 4.80$.

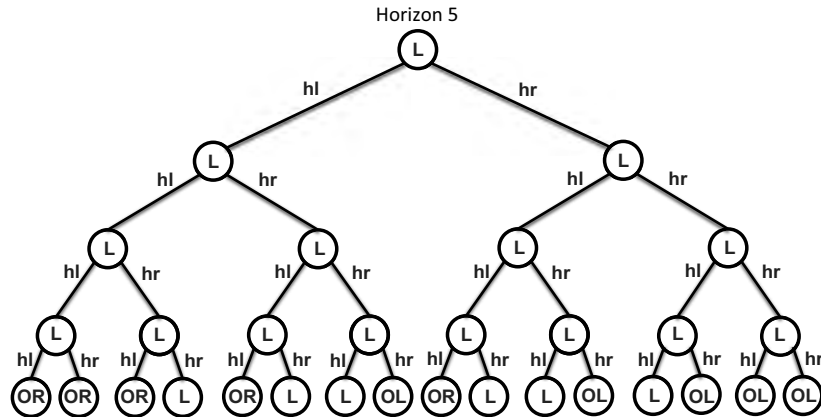


Figure 11.10: Optimal policy tree for the multiagent tiger problem with horizon 5. The expected value in this case is $V_5 = 7.03$.

memory state of the agent. Starting with an initial controller state, at each state an agent chooses an action based on its internal state and then branches to a new internal state based on the observation received. Both the action selection and controller transitions could be deterministic or stochastic; higher value could be obtained using stochastic mappings, but the optimization problem is harder.

Figure 11.11 shows optimal deterministic controllers for the infinite-horizon multiagent tiger problem with a discount factor of 0.9. The large arrow points to the initial state. The figure shows a one-node controller (same controller per

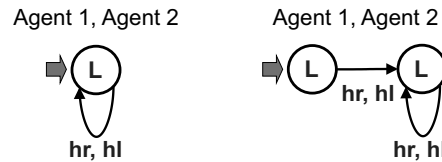


Figure 11.11: Optimal one-node and two-node deterministic controllers for the multiagent tiger problem.

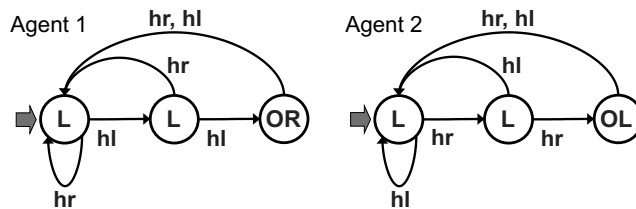


Figure 11.12: Optimal three-node deterministic controllers for multiagent tiger.

agent) with an expected value of -20 , and a two-node controller, which represents the same policy and has the same value. With so little memory, the optimal policy is to listen all the time and not risk opening a door. Figure 11.12 shows an optimal deterministic solution with three-node controllers. The value in this case is -14.12 and the policies of the agents are different in this case. Figure 11.13 shows an optimal deterministic solution with four-node controllers. The value in this case is -3.66 and the policies of the agents are again different.

Figure 11.14 shows stochastic two-node controllers for this problem. The large arrow points to the initial state. Each state leads to multiple actions shown in rectangles with the probability of the action attached to the link. Each observation then leads to a stochastic transition to one of the two states. The value in this case is -19.30 , a slight improvement over the two-node deterministic controller. A three-node stochastic controller (not shown) can achieve a value of -9.94 .

Formally, these solutions assign a *local policy* to each agent i , δ_i , which is a mapping from local histories of observations or internal memory states to actions. A *joint policy*, $\delta = \langle \delta_1, \dots, \delta_n \rangle$, is a tuple of local policies, one for each agent.

For a finite-horizon problem with T steps, the value of a joint policy δ with initial state s_0 is

$$V^\delta(s_0) = E \left[\sum_{t=0}^{T-1} R(\vec{a}_t, s_t) | s_0, \delta \right].$$

For an infinite-horizon problem, with initial state s_0 and discount factor $\gamma \in$

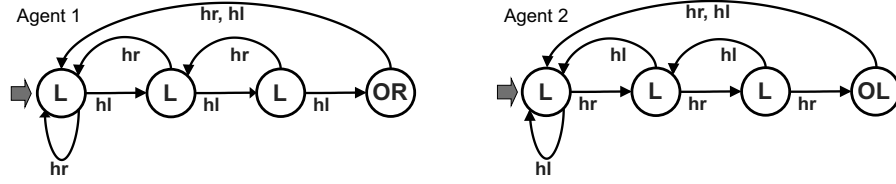


Figure 11.13: Optimal four-node deterministic controllers for multiagent tiger.

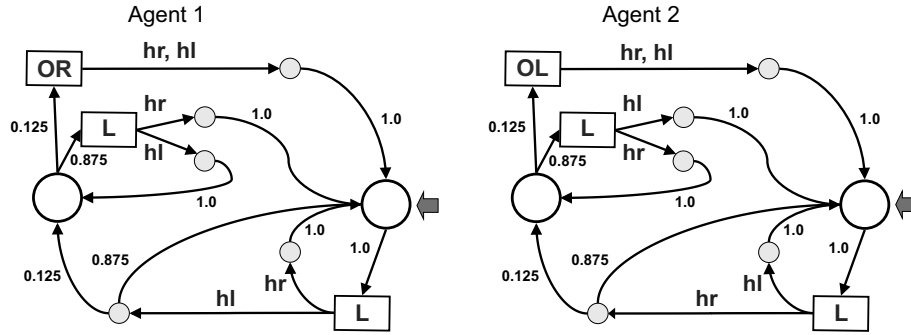


Figure 11.14: Stochastic two-node controllers for multiagent tiger.

$[0, 1)$, the value of a joint policy δ is

$$V^\delta(s_0) = E \left[\sum_{t=0}^{\infty} \gamma^t R(\vec{a}_t, s_t) | s_0, \delta \right].$$

5.1.2 The Complexity of DEC-POMDPs

Complexity analysis of DEC-POMDPs has shown that the finite-horizon problem is NEXP-hard. What is striking is that the problem remains NEXP-hard even when restricted to a two-agent DEC-MDP. This is in contrast to the complexity of finite-horizon MDPs and POMDPs, which is P-complete and PSPACE-complete, respectively. Although it is not known whether the classes P, NP, and PSPACE are distinct, it is known that $P \neq \text{NEXP}$, and thus DEC-POMDPs are provably intractable. Furthermore, assuming $\text{EXP} \neq \text{NEXP}$, the problems take super-exponential time to solve in the worst case. These complexity results reveal some fundamental differences between centralized and decentralized control of Markov decision processes.

These results, however, represent the worst-case complexity of the general model. This presents the question of whether problems that arise in practice are intractable, and whether the structure and characteristics of some real-world problems make them fundamentally easier to solve. For example, mobile robots are

largely independent agents. They move and take actions using their private actuators, which are often totally independent of the actions of other robots operating in the environment. This property is called *transition independence*. Another useful property that is sometimes satisfied is *observation independence* – guaranteeing that the observations of one agent depend only on a component of the underlying state that is not affected by the actions of the other agents. Analysis of the problem shows that these assumptions could lead to a problem of lower complexity. For example, a DEC-MDP that satisfies transition and observation independence can be solved in exponential time [30].

5.2 Solving Finite-Horizon DEC-POMDPs

Dynamic programming algorithms for solving finite-horizon DEC-POMDPs construct and evaluate policy trees incrementally, starting with the final step and moving backward toward the first step. Policy trees of the current iteration become subtrees of the policies built in succeeding iterations. The key to success is reducing the amount of time and space either by performing pruning of irrelevant policies, or – in the case of approximate algorithms – pruning policies that are less likely to be useful.

Let q_i denote a policy tree and Q_i a set of policy trees for agent i . Q_{-i} denotes the sets of policy trees for all agents but agent i . A joint policy $\vec{q} = \langle q_1, q_2, \dots, q_n \rangle$ is a vector of policy trees and $\vec{Q} = \langle Q_1, Q_2, \dots, Q_n \rangle$ denotes the sets of joint policies. Evaluating a joint policy \vec{q} can be done as follows:

$$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V(s', \vec{q}_{\vec{o}}) \quad (11.1)$$

where \vec{a} are the actions at the roots of trees \vec{q} and $\vec{q}_{\vec{o}}$ are the subtrees of \vec{q} after obtaining observations \vec{o} .

In multiagent settings, agents have to reason about the possible future policies of the other agents in order to choose optimal actions. The standard *belief-state* of a POMDP – a probability distribution over world states – is no longer suitable. Instead, it is necessary to use a *multiagent belief state*, which is a probability distribution over system states and policies of all other agents: $b_i \in \Delta(S \times Q_{-i})$. Other forms of multiagent belief states could be used to capture the uncertainty about the beliefs or intentions of other agents, but the above form of belief state, also called *generalized belief state*, is the one used in this chapter because it is most commonly used in existing algorithms.

One of the early class of algorithms for solving DEC-POMDPs is the “Joint Equilibrium-Based Search for Policies” (JESP), which seeks to find a joint policy that is locally optimal. That is, the solution cannot be improved by any single

```

1  Generate a random joint policy
2  repeat
3      foreach agent  $i$  do
4          Fix the policies of all the agents except  $i$ 
5          for  $t=T$  downto 1 do // forwards
6              Generate a set of all possible belief states:
6               $B^t(\cdot | B^{t+1}, q_{-i}^t, a_i, o_i), \forall a_i \in A_i, \forall o_i \in \Omega_i$ 
7          end
8          for  $t=1$  to  $T$  do // backwards
9              foreach  $b^t \in B^t$  do
10                 Compute the best value for  $V^t(b^t, a_i)$ 
11             end
12         end
13         forall the possible observation sequences do
14             for  $t=T$  downto 1 do // forwards
15                 Update the belief state  $b^t$  given  $q_{-i}^t$ 
16                 Select the best action according to  $V^t(b^t, a_i)$ 
17             end
18         end
19     end
20 until no improvement in the policies of all agents
21 return the current joint policy

```

Algorithm 11.3: DP-JESP for DEC-POMDPs.

agent, *given* the policies assigned to the other agents. The best algorithm in this class, DP-JESP, incorporates three key ideas [44]. First, the policy of each agent is modified while keeping the policies of the others fixed. Second, dynamic programming is used to iteratively construct policies. Third, and most notably, only reachable belief states of the DEC-POMDP are considered for policy construction. This leads to a significant improvement, because there is only an exponential number of different belief states for one agent as opposed to the doubly exponential number of possible joint policies. Algorithm 11.3 summarizes the operation of DP-JESP [44]. This approach only guarantees local optimality and still leads to exponential complexity due to the exponential number of possible belief points. The algorithm could solve small benchmark problems up to horizon 7.

An *exact dynamic programming* (ExactDP) algorithm for solving DEC-POMDPs has been developed as well [32]. In every iteration, this algorithm first exhaustively *backups* the policy trees of the previous iteration, then prunes all the *dominated* policies. A policy of an agent is dominated by another policy when the value of *every* complete policy that includes it as a subtree can be improved (or

```

1 Initialize all depth-1 policy trees
2 for  $t=1$  to  $T$  do // backwards
3   | Perform full backup on  $\vec{Q}^t$ 
4   | Evaluate policies in  $\vec{Q}^{t+1}$ 
5   | Prune dominated policies in  $\vec{Q}^{t+1}$ 
6 end
7 return the best joint policy in  $\vec{Q}^T$  for  $b^0$ 

```

Algorithm 11.4: Exact DP for DEC-POMDPs.

preserved) by replacing it with the other policy. This property must be satisfied for *every* set of policies of the other agents and *every* belief state. It is clear that dominated policies are not needed to construct an optimal solution. Dominance can be tested efficiently using a linear program. The algorithm can solve partially-observable stochastic games with minimal changes, as it can produce *all* the non-dominated policies for each agent. This process, summarized in Algorithm 11.4, is the first DP algorithm that could produce a globally optimal solution of a DEC-POMDP. Unsurprisingly, this approach runs out of memory very quickly because the number of possible (non-dominated) joint policies grows doubly exponentially over the horizon. Even with very significant pruning, the algorithm can only solve small benchmark problems with horizons 4–5. But it introduces important pruning principles that prove useful in designing effective approximations.

Several improvements of the ExactDP algorithm have been proposed. Since some regions of the belief space are not reachable in many domains, the point-based DP (PBDP) algorithm computes policies only for the subset of reachable belief states [58]. Unlike DP-JESP, PBDP generates a full set of current-step policies and identifies the reachable beliefs by enumerating all possible top-down histories. This guarantees optimality with a somewhat more aggressive pruning. The worst-case complexity is thus doubly exponential due to the large number of possible policies and histories.

While the above algorithms introduced important ideas, it became clear that to improve scalability, it is necessary to perform more aggressive pruning and limit the amount of memory used by solution methods. The memory-bounded DP (MBDP) algorithm presented a new paradigm that allowed the algorithm to have linear time and space complexity with respect to the problem horizon [53]. MBDP, shown in Algorithm 11.5, employs top-down heuristics to identify the most useful belief states and keeps only a *fixed* number of policies selected based on these belief states. The number of policies maintained per agent is a constant called *maxTrees*. To assure linear space, however, it is not sufficient to limit the number of policies per agent, because the size of each policy tree grows expo-

```

1 Initialize all depth-1 policy trees
2 for  $t=1$  to  $T$  do // backwards
3   Perform full backup on  $\vec{Q}^t$ 
4   Evaluate policies in  $\vec{Q}^{t+1}$ 
5   for  $k=1$  to  $maxTrees$  do
6     Select a heuristic  $h$  from the heuristic portfolio
7     Generate a state distribution  $b \in \Delta(S)$  using  $h$ 
8     Select the best joint policy  $\vec{q}^{t+1}$  in  $\vec{Q}^{t+1}$  for  $b$ 
9   end
10  Prune all the policies except the selected ones
11 end
12 return the best joint policy in  $\vec{Q}^T$  for  $b^0$ 

```

Algorithm 11.5: Memory-bounded DP for DEC-POMDPs.

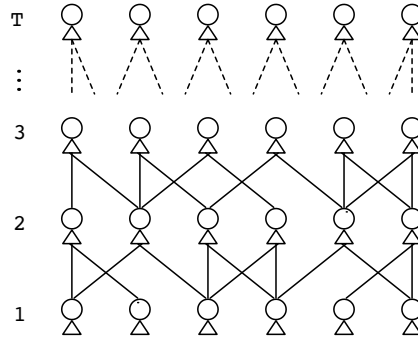


Figure 11.15: A set of $maxTrees$ policy tree can be represented compactly by reusing a fixed number of $maxTrees$ subpolicies of the previous level.

nentially with the horizon. To address that, MBDP deploys an efficient method to reuse subpolicies in a given policy tree. At each level, the new branches of the tree point to one of the $maxTrees$ policies of the previous level, as illustrated in Figure 11.15. This memory-bounded policy representation enables the algorithm to solve much larger problems with essentially arbitrary horizons. A number of algorithmic improvements in MBDP and its variants have made it possible in recent years to solve effectively larger problems using dozens of $maxTrees$ per level.

5.3 Solving Infinite-Horizon DEC-POMDPs

As illustrated earlier, finite-state controllers can be used to summarize unbounded observation histories using finite memory. One controller is used per agent, where

the state of the controller changes based on the observation sequence of the agent, and in turn the agent's actions are based on the state of its controller. When these functions are deterministic, optimizing controllers can be tackled using standard heuristic search methods. When these mappings are stochastic, the search for optimal controllers becomes a continuous optimization problem. We describe in this section two approaches for optimizing such stochastic controllers.

Definition 11.8 (Local finite-state controller) *Given a DEC-POMDP, a local finite-state controller for agent i is a tuple $\langle Q_i, \psi_i, \eta_i \rangle$, where Q_i is a finite set of controller nodes, $\psi_i : Q_i \rightarrow \Delta A_i$ is a stochastic action selection function, and $\eta_i : Q_i \times A_i \times O_i \rightarrow \Delta Q_i$ is a stochastic transition function.*

An *independent joint controller* is a set of local finite-state controllers, one for each agent, that together determine the conditional distribution $P(\vec{a}, \vec{q}' | \vec{q}, \vec{o})$. The controllers are *independent* in that the local memory state transitions and action selection functions of one agent are independent of the memory states and observations of the other agents, making the policy suitable for decentralized operation.

5.3.1 Correlated Joint Controllers

While agents do not have access to the local information of other agents in a DEC-POMDP, they *can* benefit from performing correlated actions. This can be achieved using a *correlation device* – a mechanism that can facilitate coordination using an additional finite-state controller whose state is accessible by all the agents [6]. The correlation device mimics a random process that is independent of the controlled system. Agents use the extra signal from the device to select actions, but they cannot control the correlation device. Such mechanism can be implemented in practice by giving each agent the same stream of random bits.

Definition 11.9 (Correlation device) *A correlation device is a tuple $\langle C, \psi \rangle$, where C is a finite set of states, and $\psi : C \rightarrow \Delta C$ is a state transition function. At each time step, the device makes a transition and all the agents observe its new state.*

The definition of a local controller can be extended to consider the shared signal c provided by a correlation device. The local controller for agent i becomes a conditional distribution of the form $P(a_i, q'_i | c, q_i, o_i)$. A correlation device together with the local controllers for each agent form a joint conditional distribution $P(c', \vec{a}, \vec{q}' | c, \vec{q}, \vec{o})$, called a *correlated joint controller*.

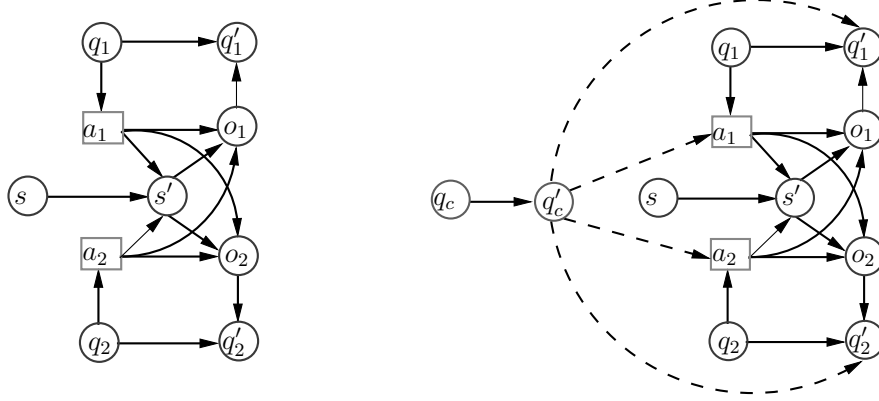


Figure 11.16: A slice of a two-agent DEC-POMDP where actions are selected based on internal states q_i with (right) and without (left) a correlation device q_c .

The value of a correlated joint controller can then be computed by solving a set of linear equations, one for each $s \in S$, $\vec{q} \in \vec{Q}$, and $c \in C$:

$$V(s, \vec{q}', c) = \sum_{\vec{a}} P(\vec{a}|c, \vec{q}) \left[R(s, \vec{a}) + \gamma \sum_{s', \vec{o}, \vec{q}', c'} P(s', \vec{o}|s, \vec{a}) P(\vec{q}'|c, \vec{q}, \vec{a}, \vec{o}) P(c'|c) V(s', \vec{q}', c') \right]$$

Figure 11.16 illustrates a slice (similar to a dynamic Bayesian network) of a two-agent DEC-POMDP with and without a correlation device. The underlying state s and observations o_i are determined by the DEC-POMDP model, while the controller state q_i and action a_i are determined by the policy parameters.

5.3.2 Policy Iteration for Infinite-Horizon DEC-POMDPs

An infinite-horizon DEC-POMDP could produce infinitely many observation sequences. To approach near-optimal value, it may be necessary to increase the controller size. In fact, ϵ -convergence can only be guaranteed when we increase the number of controller states using an *exhaustive backup* operation. An exhaustive backup introduces new controller states for each possible action and each possible branch given an observation to existing controller states [5]. This is similar to the exhaustive backup in the finite-horizon case, except that we grow existing controllers rather than policy trees.

To formalize this process, let Q_i^t denote the set of controller nodes for agent i after iteration t . For each possible one-step policy, a new controller node is added. Thus, for each agent i , $|A_i| |Q_i|^{|\Omega_i|}$ nodes are added to the controller. In the finite-horizon algorithm, the exhaustive backup was followed by a pruning step, eliminating dominated policy trees. Here, an analogous procedure can be used [5].

input : DEC-POMDP, correlated joint controller, convergence parameter ϵ
output: A correlated joint controller that is ϵ -optimal for all states.

```

1 begin
2    $t \leftarrow 0$ 
3   while  $\gamma^{t+1} \cdot |R_{max}| / (1 - \gamma) > \epsilon$  do
4      $t \leftarrow t + 1$ 
5     Evaluate correlated joint controller by solving a system of linear equations
6     Perform an exhaustive backup to add nodes to the local controllers
7     Perform value-preserving transformations on the controller
8   end
9   return correlated joint controller
10 end
```

Algorithm 11.6: Policy iteration for infinite-horizon DEC-POMDPs.

Definition 11.10 (Value-preserving transformation) *Given two correlated joint controllers C and D with node sets \vec{Q} and \vec{R} , respectively, we say that changing controller C to D is a value-preserving transformation if there exist mappings $f_i : Q_i \rightarrow \Delta R_i$ for each agent i and $f_c : Q_c \rightarrow \Delta R_c$ such that:*

$$V(s, \vec{q}) \leq \sum_{\vec{r}} P(\vec{r} | \vec{q}) V(s, \vec{r})$$

The goal of a value-preserving transformation is to reduce the size of a controller without decreasing its value, or to improve the value without changing the size. In general, reducing the size of the controller is necessary between exhaustive backup steps because those steps increase the size of the controller in a doubly exponential manner. Several such transformations that can be implemented efficiently using linear programming have been formulated [5].

The complete policy iteration procedure, sketched in Algorithm 11.6, interleaves exhaustive backups with value-preserving transformations. Unlike single agent MDPs, there is no Bellman residual for testing convergence in this case. Therefore, it is necessary to use the discount factor γ and the number of iterations to define a simpler ϵ -convergence test. Let $|R_{max}|$ denote the largest absolute value of a one-step reward in the DEC-POMDP. Then the algorithm terminates after iteration t if $\gamma^{t+1} \cdot |R_{max}| / (1 - \gamma) \leq \epsilon$. Intuitively, the algorithm exploits the fact that due to discounting, at some point the future rewards collected are negligible.

As with optimal algorithms for finite-horizon DEC-POMDPs, producing near-optimal controllers is intractable. In practice, the value-preserving transformations cannot reduce the size of the controllers sufficiently to continue executing the algorithm until the convergence criterion is met. However, several approximate techniques for infinite-horizon DEC-POMDPs have been developed based

For variables of each agent i : $x(q_i, a_i)$, $y(q_i, a_i, o_i, q'_i)$ and $z(\vec{q}, s)$

Maximize $\sum_s b_0(s) z(\vec{q}^0, s)$, subject to

The Bellman constraints:

$$\forall \vec{q}, s \quad z(\vec{q}, s) =$$

$$\sum_{\vec{a}} \left(\prod_i x(q_i, a_i) \left[R(s, \vec{a}) + \gamma \sum_{s'} P(s'|s, \vec{a}) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q}'} \prod_i y(q_i, a_i, o_i, q'_i) z(\vec{q}', s') \right] \right)$$

And probability constraints for each agent i :

$$\begin{aligned} \forall q_i \sum_{a_i} x(q_i, a_i) &= 1, \quad \forall q_i, o_i, a_i \sum_{q'_i} y(q_i, a_i, o_i, q'_i) = 1 \\ \forall q_i, a_i \quad x(q_i, a_i) &\geq 0, \quad \forall q_i, o_i, a_i \quad y(q_i, a_i, o_i, q'_i) \geq 0 \end{aligned}$$

Table 11.1: The NLP defining a set of optimal fixed-size DEC-POMDP controllers. For each agent i , variable $x(q_i, a_i)$ represents $P(a_i|q_i)$, variable $y(q_i, a_i, o_i, q'_i)$ represents $P(q'_i|q_i, a_i, o_i)$, and variable $z(\vec{q}, s)$ represents $V(\vec{q}, s)$ where \vec{q}^0 represents the initial controller node for each agent.

on these principles by simply restricting the size of each controller and optimizing value with a *bounded* amount of memory. We discuss one such approach below.

5.3.3 Optimizing Fixed-Size Controllers Using Non-Linear Programming

With a fixed controller size, the problem of optimizing the joint controller can be represented as a non-linear program (NLP) by creating a set of new variables that represent the values of each set of controller states and underlying world state. Unlike the dynamic programming backups, which iteratively improve the probabilities and could get stuck in low-quality local optima, the NLP approach allows both the values and probabilities in the controller to be optimized simultaneously. While the NLP is generally harder to solve, the approach results in a search process that is more sophisticated and can leverage state-of-the-art solvers. Existing NLP solvers do not guarantee global optimality, but experimental results show that the NLP formulation is advantageous [1]. In practice, DEC-POMDPs can have small optimal controllers or can be approximated effectively with small controllers. Furthermore, the NLP approach can optimize value for a specific given initial *belief* state, thus making better use of limited controller size.

The non-linear program for a DEC-POMDP with an arbitrary number of agents is shown in Table 11.1. It optimizes the value of a set of fixed-size con-

trollers given an initial state distribution and the DEC-POMDP model. The variables for this problem are the action selection and node transition probabilities for each node of each agent's controller as well as the joint value of a set of controller nodes. Hence, these variables are for each agent i , $P(a_i|q_i)$ and $P(q'_i|q_i, a_i, o_i)$, and for the set of agents and any state, $V(\vec{q}, s)$. The NLP objective is to maximize the value of the initial set of nodes at the initial state distribution. The constraints include the Bellman constraints and additional probability constraints. The Bellman constraints, which are non-linear, ensure that the values are correct given the action and node transition probabilities. The probability constraints ensure that the action and node transition values are proper probabilities. It is straightforward to add a correlation device to the NLP formulation simply by adding a new variable for the transition function of the correlation device. As expected, a correlation device can improve the value achieved by the NLP approach, particularly when each controller is small [1].

6 Multiagent Execution

We conclude this chapter by turning to the actual execution of multiagent planning and control decisions. To the extent that the knowledge used for decision making was correct, the actual trajectory of the world state should mimic the expectations of the agents. Even in cases where actions or observations are uncertain, as is modeled in DEC-POMDPs, the planning and control decisions should have anticipated the possibilities and formulated responses to the foreseen contingencies.

Of course, the rosy picture above can fail to materialize, when the model of the world used by agents for making planning and control decisions is incorrect or incomplete relative to the agents' true world. In such situations, agents can find themselves in unanticipated states, and need to decide how to respond in the near-term, and perhaps also how to update their models so as to make better planning and control decisions in the future. In this section, we can only scratch the surface of the challenges posed in multiagent execution, and of some approaches to overcoming them.

6.1 Multiagent Plan Monitoring

Detecting deviations from anticipated trajectories is one challenge that is significantly more difficult in multiagent settings than single-agent settings. A single agent can use its observations to form beliefs about its current state, and then determine whether it had anticipated possibly having such beliefs at this point in its execution. If not, the agent can invoke a response that could attempt to repair its existing plan by injecting new actions that in expectation will return it to an an-

anticipated trajectory. Or the agent could treat its new beliefs about its current state as the starting point for building a new plan to achieve its objectives. See [50] for techniques of this kind.

In a multiagent system, certainly the same kind of process could occur, but now recovery is harder: an agent cannot in general inject new actions or replace its old plan with a new one without coordinating with other agents, resolving any new interagent faults that its changed plan introduces. Furthermore, it could be that a better way of recovering from such a deviation would be to have one or more other agents change their plans, even though their old plans were proceeding as expected.

Even more problematic are situations where no agent in isolation perceives a deviation from expectations (each foresaw that its current state might have arisen) but the agents collectively have reached an unexpected joint state. Detecting such a deviation requires not only that agents share local information, but that one or more agents are knowledgeable about which (partial) joint states are expected and which are not. Effectively, monitoring the execution of a multiagent plan can amount to a non-trivial collaborative problem-solving effort among the agents.

6.2 Multiagent Plan Recovery

In the multiagent setting, the same dilemma occurs as in single-agent planning when execution diverges from what was planned: should the agent(s) try to reuse some of the old plan(s) so as to take advantage of the effort that went into constructing them, or should the agent(s) build entirely new plan(s) to achieve their objectives from the current state. If they choose the latter route, then replanning in the multiagent context can be accomplished with any of the techniques already discussed.

Plan repair is potentially more cost-efficient, but more complicated. If only some agents need to repair their plans (the others have not deviated), repair nonetheless could involve all of the agents as the repaired plans need to be coordinated with others' unchanged plans. This could lead to others needing to change their plans, in a chain reaction of planning and coordination efforts. And this does not even account for opportunities for agents to reallocate responsibilities among themselves, such as if an agent that has deviated from expectations has exhausted a resource in its attempt to establish a condition in the world, and it *must* now fall to another agent that has reserve resources.

The preceding discussion has further made an important assumption – that the agents should treat the deviation as an anomaly rather than as an indication that the planning knowledge is itself flawed. More broadly, agents can view divergence from expectations as a learning opportunity to correct/improve the models used for planning.

A simple example of such a response can occur in the case of social laws. The designer(s) of the laws used the models of goals/rewards to identify states to avoid, and of actions to identify and prohibit bad precursor state-and-action combinations. If a state to avoid is reached nonetheless, the agents can update their transition models to build better plans/laws in the future, or might perform Q-learning to directly learn what actions not to take in particular states.

A danger in doing such learning, as discussed in Chapter 10, is that if multiple agents learn simultaneously, their local adaptations might not combine into a coherent joint adaptation. For example, if mobile robotic agents collide in a particular location, there is a danger that they *all* might now avoid that location, which would make deliveries to that location impossible. Techniques for controlling learning and adaptation in multiagent systems (see Chapter 10) is thus pertinent in this context.

6.3 Multiagent Continuous Planning

The extreme case of multiagent plan monitoring and repair/replanning is where agents are continually reconsidering and revising their plans. When repair/replanning is punctuated, it is not unreasonable to assume that agents can all suspend their plan executions until that process is complete, and then they proceed with executing their (new) coordinated plans unless and until another deviation occurs. This is not without problems in dynamic domains, where the environment keeps changing while agents are thinking. Thus, when planning is continuous, agents cannot afford to wait for convergence to an assuredly coordinated joint plan. Instead, agents should be able to revise and pursue plans despite those plans perhaps being (temporarily) uncoordinated.

One of the earliest examples of this form of continuous planning in multiagent systems was an approach called *partial global planning* (PGP) [20]. PGP was applied to the interpretation problem of tracking vehicles in a distributed sensor network. This kind of application is particularly forgiving of lack of coordination, which at worst only wastes computational resources. That is, the interpretation problem can be solved with functionally-accurate cooperation, as described in Section 3.2.2. This is in contrast to problems, like air traffic control, where even brief periods of mis-coordination can have catastrophic consequences.

Each sensor agent builds a plan as to how it will process its data: which data it will process in what order, when it will combine partial interpretations into larger interpretations, etc. Because agents' sensor regions can overlap, if agents can communicate about their plans, then by comparing its plans with its neighbors an agent can decide which (if any) signals in an overlapping region it should process, and which it should leave to others. Furthermore, because vehicles tend to not

disappear at boundaries, agents can help each other by providing partial interpretations near their boundaries to others, to help others focus interpretation problem solving on compatible extensions.

Of course, over time new sensor signals can arrive, or data might prove more noisy and take longer to process than expected. As a result, an agent might change its plan. If others know of this change, then they might in turn change their plans, possibly leading to chain reactions (and even cycles) of plan changes. At times these efforts can lead to significant improvements in joint behavior, but at other times the improvements might be smaller than the communication and computation overhead of attaining them. PGP combats the costs and delays in coordinating responses to such dynamics with various mechanisms, including:

1. **Abstraction:** As already discussed in this chapter, abstraction benefits multi-agent planning by allowing coordination decisions to be based on fundamental agent interactions without getting distracted by details of local plans. In PGP, even though each agent builds a detailed plan for processing signals and constructing larger interpretations, the agents communicate with each other only about what partial interpretations they plan to construct and when. Thus, even though internally an agent might frequently make changes to its detailed plan, other agents' perceptions of its plans generally evolve much more slowly.
2. **Decentralization:** PGP assumes that responsibility for coordination decisions can be distributed among agents in any of a variety of ways, as expressed in a *meta-level organization* (MLO). The MLO defines roles, protocols, and authority structures that the agents use when solving the *coordination* problem, analogously to how the agents' domain-level organization guides the agents' activities in solving the interpretation problem. In particular, the MLO can distribute coordination responsibility such that each agent has authority to modify its own plans based on its current partial view of the plans of (some of) the other agents.
3. **Partial Global Reasoning:** An agent forms a partial global plan by combining the abstract plans it has received from others with its own. The agent can then search through modifications to the partial global plan to find a better joint plan. For example, it might determine that various agents should reorder their tasks to change who is responsible for portions of the overlapping regions, and who will formulate which interpretations. In the PGP implementation, the search is done heuristically and in a greedy hill-climbing fashion, since finding optimal solutions would take much more time and such solutions will likely be made obsolete quickly due to further dynamics. An agent then changes its local plan according to how *it* thinks

the joint plan should change. A key assumption behind this strategy is that all agents, given the same information, would formulate the same revised partial global plan. Hence, unilateral changes to a plan in expectation that other agents will make complementary changes to their plans is warranted given sufficient propagation of planning information based on the MLO.

4. **Communication Planning:** By examining the partial global plan, an agent can determine when an interpretation will be formed by one agent that could be of interest to another agent, and can explicitly plan the communication action to transmit the result. If results need to be integrated into a larger partial interpretation, an agent using PGP will construct a tree of exchanges such that, at the root of the tree, partially integrated results will be at the same agent, who can then construct the complete result.
5. **Asynchrony:** Agents asynchronously adapt their local plans and communication plans to their partial global views. This sacrifices global consistency across plans for rapid responsiveness to changing awareness. Depending on the MLO and the relative domain dynamics, the agents could ultimately converge on consistent joint plans, but PGP allows each agent to pursue its best guess as to what its local plan should be at any given time. If it guesses wrong, its efforts were wasted, but had it idled waiting for consistent plans those cycles would have gone to waste anyway.
6. **Dampened Responsiveness:** As an agent's plan changes in the dynamic environment, it can determine how its abstract plan that it has told other agents about has been affected, if at all. Some changes to a local plan might, for example, delay the formation and transmission of a partial interpretation. The agent must determine whether a change to its abstract plan is worth telling other agents about. In PGP, a simple thresholding strategy was used: if an abstract plan step will occur earlier or later than expected by more than a parameterized number of time steps, then the agent should alert others. A larger threshold effectively introduced "slack" into the system, cutting down on coordination overhead by allowing greater degrees of interagent plan slippage. The parameter would be tuned empirically.

Advances in continual planning in multiagent systems have extended and refined these types of strategies over the years. For example, the DARPA Coordinators program [35] emphasized helping teams of humans who are distributed geographically to manage and time their activities so as to achieve joint objectives. One research thrust for this problem was for the teams to each represent a space of possible schedules to follow, where unfolding events can narrow the space of remaining possibilities, and where a threshold defined on this process determines

whether an agent should inform others about such changes [3]. Another was for agents to explicitly model the probabilities of satisfying their contributions to joint objectives, and updating each other as these probabilities decreased significantly [43].

7 Conclusions

Multiagent planning and control with more agents, capable of more behaviors, operating in uncertain and partially-observable worlds, introduces and compounds daunting computational challenges. Research has sought to exploit structure in problems that allow solutions to be composed from solutions to localized sub-problems, and this chapter has illustrated various strategies for different types of problem structures and performance requirements. Significant progress has been made, and yet substantial challenges remain. We conclude by summarizing other important past and ongoing work in this area.

Multiagent planning has been studied since the founding of the field of distributed AI. Some of the earliest work in this area includes that of Georgeff [26, 27], who developed some of the earliest multiagent plan deconfliction techniques, and of Corkill [14], who developed a distributed version of the NOAH planner created by Sacerdoti [51]. Corkill and colleagues, especially Lesser, pioneered the use of organizational techniques for multiagent control [16]. Decker and Lesser generalized techniques for coordinating agent plans in their work on GPGP, and for representing complex multiagent task networks in their work on TAEMS [39].

Planning for teams of agents was investigated not only by Tambe [49, 59] (Section 4.3), but also by Grosz and Kraus [31], building on concepts from Cohen and Levesque [12]. Multiagent planning and scheduling, involving dealing with temporal constraints, also has a rich literature (e.g., [8, 61]). Other work for coordinating plans that agents largely form separately includes that of Tonino et al. [60].

Other techniques that have formulated the multiagent planning problem in decision-theoretic terms include those that solve problems where agents interact through the assignment (and reassignment) of resources [19, 66], and where agents interact by changing shared state in structured ways that enable each other [4, 64].

Memory-bounded dynamic programming (MBDP) [53] has been dramatically improved in recent years by introducing a variety of methods to reduce the number of observations considered by the policy, and employing efficient pruning techniques. Point-based methods have been recently introduced to cope with the NP-hardness of the backup operation [37]. This algorithm exploits recent advances

in the weighted CSP literature to offer a polytime approximation scheme that can handle a much larger number of belief points (*MaxTrees*). Another technique, trial-based dynamic programming (TBDP) [65], combines the main advantages of DP-JESP with MBDP to avoid the expensive backup operations, allowing problems with much larger state spaces to be tackled.

The locality of agent interaction – the fact that each agent interacts with a small number of neighboring agents – has proved crucial to the development of DEC-POMDP algorithms that can handle dozens of agents. Specialized models such as network distributed POMDPs (ND-POMDPs) have been introduced to capture structured interactions and develop early algorithms that can exploit such structures [45]. More recently, the constrained-based dynamic programming (CBDP) algorithm has been shown to provide magnitudes of speedup thanks to its linear complexity in the number of agents [36]. Algorithms for solving loosely-coupled infinite-horizon problems have also been developed. One promising direction is based on transforming the policy optimization problem to that of likelihood maximization in a mixture of dynamic Bayesian networks [38]. Based on this reformulation, the expectation-maximization (EM) algorithm has been used to compute the policy via a simple message-passing paradigm guided by the agent interaction graph.

8 Exercises

1. **Level 1** Analyze a multiperson problem that you have been involved in solving. Identify localities in the structure of the problem, and strategies for composing an overall solution from solutions to the localized subproblems.
2. **Level 1** Do you agree with the stance taken in this chapter that multi-agent planning requires both that the plan formulation process be distributed among agents and that the resulting plan construct be distributed as well? If so, justify in your own words why you believe this stance is warranted. If not, give counterexamples to this stance and justify why they arguably embody multiagent planning.
3. **Level 2** The broad brushstrokes of social laws to avoid collisions between robots operating in a gridworld was provided in this chapter. This question asks you to elaborate a bit.
 - (a) For an 8 x 8 gridworld (empty other than robots), flesh out an example of the social laws that avoid collisions, indicating for each location where a robot is allowed to go, along with any other laws that the robots should follow.

- (b) Are there gridworld shapes (again, empty other than robots) for which useful social laws cannot be constructed? If so, give examples, and if not, explain why not.
 - (c) Now, say in the 8 x 8 gridworld there is a wall that partitions the environment in half except there is one pair of cells on each side that are connected. How would you build social laws to handle such a case, assuming that a robot might need to visit locations on both sides of the world?
4. **Level 4** Implement a simulation of a robot gridworld. Create in that world simulated robots, and endow them with the capabilities and constraints associated with Traffic Law 2 in [56]. Experimentally investigate the performance of the robots following that traffic law to determine how well they perform as the number of robots increases in a fixed-size grid.
5. **Level 2** Describe and analyze a real-world (human) instance of organizational redesign. What alternative decompositions of the larger problem into interacting roles were possible, and why was the particular choice made during the redesign? Develop a specification for the space of designs, and suggest a search strategy for finding and implementing a good design from that space. Is your search assured to return an optimal solution? Is it efficient?
6. **Level 1** Consider the contract net protocol where announcements can be either about tasks that need to be done or the availability of resources that could be assigned tasks.
- (a) Name a real-life example where task announcement makes much more sense than availability announcement. Justify why.
 - (b) Now name a real-life example where availability announcement makes much more sense. Justify why.
 - (c) Let's say that you are going to build a mechanism that oversees a distributed problem-solving system, and can "switch" it to either a task or availability announcement mode.
 - i. Assuming communication costs are negligible, what criteria would you use to switch between modes? Be specific about what you would test.
 - ii. If communication costs are high, now what criteria would you use? Be specific about what you would test.

7. **Level 2/3** We noted that task announcing can be tricky: If a manager is too fussy about eligibility, it might get no bids, but if it is too open it might have to process too many bids, including those from inferior contractors. Let us say that the manager has n levels of eligibility specifications from which it needs to choose one. Describe how it would make this choice based on a decision-theoretic formulation. How would this formulation change if it needed to consider competition for contractors from other managers?
8. **Level 2** A folk theorem in the organization literature is that in human organizations, task decompositions invariably lead to clear assignments of subtasks to members of the organization. Give an example of where decomposition without look-ahead to available contractors can be detrimental. Give an example where biasing decomposition based on available contractors can instead be detrimental. Finally, give an algorithm for alternating between decomposition and assignment to incrementally formulate a distributed problem-solving system. Is your algorithm assured of yielding an optimal result? Is it complete?
9. **Level 1** Consider the pursuit task, with four predators attempting to surround and capture a prey. Define an organizational structure for the predators. What are the roles and responsibilities of each? How does the structure indicate the kinds of communication patterns (if any) that will lead to success?
10. **Level 2** Consider the following simple instance of the distributed delivery task. Robot A is at position α and robot B is at position β . Article X is at position ξ and needs to go to position ψ , and article Y is at position ψ and needs to go to ζ . Positions α , β , ξ , ψ , and ζ are all different.
 - (a) Define in STRIPS notation, suitable for partial-order planning, simple operators Pickup, Dropoff, PickDrop, and Return, where Pickup moves the robot from its current position to a pickup position where it then has the article associated with that position; Dropoff moves a robot and an article it holds to a dropoff position where it no longer has the article; PickDrop combines the two (it drops off its article and picks up another associated with that position); and Return moves a robot back to its original position.
 - (b) Using these operators, generate the partial-order plan with the fewest plan steps to accomplish the deliveries. Decompose and distribute this plan to the robots for parallel execution, inserting any needed synchronization actions. How does the use of multiple robots affect the plan execution?

- (c) Using the operators, generate the partial-order plan that, when distributed, will accomplish the deliveries as quickly as possible. Is this the same plan as in the previous part of this problem? Why or why not?
11. **Level 2** Given the previous problem, include in the operator descriptions conditions that disallow robots to be at the same position at the same time (for example, a robot cannot do a pickup in a location where another is doing a dropoff). Assuming each robot was given the task of delivering a different one of the articles, generate the individual plans and then use the plan modification algorithm to formulate the synchronized plans, including any synchronization actions into the plans. Show your work.
12. **Level 2** Consider the delivery problem given before the previous problem. Assume that delivery plans can be decomposed into 3 subplans (pickup, dropoff, and return), and that each of these subplans can further be decomposed into individual plan steps. Furthermore, assume that robots should not occupy the same location at the same time – not just at dropoff/pickup points, but throughout their travels. Use the hierarchical behavior-space search algorithm to resolve potential conflicts between the robots' plans, given a few different layouts of the coordinates for the various positions (that is, where path-crossing is maximized and minimized). What kinds of coordinated plans arise depending on at what level of the hierarchy the plans' conflicts are resolved through synchronization?
13. **Level 3** Assume that distributed delivery robots are in an environment where delivery tasks pop up dynamically. When a delivery needs to be done, the article to be delivered announces that it needs to be delivered, and delivery agents within a particular distance from the article hear the announcement.
- (a) Assume that the distance from which articles can be heard is small. What characteristics would an organizational structure among the delivery agents have to have to minimize the deliveries that might be overlooked?
 - (b) Assume that the distance is instead large. Would an organizational structure be beneficial anyway? Justify your answer.
 - (c) As they become aware of deliveries to be done, delivery agents try to incorporate those into their current delivery plans. But the dynamic nature of the domain means that these plans are undergoing evolution. Under what assumptions would partial global planning be a good approach for coordinating the agents in this case?

- (d) Assume you are using partial global planning for coordination in this problem. What would you believe would be a good planning level for the agents to communicate and coordinate their plans? How would the agents determine whether they were working on related plans? How would they use this view to change their local plans? Would a hill-climbing strategy work well for this?
14. **Level 1** Given a flawed multiagent plan M with more than one unflagged causal link to adjust, which causal link should the plan modification algorithm prefer to adjust? Justify your (heuristic) selection strategy.
 15. **Level 1** Give an example of a real-world situation in which multiple agents operate under partial observability and each agent has access to *different* partial information about the overall state. Can agents share all their knowledge all the time in your example? If yes, explain how. If not, explain why.
 16. **Level 2** The DEC-POMDP model (Definition 11.7) does not include explicit communication. Suppose that each agent can broadcast certain messages to all the other agents in each action cycle. Define precisely this kind of a DEC-POMDP with *two* agents and explain why it is *not* an extension of the standard model (i.e., show that every DEC-POMDP with explicit communication can be reduced to a standard DEC-POMDP).
 17. **Level 3** The communication model presented in the previous question allows each agent to broadcast a message to all the other agents in each step. This means that the space of possible *joint* messages received by each agent grows exponentially with the number of agents. Consider a more scalable communication model that allows only one agent to broadcast a message in each cycle (e.g., when multiple agents try to broadcast messages simultaneously, this may either result in failure or success of just one agent). Define precisely one such model and determine whether it is reducible to a standard DEC-POMDP or not.
 18. **Level 2** Consider the complete specification of the multiagent tiger problem shown in Table 11.2.
 - (a) Derive the values of the deterministic policies for horizons 1–3 shown in Figure 11.9.
 - (b) Derive the values of the deterministic finite-state controller policies shown in Figure 11.11.

Tiger observation table

Joint action	State	hl	hr
$\langle L, L \rangle$	TL	0.85	0.15
$\langle L, L \rangle$	TR	0.15	0.85
$\langle O^*, * \rangle$	*	0.5	0.5
$\langle *, O^* \rangle$	*	0.5	0.5

Tiger transition table

Joint action	Current state	Next state	Probability
$\langle L, L \rangle$	TL	TL	1.0
$\langle L, L \rangle$	TR	TR	1.0
$\langle O^*, * \rangle$	TL	TL	0.5
$\langle O^*, * \rangle$	TR	TR	0.5
$\langle *, O^* \rangle$	TR	TR	0.5
$\langle *, O^* \rangle$	TR	TL	0.5

Tiger reward table

Joint action	State	Value	Joint action	State	Value	Joint action	State	Value
$\langle L, L \rangle$	*	-2	$\langle OR, L \rangle$	TR	-101	$\langle OL, L \rangle$	TR	9
$\langle L, OR \rangle$	TR	-101	$\langle OR, L \rangle$	TL	9	$\langle OL, L \rangle$	TL	-101
$\langle L, OR \rangle$	TL	9	$\langle OR, OR \rangle$	TR	-50	$\langle OL, OR \rangle$	*	-100
$\langle L, OL \rangle$	TR	9	$\langle OR, OR \rangle$	TL	20	$\langle OL, OL \rangle$	TR	20
$\langle L, OL \rangle$	TL	-101	$\langle OR, OL \rangle$	*	-100	$\langle OL, OL \rangle$	TL	-50

Table 11.2: Tiger observation, transition, and reward tables.

19. **Level 2** In the multiagent tiger problem, suppose that the reward for opening the correct door (e.g., $\langle OR, OR \rangle$ when the state is TR) is increased to 50. Is the horizon 1 policy in Figure 11.9 still optimal? If not, what is the optimal policy (and its value)? Repeat the question for horizons 2 and 3.
20. **Level 2** In the multiagent tiger problem, the optimal policy is to listen for several steps before opening any door. If the observation probabilities increase from 0.85 to 0.9, does that change the optimal horizon 1 policy? What about horizons 2 and 3?
21. **Level 2/3** If all agents share their observations with each other at each step, the problem becomes centralized. In the multiagent tiger problem, what would the resulting observations (and their probabilities) be for each agent when observations are shared? How does this change the optimal policies for horizons 1 and 2? Would the agents ever choose to open different doors? Is a centralized solution (with shared observations) always guaranteed to have value at least as high as a decentralized solution?
22. **Level 2/3** If the transition and observation probabilities are independent for each agent and the reward values are additive between the agents, the problem can be solved as a set of independent problems whose solutions can be summed together. In the multiagent tiger problem, the observations are

independent, but the transitions and rewards depend on all agents. Consider the case where the tiger does not transition after a door is opened and each agent receives a reward of 10 for opening the correct door, -50 for opening the incorrect door and -1 for listening. What are the optimal horizon 1, 2, and 3 policies for this case?

23. **Level 2/3** Given the same number of nodes, stochastic controllers often allow higher-valued policies to be constructed compared to deterministic controllers. Is there a one-node stochastic controller with a higher value than the optimal one-node deterministic controller in the multiagent tiger problem? If there is, construct one. Otherwise, prove that this is not possible in this case.

Acknowledgments

We thank Christopher Amato for invaluable feedback on the presentation of decentralized POMDPs, and for his help generating the corresponding examples, figures, and exercises. Jeffrey Cox's work on multiagent partial-order causal-link planning has been drawn on heavily in this chapter as well. We thank all our former students and collaborators for the many fruitful discussions on the topics covered in this chapter, which helped sharpen our understanding of multiagent planning.

References

- [1] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 21(3):293–320, 2010.
- [2] Christer Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.
- [3] Laura Barbulescu, Zack Rubinstein, Stephen Smith, and Terry Lyle Zimmerman. Distributed coordination of mobile agent teams: The advantage of planning ahead. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1331–1338, 2010.
- [4] Raphen Becker, Victor Lesser, and Shlomo Zilberstein. Decentralized Markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, volume 1, pages 302–309, 2004.

- [5] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.
- [6] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1287–1292, 2005.
- [7] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 32–37, 2000.
- [8] Aurélie Beynier and Abdel-Ilhah Mouaddib. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, pages 963–969, 2005.
- [9] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, pages 195–210, 1996.
- [10] Kathleen M. Carley and Les Gasser. *Computational Organization Theory*. MIT Press, 1999.
- [11] Bradley J. Clement, Edmund H. Durfee, and Anthony C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, 28:453–515, 2007.
- [12] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.
- [13] Susan E. Conry, Kazuhiro Kuwabara, Victor R. Lesser, and Robert A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1462–1477, 1991.
- [14] Daniel Corkill. Hierarchical planning in a distributed problem-solving environment. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 168–175, 1979.
- [15] Daniel Corkill. Blackboard systems. *AI Expert*, 6(9), January 1991.
- [16] Daniel Corkill and Victor Lesser. The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, 1983.
- [17] Daniel D. Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts, 1982.

- [18] Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.
- [19] Dmitri A. Dolgov and Edmund H. Durfee. Resource allocation among agents with MDP-induced preferences. *Journal of Artificial Intelligence Research*, 27:505–549, 2006.
- [20] Edmund H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Press, 1988.
- [21] Edmund H. Durfee. Organizations, plans, and schedules: An interdisciplinary perspective on coordinating AI systems. *Journal of Intelligent Systems*, 3(2-4):157–187, 1993.
- [22] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Cooperation through communication in a distributed problem solving network. In M. Huhns, editor, *Distributed Artificial Intelligence*, chapter 2. Pitman, 1987.
- [23] Eithan Ephrati and Jeffrey S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 375–380, 1994.
- [24] Eithan Ephrati, Martha Pollack, and Jeffrey S. Rosenschein. A tractable heuristic that maximizes global utility through local plan combination. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 94–101, 1995.
- [25] Michael Fisher and Michael Wooldridge. Distributed problem-solving as concurrent theorem-proving. In *Proceedings of MAAMAW’97*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1997.
- [26] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the 3rd National Conference on Artificial Intelligence (AAAI-83)*, pages 125–129, 1983.
- [27] Michael P. Georgeff. A theory of action for multiagent planning. In *AAAI*, pages 121–125, 1984.
- [28] Piotr J. Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.
- [29] Claudia Goldman and Jeffrey S. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 408–413, 1994.

- [30] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.
- [31] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.
- [32] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.
- [33] Nick R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8:223–250, 1993.
- [34] Subbarao Kambhampati, Mark Cutkosky, Marty Tenenbaum, and Soo Hong Lee. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 199–205, 1991.
- [35] Robert Kohout. The DARPA COORDINATORS program: A retrospective. In *Proceedings of the 2011 International Conference on Collaborative Technologies and Systems (CTS-10)*, page 342, 2011.
- [36] Akshat Kumar and Shlomo Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 561–568, 2009.
- [37] Akshat Kumar and Shlomo Zilberstein. Point-based backup for decentralized POMDPs: Complexity and new algorithms. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 1315–1322, 2010.
- [38] Akshat Kumar and Shlomo Zilberstein. Message-passing algorithms for large structured decentralized POMDPs. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-11)*, pages 1087–1088, 2011.
- [39] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Nagendra Prasad, A. Raja, R. Vincent, P. Xuan, and X. Q. Zhang. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, 2004.
- [40] Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:81–96, 1981.

- [41] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and an experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163, 1980.
- [42] Douglas MacIntosh, Susan Conry, and Robert Meyer. Distributed automated reasoning: Issues in coordination, cooperation, and performance. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1307–1316, 1991.
- [43] Rajiv T. Maheswaran, Craig Milo Rogers, Romeo Sanchez, and Pedro A. Szekely. Decision-support for real-time multi-agent coordination. In *AAMAS'10*, pages 1771–1772, 2010.
- [44] Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multi-agent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, 2003.
- [45] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 133–139, 2005.
- [46] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1323–1330, 2010.
- [47] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
- [48] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.
- [49] David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, pages 71–100, 2003.
- [50] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.
- [51] Earl Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland Inc., 1977.
- [52] Sandip Sen and Edmund H. Durfee. A contracting model for flexible distributed scheduling. *Annals of Operations Research*, 65:195–222, 1996.

- [53] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2009–2015, 2007.
- [54] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.
- [55] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 276–281, 1992.
- [56] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73:231–252, 1995.
- [57] Mark Sims, Daniel Corkill, and Victor Lesser. Automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 16(2):151–185, 2008.
- [58] Daniel Szer and Francois Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 1233–1238, 2006.
- [59] Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of AAAI/IAAI'97*, pages 22–28, 1997.
- [60] Hans Tonino, André Bos, Mathijs de Weerd, and Cees Witteveen. Plan coordination by revision in collective agent based systems. *Artificial Intelligence*, 142(2):121–145, 2002.
- [61] Ioannis Tsamardinos, Martha E. Pollack, and John F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Conference on AI Planning Systems*, pages 264–272, 2000.
- [62] Guandong Wang, Weixiong Zhang, Roger Mailler, and Victor Lesser. Analysis of negotiation protocols by distributed search. In Victor Lesser, Charles Ortiz, and Milind Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 339–361. Kluwer Academic Publishers, 2003.
- [63] Daniel S. Weld. An introduction to least commitment planning. *Artificial Intelligence Magazine*, 15(4):27–61, 1994.
- [64] Stefan J. Witwicki and Edmund H. Durfee. Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 185–192, 2010.

- [65] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Trial-based dynamic programming for multi-agent planning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*, pages 908–914, 2010.
- [66] Jianhui Wu and Edmund H. Durfee. Resource-driven mission-phasing techniques for constrained agents in stochastic environments. *Journal of Artificial Intelligence Research*, 38:415–473, 2010.
- [67] 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*, 10:673–685, 1998.
- [68] Chengqi Zhang. Cooperation under uncertainty in distributed expert systems. *Artificial Intelligence*, 56(1):21–69, 1992.

Chapter 12

Distributed Constraint Handling and Optimization

Alessandro Farinelli, Meritxell Vinyals, Alex Rogers, and Nicholas R. Jennings

1 Introduction

Constraints pervade our everyday lives and are usually perceived as elements that limit solutions to the problems that we face (e.g., the choices we make everyday are typically constrained by limited money or time). However, from a computational point of view, constraints are key components for efficiently solving hard problems. In fact, constraints encode *knowledge* about the problem at hand, and so restrict the space of possible solutions that must be considered. By doing so, they greatly reduce the computational effort required to solve a problem.

Against this background, constraint processing can be viewed as a wide and diverse research area unifying techniques and algorithms that span across many different disciplines including planning and scheduling, operation research, computer vision, automated reasoning, and decision theory. All these areas deal with hard computational problems that can be made more tractable by carefully considering the constraints that define the structure of the problem.

Here we will focus on how constraint processing can be used to address optimization problems in multiagent systems. Specifically, we will consider distributed constraint optimization problems (DCOPs), whereby a set of agents must come to some agreement, typically via some form of negotiation, about which action each agent should take in order to jointly obtain the best solution for the whole system. This framework has been frequently used in the MAS literature

to address problems such as meeting scheduling in human organizations, where agents must agree on a valid meeting schedule while maximizing the sum of individual preferences about when each meeting should be held, or target tracking in sensor networks, where sensors must agree on which target they should focus on to obtain the most accurate estimation of the targets' positions. A key common aspect of DCOPs for MAS is that each agent negotiates *locally* with just a subset of other agents (usually called neighbors), who are those that can directly influence its behaviors. Depending on the problem setting and on the solution technique used, this aspect can greatly reduce the computational effort that each agent faces, making hard problems tractable even for large-scale systems. For example, in the meeting scheduling problem, an agent will directly negotiate with and care about people that it must meet, which is usually a small subset of the agents involved in the whole problem.

In more detail, this chapter aims to provide the reader with a broad knowledge of the main DCOP solution approaches. It describes both exact algorithms and approximate approaches, including those that provide quality guarantees. After reading it, you will understand:

- The mathematical formulation of a DCOP, and how to model distributed decision-making problems using the DCOP framework.
- The main exact solution techniques for DCOPs and the key differences between them in terms of benefits and limitations.
- Why and when approximate solution techniques should be used for DCOPs and the main algorithms in this space.
- Why quality guarantees are important, how to differentiate between various types of quality guarantees, and which techniques are currently available in the literature to achieve them.

The chapter is organized as follows. First, Section 2 presents the mathematical background for constraint networks and distributed constraint processing; next Section 3 provides examples of practical problems that can be addressed using DCOPs and a description of the abstract benchmarking problems commonly used to empirically evaluate the different solution techniques. Section 4 introduces a selection of exact solution algorithms for DCOPs, focusing on two representative examples: (i) ADOPT, as an example of a search-based technique and (ii) DPOP, as an example of a dynamic programming based approach. Section 5 discusses the need for approximate solutions for DCOPs describing some representative techniques in this area, and Section 6 discusses quality guarantees for approximate DCOP solution algorithms, focusing on two representatives: the k -optimality

framework and the bounded max-sum approach. Finally, Section 7 concludes the chapter.

2 Distributed Constraint Handling

A key element for distributed constraint handling is the concept of the constraint network. Here we provide standard formal definitions relating to constraint networks and then we focus on the distributed constraint processing paradigm itself.

2.1 Constraint Networks

A constraint network \mathcal{N} is formally defined as a tuple $\langle X, D, C \rangle$, where $X = \{x_1, \dots, x_n\}$ is a set of *discrete* variables, $D = \{D_1, \dots, D_n\}$ is a set of variable domains, which enumerate all possible values of the corresponding variables, and $C = \{C_1, \dots, C_m\}$ is a set of *constraints*. A constraint C_i can be of two types: hard or soft.

A hard constraint C_i^h is a *relation* R_i defined on a subset of variables $S_i \subseteq X$. Variables in S_i are the *scope* of the constraint and the relation R_i enumerates all the *valid* joint assignments of all variables in the scope of the constraint. Therefore R_i is a subset of the Cartesian product of variable domains that are in the constraint's scope: $R_i \subseteq D_{i_1} \times \dots \times D_{i_r}$ and $r = |S_i|$ is the arity of the relation. A soft constraint C_i^s is a *function* F_i defined again on a subset of variables $S_i \subseteq X$ that comprise the scope of the function. Each function F_i maps every possible joint assignment of all variables in the scope to a real value, therefore $F_i : D_{i_1} \times \dots \times D_{i_r} \Rightarrow \mathfrak{R}$ and, as above, $r = |S_i|$ is the arity of the function.

Notice that constraints can be defined over any subset of the variables; however, most of the work in constraint networks (both solution algorithms and theoretical analysis) focus on binary constraint networks, where each constraint (soft or hard) is defined over two variables. Actually, it is possible to show that every constraint network can be mapped to a binary constraint network;¹ nonetheless here we use the general formalization and specify when the analysis is restricted to the binary case.

When the constraint set involves only hard constraints, the problem we face is known as a constraint satisfaction problem (CSP). In this context, we aim to find an assignment for all the variables in the network that satisfies all constraints. An assignment satisfies a constraint if the tuple of values of variables in the constraint's scope belongs to the constraint's relation. If $(a_{i_1}, \dots, a_{i_r}) \in R_i$ where

¹In general this requires the combination or addition of both variables and constraints [4].

$a_j \in D_j$ and $S_i = \{x_{i_1}, \dots, x_{i_r}\}$, then such an assignment is referred to as a *solution* of the network.

If the constraint set involves soft constraints, then we face a constraint optimization problem (COP) and our aim is to find the *best* solution – that is, an assignment for all variables that satisfies all constraints and that optimizes a global function $F(\bar{a})$. The global function $F(\bar{a})$ is an aggregation of the functions representing the soft constraints (local functions): $F(\bar{a}) = \sum_i F_i(\bar{a}_i)$, where $\bar{a} = (a_1, \dots, a_n)$ with $a_j \in D_j$, and \bar{a}_i is a restriction of \bar{a} to S_i . A COP can be a maximization or a minimization problem. Without loss of generality, we focus here on maximization problems; therefore our aim is to find the assignment \bar{a}^* that satisfies all hard constraints such that:

$$\bar{a}^* = \arg \max_{\bar{a}} \sum_i F_i(\bar{a}_i) \quad (12.1)$$

In general, every CSP can be viewed as an optimization task, where we aim to find the assignment that violates the least number of constraints. This is particularly useful for overconstrained problems where a solution for the CSP might not exist. Specifically, we can assign a constant fixed cost to every violated constraint and search for an assignment that minimizes the sum of the costs. This problem is known as the *max-CSP* problem.

Moreover, we can always encode hard constraints using only soft constraints by using infinite values to penalize assignments that do not satisfy hard constraints. For example, assume, without loss of generality, that we are solving a maximization problem, and let R_i be a relation that corresponds to a hard constraint C_i . We can construct a function $F_i(\bar{a}) = -\infty$ if $\bar{a} \notin R_i$ and $F_i(\bar{a}) = 0$ otherwise.²

2.2 Distributed Constraint Processing

Distributed constraint processing extends the standard constraint processing framework by considering a set of agents that have control over variables and interact to find a solution to the constraint network. As before, we can have satisfaction or optimization tasks resulting in two types of problems: distributed CSP (DCSP) and distributed COP (DCOP). The DCSP paradigm was originally proposed to address coordination problems in a multiagent setting [41]. However in recent years the DCOP framework has received increasing attention as it can better represent many practical application scenarios.

Formally a DCOP can be represented as a constraint network $\mathcal{N} = \langle X, D, C \rangle$ containing soft constraints, plus a set of agents $A = \{A_1, \dots, A_k\}$. Each agent can

²Notice that this may result in inferior solution techniques, as explicit hard constraints might be exploited to reduce the solution search space [9].

control only a subset of the variables $X_i \subseteq X$, and each variable is assigned to exactly one agent. In other words, the assignment of variables to agents must be a partition of the set of variables. Agents can *control* only the variables assigned to them, meaning that they can observe and change the values of their assigned variables only. Moreover, agents are only aware of constraints that involve variables that they can control. Such constraints are usually termed local functions and the sum of these local functions is the local utility of the agent. Finally, two agents are considered neighbors if there is at least one constraint that depends on variables that each controls. Only neighboring agents can directly communicate with each other.

Within this context, the goal for the agents is to find the optimal solution to the constraint network, i.e., to find the assignment for all the variables in the system that optimizes the global function. Thus, in a standard DCOP setting, agents are assumed not to be self-interested, i.e., their goal is to optimize the *global* function and not their local utilities.

Finding an optimal solution for a DCOP is an NP-hard problem, which can be seen by reducing a DCOP to the problem of deciding on the 3-colorability of a graph, a problem known to be NP-complete [26].

In the next section we will present a number of practical problems that can be addressed using the DCOP framework, as well as some exemplar and benchmarking DCOP instances.

3 Applications and Benchmarking Problems

To provide concrete examples of how the DCOP framework can be applied to real-world scenarios we report here on a number of practical problems that can be successfully addressed using the DCOP framework discussed above. Following this we go on to show a set of abstract benchmarking problems that are commonly used to evaluate and compare DCOP solution techniques.

3.1 Real-World Applications

Many real-world applications can be modeled using the DCOP framework, ranging from human-agent organizations to sensor networks and robotics. Here we focus on two such applications that have been frequently used as motivating scenarios for work in the MAS literature.

3.1.1 Meeting Scheduling

The problem of scheduling a set of tasks over a set of resources (e.g., schedule a set of lectures over a set of lecture halls or a set of jobs to a set of processors) is a very common and important problem, which can be conveniently formalized using constraint networks.

A typical example of this is the meeting scheduling problem, which is a very relevant problem for large organizations (e.g., public administration, private companies, research institutes, etc.), where many people, possibly working in different departments, are involved in a number of work meetings. In more detail, people involved in a meeting scheduling problem might have various private preferences for meeting start times; for example, a given employee might prefer his or her meetings to start in the afternoon rather than in the morning (to happily conjugate a late night social life with work!). Given this, the aim is to agree on a *valid* schedule for the meeting while maximizing the sum of the individuals' private preferences. To be valid, a schedule must meet obvious hard constraints, for example, two meetings that share a participant cannot overlap.

A possible DCOP formalization for the meeting scheduling domain involves a set of agents representing the people participating in the meeting and a set of variables that represent the possible starting time of a given meeting according to a participant. Constraints force equality on variables that represent the starting time of the same meeting across different agents and ensure that variables that represent the starting times of different meetings for the same agent are non-overlapping. Finally, preferences can be represented as soft constraints on meeting starting times and the overall aim is to optimize the sum of all the soft constraints. Notice that although in this setting we do have private preferences, we are maximizing the sum of preferences of all the agents, and thus we are still considering a scenario where agents are fully cooperative, i.e., they are willing to diminish their own local utility if it will maximize the global utility.

While this problem could be easily formalized as a centralized COP, in this case a distributed approach not only provides a more robust and scalable solution, but it can also minimize the amount of information agents must reveal to each other (thus preserving their privacy). This is because, as mentioned above, in a DCOP, agents are required to be aware only of constraints that they are involved in. For example, consider a situation whereby Alice must meet Bob and Charles in two separate meetings. In a centralized approach, Alice would have to reveal the list of people she has to meet with. On the other hand, in a DCOP only people involved in any particular meeting will be aware that the meeting is taking place. Thus in our example, Bob does not need to know that Alice will also meet with Charles.

3.1.2 Target Tracking

Target tracking is a crucial and widely studied problem for surveillance and monitoring applications. It involves a set of sensors tracking a set of targets in order to provide an accurate estimate of their positions. Sensors can have different sensing modalities that impact on the accuracy of the estimation of the targets' positions. For example a pan, tilt, and zoom (PTZ) camera could move to focus on a specific area of the environment, reducing observation uncertainty for targets in that area. Moreover, collaboration among sensors is crucial to improve the performance of the system. For example two cameras could decide to track different targets to maximize coverage of the environment, or to both focus on a potentially dangerous target, thus providing a more accurate estimate of its position.

There are several ways to formalize this scenario using the DCOP framework. A widely accepted formulation is one where sensors are represented by agents, and variables encode the different sensing modalities of each sensor. Constraints are usually defined among sensors that have an overlapping sensing range. Each constraint relates to a specific target and represents how the joint choice of sensor modalities impacts on the tracking performance for that target. Constraints can specify the minimum number, or particular number, of sensors that are required to correctly identify a target, or provide a measure of position tracking accuracy for each possible combination of agents' sensing modalities. The global function for the DCOP is the sum of constraints' values. For example, the system could aim to maximize the number of targets correctly identified or to maximize the sum of tracking accuracy over all targets.

Within this context, the main reasons for using a distributed approach for the optimization problem are robustness and scalability, which are both crucial issues in surveillance and monitoring applications. Specifically, a distributed solution improves scalability as it exploits the decomposition of the problem to reason locally – thus reducing the communication and computation that each agent must perform. This is very important as this type of application typically involves the use of hardware devices that have inherent constraints on communication and computation. Furthermore, robustness is enhanced as each agent decides on its own sensing modalities, thus avoiding a central point of failure.

3.2 Exemplar and Benchmarking Problems

As previously remarked, finding an optimal solution for a DCOP is known to be an NP-hard problem. Therefore empirical evaluation of DCOP solution techniques is a crucial point in order to evaluate their likely practical impact. In particular, to have a meaningful comparison between the different solution techniques it is essential to be able to run the various programs on a shared testbed. This prob-

lem has been addressed by the DCOP community using benchmarking problem instances inspired by practical applications, such as meeting scheduling and target tracking.³

In addition, there are also a number of exemplar NP problems that are frequently used to test solution techniques such as propositional satisfaction (SAT) or graph coloring. Here we focus on the latter problem as it has been widely used to evaluate the techniques that will be presented later in this chapter and is particularly useful for illustrative purposes.

The graph coloring problem is an extremely simple problem to formulate and is attractive, since the computational effort associated with finding the solution can be easily controlled using few parameters (e.g., number of available colors, and the ratio of number of constraints to the number of nodes). The constraint satisfaction version of a graph coloring problem can be described as follows: given a graph of any size and k possible colors, decide whether the nodes of the graphs can be colored with no more than k colors, so that any two adjacent nodes have different colors.

In the CSP formulation of the graph coloring problem, nodes are variables, the set of k colors is the variable domain (which is the same for all the variables), and constraints are *not-equal* constraints that hold between any adjacent nodes. An assignment is a map from nodes to colors without constraint violations. The optimization version is a max-CSP problem where the aim is to minimize the number of constraint violations. The optimization version of the graph coloring problem can be generalized in many ways, for example, by assigning different weights to violated constraints or by giving different values to conflict violations based on the color that causes the conflict (e.g., a penalty of 1 if both nodes are blue and a penalty of 10 if both nodes are red).

4 Solution Techniques: Complete Algorithms

Given the previous description of a DCOP, we now focus on complete solution techniques, i.e., those that always find a solution that optimizes the global objective function. These techniques are particularly interesting and elegant from a theoretical point of view, but since we are dealing with an NP-hard problem they also exhibit an exponentially increasing coordination overhead (either in the size and/or number of messages exchanged, or in the computation required by each agent) as the number of agents in the system increases.

Broadly, these complete approaches can be divided in two classes: those that are search-based [7, 10, 15, 17, 25], and those that exploit dynamic program-

³For example see the repository of shared DCOP benchmarking problems created and maintained by the Teamcore research group, available at <http://teamcore.usc.edu/dcop/>.

ming [30]. Moreover, search-based approaches can be further divided between synchronous ones, such as SyncBB [17] and AND/OR search [10],⁴ and asynchronous ones, such as ADOPT [25], NCBB [7], and AFB [15]. In the synchronous execution model, agents wait for messages from other agents before computing and sending out new messages themselves. In contrast, in the asynchronous execution model, agents perform computation and send out messages without waiting for messages from their neighbors. Asynchronous operation is desirable in a multiagent context as it allows agents to make decisions without waiting for other agents to complete their computation, thus fully exploiting parallel computation. On the other hand, the synchronous model ensures that agents always have the most relevant and up-to-date information before executing their computation, thus minimizing redundancy in both computation and communication.

All the above techniques are completely decentralized, in the sense that each agent has complete control over its variables and is aware of only local constraints. However, centralizing part of the problem can sometimes reduce the effort required to find a globally optimal solution. This is the key concept behind the optimal asynchronous partial overlay (optAPO) approach [23]. In more detail, optAPO aims to discover parts of the problem that are particularly hard to solve in a decentralized fashion (parts that are strongly interconnected) and centralizes them into subproblems that are delegated to mediator agents (which act as centralized solvers). OptAPO has been shown to consistently reduce the communication overhead with respect to other decentralized techniques such as ADOPT. However, it is very hard to control how much of the problem will be centralized, and thus, it is difficult to predict the computational effort that mediator agents must be able to sustain.

Here we focus on two decentralized approaches that are good representatives of the two general solution classes: ADOPT for search-based techniques and DPOP for dynamic programming.

4.1 Search-Based: ADOPT

ADOPT (Asynchronous Distributed OPTimization) was proposed by Modi et al. and both guarantees solution optimality and allows agents to asynchronously change the values of the variables that they control [24, 25]. As is common in the DCOP literature, the original ADOPT formulation assumes that each agent controls only one variable and that the constraints are binary. Although there are ways to remove these assumptions without significantly changing the algorithm, most

⁴AND/OR search was originally developed for centralized optimization problems but can easily be extended to work in decentralized settings (see [28]).

of the work related to the ADOPT technique falls within this setting; therefore in what follows we embrace these assumptions. Moreover, to maintain a close correspondence with the original ADOPT description we assume that our task here is a minimization problem. Hence constraints represent costs and agents wish to find an assignment that minimizes the sum of these costs.

ADOPT is a search-based technique that performs a distributed backtrack search using a best-first strategy; each agent always assigns to its variable the best value based on local information. The key components of the ADOPT algorithm are: (i) local lower-bound estimates, (ii) backtrack thresholds, and (iii) termination conditions. In particular, each agent maintains a lower-bound estimate for each possible value of its variable. This lower bound is initially computed based only on the local cost function, and is then refined as more information is passed between the agents. Each agent will choose the value of its variable that minimizes this lower bound, and this decision is made asynchronously as soon as the local lower bound is updated.

Backtrack thresholds are used to speed up the search of previously explored solutions. This can happen because the search strategy is based on local lower bounds, and thus agents can abandon values before they are proven to be suboptimal. Backtrack thresholds are lower bounds that have been previously determined and can prevent agents from exploring useless branches of the search tree.

Finally, ADOPT uses a bound interval to evaluate the search progress. Specifically, each agent maintains not only a lower bound but also an upper bound on the optimal solution. Therefore, when these two values agree, the search process can terminate and the current solution can be returned. In addition, this feature can be used to look for solutions that are suboptimal but within a given predefined bound of the optimal solution. The user can specify a valid error bound (i.e., the distance between the optimal solution value and an acceptable suboptimal solution) and as soon as the bound interval becomes less than this value the search process can be stopped.

Before executing the ADOPT algorithm, agents must be arranged in a depth-first search (DFS) tree. A DFS tree order is defined by considering direct parent/child relationships between agents. DFS tree orderings have been frequently used in optimization (see for example [30]) because they have two interesting properties: (i) agents in different branches of the tree do not share any constraints, and (ii) every constraint network can be ordered in a DFS tree and this can be done in polynomial time with a distributed procedure [28]. The fact that agents in different branches do not share constraints is an important property as it ensures that they can search for solutions independently of each other. Figure 12.1 shows an exemplar constraint network. Figure 12.2 reports a possible DFS order, where solid lines show parent/child relationships and constraints are not repre-

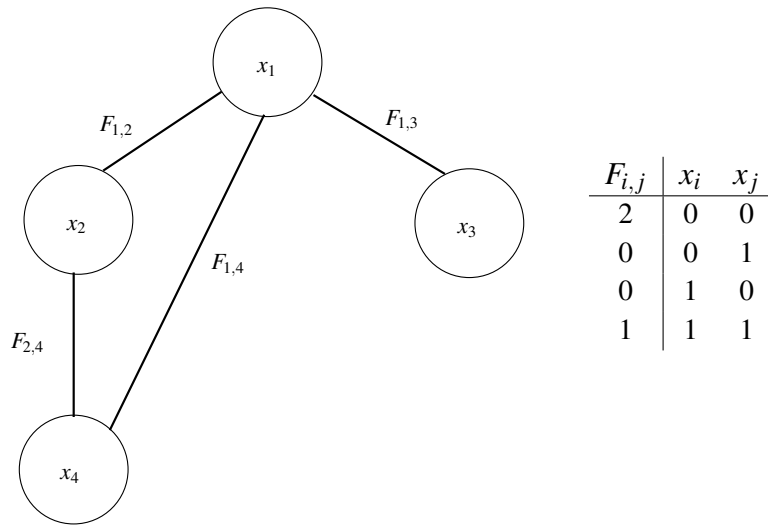


Figure 12.1: Exemplar constraint network.

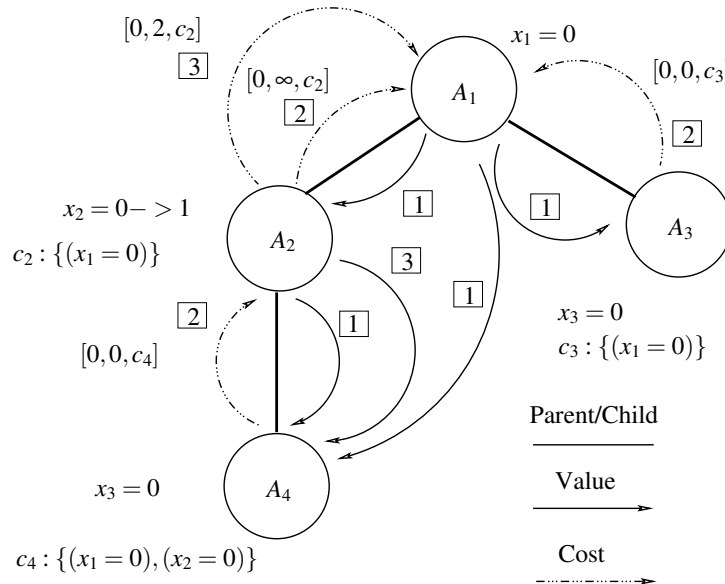


Figure 12.2: Message exchange in the ADOPT algorithm: *Value* and *Cost* messages for one possible trace of execution. Numbers within squares indicate the (partial) order of the messages.

sented. Given a constraint network, the DFS ordering is not unique and ADOPT's performance (in terms of coordination overhead) depends on the actual DFS or-

dering used. Finding the optimal DFS tree is a challenging problem, which the ADOPT technique does not address.

Given a DFS ordering of the agents, the algorithm proceeds by exchanging three types of messages: *Value*, *Cost*, and *Threshold*. When the algorithm starts, all agents choose a random value for their variables, and initialize the lower bound and upper bound of their variables' possible values to zero and infinity, respectively. These bounds are then iteratively refined as more information is transmitted across the network. Figure 12.2 reports messages exchanged among the agents during the first stages of the algorithm. Since the algorithm is asynchronous, we report here one possible trace of execution, and numbers within squares indicate the (partial) order of the messages.

In more detail, *Value* messages are sent by an agent to all its neighbors that are lower in the DFS tree order than itself, reporting the value that the agent has assigned to its variable. For example, in Figure 12.2 agent A_1 sends three value messages to A_2 , A_3 , and A_4 , informing them that its current value is 0. Notice that this message is sent to A_4 even though there is no parent/child relationship between A_1 and A_4 because A_4 is a neighbor of A_1 , who is lower in the DFS order.

Cost messages are sent by an agent to its parent, reporting the minimum lower and upper bound across all the agent's variable values, and the current context. The current context is a partial variable assignment, and, in particular, it records the assignment of all higher neighbors. For example, in Figure 12.2 the current context for A_4 , c_4 , is $\{(x_1, 0), (x_2, 0)\}$. The minimum lower bound and minimum upper bound are computed with respect to the current context. To compute the minimum lower bound each agent evaluates its own local cost for each possible value of its variable, adding all the lower-bound messages received from children that are compatible with the current variable value. The local cost for an agent is the sum of the values of local cost function for all the higher neighbors. For example, consider the cost message sent by A_4 . The minimum lower bound (which is 0) is computed by finding the minimum between $\delta(x_4 = 0) = 4$ and $\delta(x_4 = 1) = 0$, where $\delta(a)$ is the local cost function when the variable assumes the value a . The local cost function is computed by summing up the values of the cost functions for all neighbors higher in the DFS order and by assigning their values according to the current context. A similar computation is performed for the upper bound.

Cost messages for agents that are not leaves of the DFS tree (e.g., A_2) also include the lower and upper bound for each child. For example consider the cost message sent by A_2 to A_1 and let **LB** represent the minimum lower bound across all variables' values. **LB** is then computed by finding the minimum between $LB(x_2 = 0) = \delta(x_2 = 0) + lb(x_2 = 0, x_4) = 2$ and $LB(x_2 = 1) = \delta(x_2 = 1) + lb(x_2 = 1, x_4) = 0$, resulting in **LB** = 0. Here $lb(a, x_i)$ is the lower bound for the child variable x_i when the current variable is assigned to a in the current context.

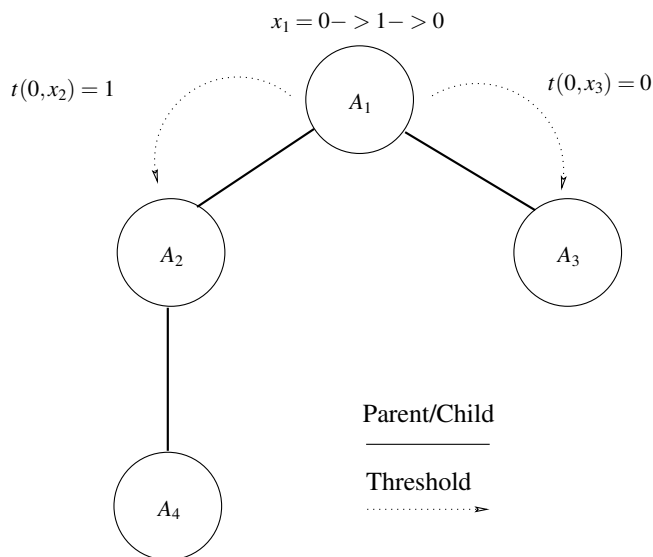


Figure 12.3: Message exchange in the ADOPT algorithm: *Threshold* messages for a revisited context.

Threshold messages are sent from parents to children to update the agent's backtrack thresholds. Backtrack thresholds are particularly useful when a previously visited context is revisited. Each agent stores cost information (e.g., upper and lower bounds) only for the current context and deletes previously stored information as soon as the context changes. In fact, if an agent did maintain such information for every visited context, it will need an exponential space in memory. However, since a context might be visited multiple times during the search process, whenever this happens the agent starts computing cost information from scratch. Now, since this context was visited before, the agent reported the sum of cost information to its parent and since the parent has that information stored, it can now send it back to the agent via a threshold message. The threshold message is used to set the agent's threshold to a previous valid lower bound, and propagate cost information down the tree, avoiding needless computation. Notice that the information that the parent stores is the accumulated cost information. Therefore, to propagate information down the tree, the agent must subdivide this accumulated cost across its children using some heuristic, as the original cost subdivision is lost, and then correct this subdivision over time as cost feedback is received from the children. For example, assume that during the search process, x_1 changes its value and then the context with $x_1 = 0$ is visited again. Agent A_1 will then send threshold messages to A_2 and A_3 as shown in Figure 12.3. Notice that the value of these messages is the lower-bound value sent by the corresponding child agent

for that context, e.g. the message $t(x_1 = 0, x_2)$ equals the lower bound sent by A_2 to A_1 with context $\{(x_1, 0)\}$.

Finally, agents asynchronously update a variable's value whenever the stored lower bound for the current value exceeds the backtrack threshold and the new variable's value is the one that minimizes the stored lower bounds. For example, consider agent A_2 in Figure 12.2, when receiving the cost message from A_1 . In this case, the lower bound for the current value ($LB(x_2 = 0) = 2$) exceeds the threshold (initially set to 0). Therefore, the agent updates its variable value to the one that minimizes the lower bound, which in our case is $x_2 = 1$. It then sends cost and value messages accordingly. When the minimum lower bound for a variable value is also an upper bound for that value, the agent can stop propagating messages as that value will be optimal given the current context. When this condition is true at the root agent, a terminate message is sent to all the children. Agents propagate the termination message if the termination condition is true for them as well. When the terminate message has propagated to all the agents, the algorithm stops, and the optimal solution has been found.

ADOPT is particularly interesting because it is asynchronous and because the memory usage of each agent is polynomial in the number of variables. Moreover, messages are all of a fixed size. However, the number of messages that agents need to exchange is, in the worst case, exponential in the number of variables. This impacts on the time required to find the optimal solution. In particular, the number of message synchronization cycles, defined as all agents receiving incoming messages and sending outgoing messages simultaneously, is exponential in the number of variables. This is a frequently used measure to evaluate DCOPs solution techniques as it is less sensitive to variations in agents' computation speed and communication delays than the wall clock. As previously remarked, such exponential elements are unavoidable in complete approaches and they can severely restrict the scalability of the approach.

Several works build on ADOPT attempting to reduce computation time. For example, Yeoh et al. propose BnB-ADOPT, which is an extension of ADOPT that consistently reduces computation time by using a different search strategy: depth-first search with branch and bound instead of best-first search [39]. Moreover, Ali et al. suggest the use of preprocessing techniques for guiding ADOPT search and show that this can result in a consistent increase in performance [3]. Finally, Gutierrez and Meseguer show that many messages that are sent by BnB-ADOPT are in fact redundant and most of them can be removed, resulting in significant reduction in communication costs [16].

In the next section we describe a completely different approach based on dynamic programming.

4.2 Dynamic Programming: DPOP

DPOP (dynamic programming optimization protocol) was proposed by Petcu and Falting [30], and is based on the dynamic programming paradigm, and more specifically, on the bucket elimination (BE) algorithm [9].

In more detail, BE is a dynamic programming approach for solving both constraint networks and also more general graphical models such as Bayesian networks, Markov random fields, and influence diagrams. BE takes as input a constraint network and an ordering of the network variables. It then associates a bucket to each variable and partitions the constraints, assigning them to the bucket following the given ordering. The optimal assignment is obtained by running two phases. First, the buckets are processed (from last to first), essentially running a variable elimination algorithm. Specifically, when processing a bucket, all constraints in the bucket are summed together and the variable that corresponds to the bucket is eliminated by maximization. This results in a new constraint that is placed in the first bucket, following the specified order, that contains one of the variables that are in the constraint scope. When the first phase is completed, the optimal value for the variable associated with the first bucket can be computed. This optimal value can be fixed, and then, given this value, the optimal value for the next variable in the ordering can be found. Proceeding in this way the entire optimal assignment is generated.

Now, although BE is normally defined over a linear ordering of the variables, it can be extended to work on a tree ordering via message-passing between the nodes, resulting in the bucket tree elimination algorithm (BTE) [9]. Against this background, DPOP can essentially be seen as a special case of BTE that operates on a DFS tree ordering of the constraint network. This specific arrangement is important because it ensures that during the optimization process, agents have knowledge of, and can control, only their own variables, and that they communicate only with other agents that share at least one constraint.

The DPOP algorithm can be divided into three phases: (i) arrangement of the variables into a DFS tree; (ii) propagation of *Util* messages bottom-up along the DFS tree (from leaves to root); and (iii) propagation of *Value* messages top-down (from root to leaves). We will briefly discuss these three phases in the following. As with the ADOPT example, for ease of presentation, we assume that each agent controls only one variable and that constraints are binary. However, relaxing this assumption does not result in significant changes to the algorithm. In contrast to the description of ADOPT, and in line with the original description of DPOP, we deal here with maximization problems.

As with ADOPT, DPOP first preprocesses the constraint network to create a DFS tree. In contrast to ADOPT, however, DPOP guarantees that the optimal solution can be obtained with a linear number of messages, resulting in messages

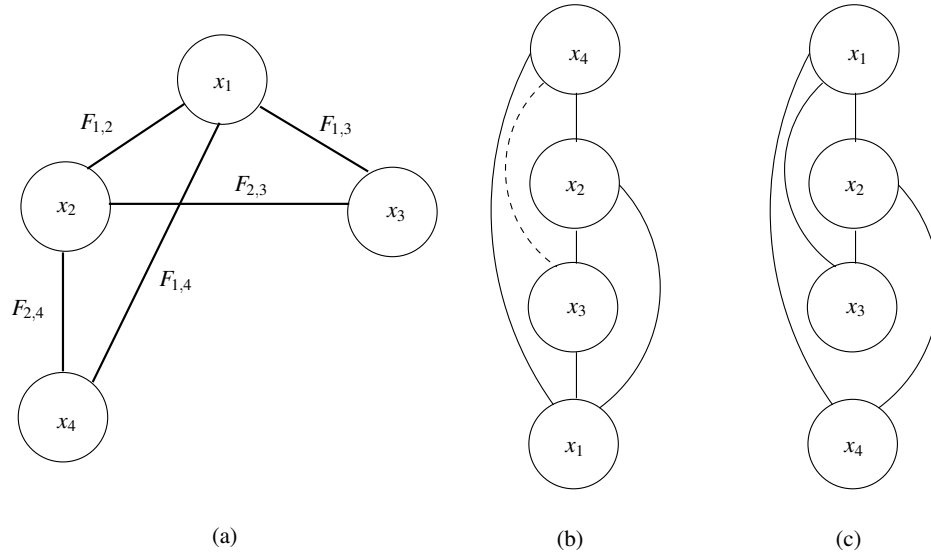


Figure 12.4: (a) Exemplar constraint network, with (b) induced graph with DFS order $\{x_4, x_2, x_3, x_1\}$, and (c) induced graph with DFS order $\{x_1, x_2, x_3, x_4\}$.

whose size is exponential in the induced width of the DFS tree ordering.

More specifically, DPOP can operate on a pseudo-tree ordering of the constraint network. A pseudo-tree ordering is one where nodes that share a constraint fall in the same branch of the tree. DFS tree ordering is thus a special case of a pseudo-tree that can be easily obtained with a DFS traversal of the original graph. Now, the complexity of the DPOP algorithm is strongly related to the DFS arrangement on which the algorithm is run, and, in particular, it is exponential in the *induced width* of the DFS tree ordering. Given a graph and an ordering of its nodes, the width induced by the ordering is the maximum induced width of a node, which is simply given by how many parents it has in the induced graph. The induced graph can be computed by processing the nodes from last to first, and for each node, adding edges to connect all the parents of that node (i.e., neighbors that precede the node in the order).

In particular, Figure 12.4 shows a constraint network and two induced graphs given by different orderings. The induced width for the graph in Figure 12.4(b) is 3, while the induced width for the graph in Figure 12.4(c) is 2. Notice the dashed edge between x_3 and x_4 in Figure 12.4(b) that was added when building the induced graph. While there are various heuristics to generate DFS orderings with small induced width, finding the one with the minimum induced width is an NP-hard problem. Figure 12.5 reports a DFS arrangement for the constraint network shown in Figure 12.4(a), along with messages that will be exchanged during the following phases. Dashed edges represent constraints that are part of

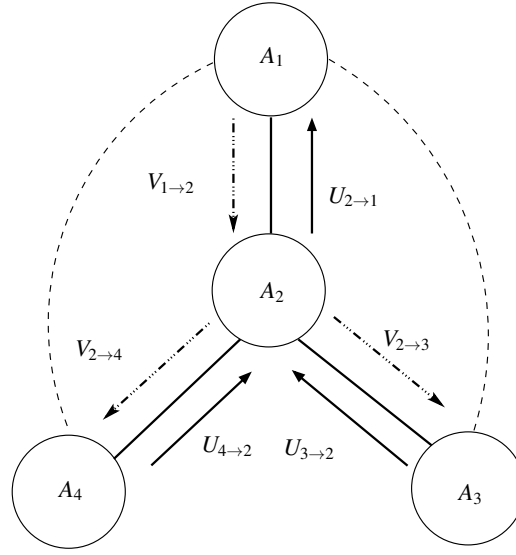


Figure 12.5: Message exchange in DPOP.

the constraint network but are not part of the DFS tree. These are usually called back-edges.

Once the variables have been arranged in a DFS tree structure, the *Util* propagation phase starts. *Util* propagation goes from leaves, up the tree, to the root node. Each agent computes messages for its parent considering both the messages received from its children and the constraints that the agent is involved in. In general, the *Util* message $U_{i \rightarrow j}$ that agent A_i sends to its parent A_j can be computed according to the following equation:

$$U_{i \rightarrow j}(Sep_i) = \max_{x_i} \left(\bigoplus_{A_k \in C_i} U_{k \rightarrow i} \oplus \bigoplus_{A_p \in P_i \cup PP_i} F_{i,p} \right) \quad (12.2)$$

where C_i is the set of children for agent A_i , P_i is the parent of A_i , PP_i is the set of agents preceding A_i in the pseudo-tree order that are connected to A_i through a back-edge (pseudo-parents), and Sep_i is the set of agents preceding A_i in the pseudo-tree order that are connected with A_i or with a descendant of A_i .⁵ The \oplus operator is a join operator that sums up functions with different but overlapping scopes consistently, i.e., summing the values of the functions for assignments that agree on the shared variables. For example, considering again Figure 12.5, agent A_3 sends the message $U_{3 \rightarrow 2}(x_1, x_2) = \max_{x_3} (F_{1,3}(x_1, x_3) \oplus F_{2,3}(x_2, x_3))$ because there are no messages from its children, while agent A_2 sends the message

⁵This set is called the separator because it is precisely the set of agents that should be removed to completely separate the subtree rooted at A_i from the rest of the network.

$U_{2 \rightarrow 1}(x_1) = \max_{x_2} (U_{3 \rightarrow 2}(x_1, x_2) \oplus U_{4 \rightarrow 2}(x_1, x_2) \oplus F_{1,2}(x_1, x_2))$. It is possible to show that the size of the largest separator in a DFS tree equals the induced width of the tree, which clarifies the exponential dependence on the induced width of message size.

Finally, the *Value* message propagation phase builds the optimal assignment proceeding from root to leaves. Root agent A_r computes x_r^* , which is the argument that maximizes the sum of the messages received by all its children (plus all unary relations it is involved in) and sends a message $V_{r \rightarrow c} = \{x_r = x_r^*\}$ containing this value to all its children $A_c \in C_r$. The generic agent A_i computes $x_i^* = \arg \max_{x_i} (\sum_{A_j \in C_i} U_{j \rightarrow i}(\mathbf{x}_p^*) + \sum_{A_j \in P_i \cup PP_i} F_{i,j}(x_i, x_j^*))$, where $\mathbf{x}_p^* = \bigcup_{A_j \in P_i \cup PP_i} \{x_j^*\}$ is the set of optimal values for A_i 's parent and pseudo-parents received from A_i 's parent. Finally, the generic agent A_i sends a message to each child A_j with value $V_{i \rightarrow j} = \{x_i = x_i^*\} \cup \bigcup_{x_s \in \text{Sep}_i \cap \text{Sep}_j} \{x_s = x_s^*\}$. For example, assume agent A_1 's optimal value is $x_1^* = 1$, then agent A_2 computes $x_2^* = \arg \max_{x_2} (U_{3 \rightarrow 2}(1, x_2) \oplus U_{4 \rightarrow 2}(1, x_2) \oplus F_{1,2}(1, x_2))$ and propagates the message $\{(x_1 = 1), (x_2 = x_2^*)\}$ to agents A_3 and A_4 . Notice that the maximization performed by agent A_4 in the value propagation phase is the same as the one previously done to compute the *Util* messages, but now with the aim of finding the value that maximizes the equation. Hence computation can be reduced by storing the appropriate values during the *Util* propagation phase.

As discussed above, DPOP message size, and hence the computation that agents need to compute it, is exponential. However, it is only exponential in the induced width of the DFS tree ordering used, which, in general, is much less than the total number of variables. Furthermore, there are many extensions of DPOP that address various possible trade-offs in the approach. In particular, MB-DPOP exploits the cycle-cut set idea to address the trade-off between the number of messages used and the amount of memory that each message requires [31]. On the other hand, A-DPOP addresses the trade-off between message size and solution quality [29]. Specifically, A-DPOP attempts to reduce message size by optimally computing only a part of the messages and approximating the rest (with upper and lower bounds). Given a fixed approximation ratio, A-DPOP can then reduce message size to meet this ratio, or, alternatively, given a fixed maximum message size, it propagates only those messages that do not exceed that size.

As a final remark, note that there is a close relationship between DPOP and the generalized distributive law (GDL) framework, which we shall discuss further in Section 5.2 [2]. GDL represents a family of techniques frequently used in information theory for decoding error correcting codes⁶ [21], and solving graphical

⁶Decoding turbo codes is probably the most important representative application for which GDL techniques are used. See [21], Section 48.4, for details.

models (e.g., to find the maximum *a posteriori* assignment in Markov random fields [38], or the posterior probabilities [37]).

5 Solution Techniques: Approximate Algorithms

As discussed earlier, solving a constraint network is an NP-hard problem. Therefore the worst-case complexity of complete methods are often prohibitive for practical applications. This is particularly the case for applications involving physical devices, such as sensor networks or mobile robots, which have severe constraints on memory and computation.

In these settings, approximate algorithms are often preferred, as they require very little local computation and communication, and are, as such, well suited for large-scale practical distributed applications in which the optimality of the solution can be sacrificed in favor of computational and communication efficiency (see [6] for a review of such algorithms). Furthermore, such approximate techniques have been shown to provide solutions that are very close to optimality in several problem instances [12, 22]. However, these approaches do not provide guarantees on the solution quality in general settings. This is particularly troublesome because the quality of solution to which most approximate algorithms converge is highly dependent on many factors that cannot always be properly assessed before deploying the system. Therefore there is no guarantee against particularly adverse behavior in specific pathological instances.

Thus, we next describe two classes of approximate algorithms for addressing DCOPs: local greedy methods and GDL-based approaches.

5.1 Local Greedy Approximate Algorithms

A local greedy search starts with a random assignment for all the variables and then performs a series of local moves trying to greedily optimize the objective function. A local move usually involves changing the value of a small set of variables (in most cases just one) so that the difference between the value of the objective function with the new assignment and the previous value is maximized. This difference is usually called the *gain*. The search stops when there is no local move that provides a positive gain, i.e., when the process reaches a *local* maximum. Local greedy search is a very popular approximate optimization technique, as it requires very little memory and computation, and can obtain extremely good solutions in many settings. The main problem for this type of approach is the presence of *local* maxima that can, in general, be arbitrarily far from the global optimal solution. Many heuristics can be used to avoid local maxima such as using random restart (sometimes called stochastic local search [9]) or introducing

stochastic steps in the search process (resulting in algorithms such as walkSAT and simulated annealing [9]).

Greedy local search methods have been widely used for DCOPs resulting in many successful approaches [12, 22]. When operating in a decentralized context, an important issue for these techniques is that to execute a greedy local move, agents need some type of coordination. In fact, the gain for a local move involving a variable x_i is computed assuming that all other variables $X \setminus \{x_i\}$ do not change their values. If all agents are allowed to execute in parallel, a potentially greedy move can become harmful (i.e., result in negative gain) because each agent has out-of-date knowledge about the choices of other agents. Such incoherence may also compromise the convergence of the approach leading to thrashing behaviors.

5.1.1 The Distributed Stochastic Algorithm

A simple and effective way to reduce such incoherence is to introduce a stochastic decision on whether agents should actually perform a move when they see the opportunity to optimize the gain [12]. This approach is usually called the distributed stochastic algorithm (DSA) and has been widely studied and applied in many domains. In more detail, assuming a synchronous execution model (each agent waits for the messages from all its neighbors), DSA has an initialization phase, where each agent chooses a random value for its variable, and then an infinite loop is executed by each agent. At each execution step, each agent A_i executes the following operations:

- Choose an activation probability $p_i \in [0, 1]$.
- Generate a random number $r_i \in [0, 1]$.
- If $r_i < p_i$ choose a value a_i such that the local gain is maximized.
- If $r_i \geq p_i$ do not change its value.
- If the variable value changed, send information to all neighbors notifying them of the change.
- Receive messages from neighbors and update information accordingly.

DSA can also be used in an asynchronous context and empirical results show that the algorithm is still effective if the rate of variable change is low with respect to the communication latency, thus allowing information to be propagated coherently in the system. Moreover, in most work, the activation probability is not decided at each optimization step, but is fixed at the beginning of the execution and is the same for all agents. The main strength of the DSA algorithm is its extremely low

overhead in terms of memory, computation, and communication. In fact, each agent needs to store and reason only about its direct neighbors, which, in general, are far fewer than the total number of agents in the system. Moreover, there is no exponential increase in computation and communication, as the optimization step considers only the current values of neighbors, and the communication step involves a message to communicate just the new value of the agent's variable. Finally, empirical results show that the algorithm typically monotonically increases the solution quality with each execution step, resulting in anytime behavior that is very well suited for practical applications; a solution is always available and the longer an agent waits before acting, the better the solution will be. Note, however, that there is no theoretical guarantee of such anytime behavior, but this is often obtained in practice with a suitable tuning of the activation probability.

The main drawback of DSA is that the solution quality can be strongly dependent on the activation probability, and there is no way to compute its value from an analysis of the problem instance. Moreover, the sensitivity of the algorithm's performance, with respect to the activation probability, is domain dependent, and it is hard to generalize the behavior of the algorithm across different domains.

5.1.2 The Maximum Gain Message Algorithm

An alternative approach to address the possible out-of-date knowledge about other agents' variable values is for neighboring agents to agree on who the agent is that can perform a move, while the others wait without changing their values. This approach is the basic idea behind the maximum gain message algorithm (MGM) [22]. MGM is based on the well-known distributed breakout algorithm (DBA), but is adapted to avoid outdated knowledge of the agent about its neighbors. Assuming again a synchronous execution model, at each execution step each agent A_i executes the following operations:

- Send its current value a_i to neighbors and receive values from neighbors.
- Choose a value a_i^* such that the local gain g_i^* is maximized (assuming neighbors do not change value).
- Send the gain g_i^* to neighbors and receive gain from neighbors.
- If the gain for the agent is the highest in the neighborhood, update the value of x_i to a_i^* .

MGM is guaranteed to be anytime and several empirical evaluations show that it has comparable performance with respect to DSA in various domains [22]. Moreover, unlike DSA, MGM does not require any parameter tuning.

A common characteristic of both DSA and MGM is that decisions are made considering local information only, i.e., when deciding the next value for its variable each agent optimizes only with respect to the current assignments of its neighbors. This provides for extremely low cost and scalable techniques; however, solution quality is strongly compromised by local maxima, which can, in general, be arbitrarily far from the optimal solution. In the next section, we present an algorithm for solving DCOP, based on the generalized distributive law (GDL) framework, that overcomes this limitation.

5.2 GDL-Based Approximate Algorithms

As previously mentioned, the GDL is a unifying framework to perform inference in graphical models. Specifically, the GDL framework operates on commutative semi-rings, and depending on the specific semi-ring used, we obtain different algorithms such as the max-sum, max-product, or sum-product. Such algorithms are widely used to perform inference tasks such as finding the maximum *a posteriori* assignment in Markov random fields (max-product) [38], or computing marginal distributions in Bayesian networks (sum-product or belief propagation) [21]. In particular, the max-sum algorithm can be used to solve constraint networks as it can find the argument that maximizes a global optimization function expressed as the sum of local functions.

5.2.1 The Max-Sum Algorithm

In more detail, the max-sum algorithm is an iterative message-passing algorithm, where agents continuously exchange messages to build a local function that depends only on the variables they control. This function represents the dependence of the global function on the agents' values and is used to find the optimal assignment. The max-sum algorithm can directly handle n -ary constraints and more variables per agent, however for ease of presentation, we report here a description of the algorithm in line with the earlier assumptions that each agent controls one variable and all constraints are binary. The interested reader can find the description of the algorithm in more general settings in [32]. Finally, we again assume a synchronous execution model.

Given this, at each execution step each agent A_i updates and sends to each of its neighbors A_j the message, $m_{i \rightarrow j}(x_j)$, given by:

$$m_{i \rightarrow j}(x_j) = \alpha_{ij} + \max_{x_i} \left(F_{ij}(x_i, x_j) + \sum_{k \in N(i) \setminus j} m_{k \rightarrow i}(x_i) \right) \quad (12.3)$$

where α_{ij} is a normalization constant, $N(i)$ is the set of indices for variables that are connected to x_i , and F_{ij} is the constraint defined over the variables controlled

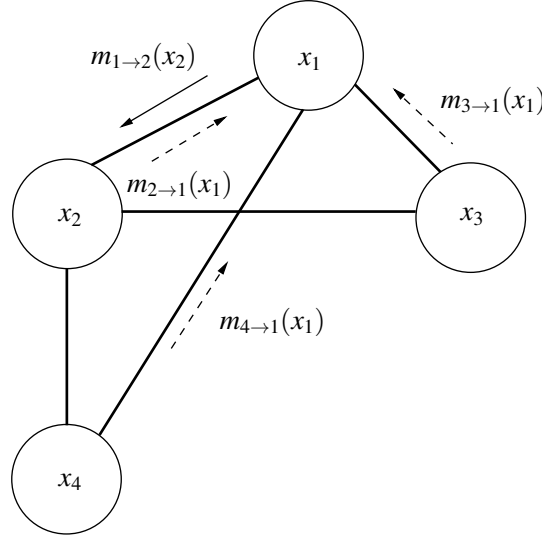


Figure 12.6: Message exchange in max-sum.

by A_i and A_j . The normalization constant α_{ij} is added to all the components of the message so that $\sum_{x_j} m_{i \rightarrow j}(x_j) = 0$. This is necessary on graphs with loops because otherwise message values might grow indefinitely, possibly leading to numerical errors.

At the first iteration all messages are initialized to constant functions, and at each subsequent iteration, each agent A_i aggregates all incoming messages and computes the local function, $z_i(x_i)$, which is given by:

$$z_i(x_i) = \sum_{k \in N(i)} m_{k \rightarrow i}(x_i) \quad (12.4)$$

This is then used to obtain the max-sum assignment, \tilde{x} , which, for every variable $x_i \in X$ is given by:

$$\tilde{x}_i = \arg \max_{x_i} z_i(x_i) \quad (12.5)$$

Figure 12.6 shows input and output messages for agent A_1 . In this example, the message to agent A_2 is computed as $m_{1 \rightarrow 2}(x_2) = \max_{x_1} (F_{1,2}(x_1, x_2) + m_{3 \rightarrow 1}(x_1) + m_{4 \rightarrow 1}(x_1))$ and $z_1(x_1) = m_{2 \rightarrow 1}(x_1) + m_{3 \rightarrow 1}(x_1) + m_{4 \rightarrow 1}(x_1)$.

The max-sum technique is guaranteed to solve the problem optimally on acyclic structures, but when applied to general graphs that contain loops, only limited theoretical results hold for solution quality and convergence. Nonetheless, extensive empirical evidence demonstrates that, despite the lack of convergence guarantees, the max-sum algorithm does in fact generate good approximate solutions when applied to cyclic graphs in various domains [11, 13, 19]. When the

algorithm does converge, it does not converge to a simple local maximum but rather to a neighborhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space [38]. Characterizing this region is an ongoing area of research and to date has only considered small graphs with specific topologies (e.g., several researchers have focused on the analysis of the algorithm's convergence in graphs containing just a single loop [1, 38]).

The max-sum algorithm is attractive for decentralized coordination of computationally and communication constrained devices since the messages are small (they scale with the domain of the variables), and the number of messages exchanged typically varies linearly with the number of agents within the system. Moreover, when constraints are binary, the computational complexity to update the messages and perform the optimization is polynomial. In the more general case of n -ary constraints, this complexity scales exponentially with just the number of variables on which each function depends (which is typically much less than the total number of variables in the system). However, as with the previously discussed approximate algorithms, the lack of guaranteed convergence and guaranteed solution quality in general cases limits the use of the standard max-sum algorithm in many application domains.

A possible solution to address this problem is to remove cycles from the constraint graph by arranging it into tree-like structures such as junction trees [20] or pseudo-trees [30]. However, such arrangements result in an exponential element in the computation of the solution or in the communication overhead. For example, DPOP is functionally equivalent to performing max-sum over a pseudo-tree formed by depth-first search of the constraint graph, and the resulting maximum message size is exponential with respect to the width of the pseudo-tree. This exponential element is unavoidable in order to guarantee optimality of the solution and is tied to the combinatorial nature of the optimization problem. However, as discussed in the introduction of this chapter, such exponential behavior is undesirable in systems composed of devices with constrained computational resources.

To address these issues, low overhead approximation algorithms that can provide quality guarantees on the solution are a key area of research, and we discuss the most prominent approaches in this area in the next section.

6 Solution Techniques with Quality Guarantees

Developing approximate algorithms that can provide guarantees on solution quality is a growing area of research that is gaining increasing attention. Such approaches are particularly promising as they can conveniently address the unavoidable trade-off between guarantees on solution quality and computation effort. Ad-

addressing this trade-off is particularly important in dynamic settings and when the agents have severe constraints on computational power, memory, or communication (which is usually the case for applications involving embedded devices, such as mobile robots or sensor networks). Moreover, having a bound on the quality of the provided solutions is particularly important for safety-critical applications (such as disaster response, surveillance, etc.) because pathological behavior of the system is, in this case, simply unacceptable.

Guarantees that can be provided by approximate algorithms can be broadly divided in two main categories: *off-line* and *online*. The former can provide a characterization of the solution quality without running any algorithm on the specific problem instances. In contrast, the latter can only provide quality guarantees for a solution after processing a specific problem instance. Off-line guarantees represent the classical definition of approximation algorithms [8], and they provide very general results not tied to specific problem instances. In this sense they are generally preferred to online guarantees. However, online guarantees are usually much tighter than off-line ones, precisely because they can exploit knowledge on the specific problem instance, and, thus, better characterize the bound on solution quality.

Here we present two representative approaches for these two classes: the k -optimality framework and the bounded max-sum approach.

6.1 Off-line Guarantees

A widely used approach to providing off-line guarantees for solution quality in DCOPs is based on the k -size optimality framework. The main idea behind k -size optimality is to consider optimal solutions for subgroups of k agents, and then provide a bound on the globally optimal solution. More specifically, a solution is k -optimal if the corresponding value of the objective function cannot be improved by changing the assignment of k or fewer agents. For example, consider again the constraint network in Figure 12.1. The solution $\hat{\mathbf{x}} = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$ is a 2-optimal solution but not a 3-optimal solution. In fact, $F(\hat{\mathbf{x}}) = F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} = 1 + 1 + 1 + 1 = 4$, and, thus, clearly if only two agents can change their variables' values, there is no solution that obtains a value higher than four. However, if we allow three agents to change their values, then we can obtain a better solution. Consider for example the assignment $\hat{\mathbf{x}}' = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0\}$; we can see that $F(\hat{\mathbf{x}}') = F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} = 2 + 0 + 2 + 2 = 6 \geq 4$. Moreover, notice that $\hat{\mathbf{x}}'$ is not the optimal solution, as the optimal solution is $\mathbf{x}^* = \{x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0\}$ and $F(\mathbf{x}^*) = 8$.

Building on the k -optimality concept, Pearce and Tambe were able to provide an approximation ratio (i.e., the ratio between the unknown optimal solution and the approximate solution [8]) for k -optimal algorithms that is valid for any DCOP

with non-negative reward structure⁷ [27]. The accuracy of such an approximation ratio, in any particular setting, depends on the number of agents, on the arity of the constraints, and on the value of k . Specifically, for any DCOP with non-negative rewards, the following equation holds:

$$F(\hat{\mathbf{x}}) \geq \frac{\binom{n-m}{k-m}}{\binom{n}{k} - \binom{n-m}{k}} F(\mathbf{x}^*) \quad (12.6)$$

where $\hat{\mathbf{x}}$ is a k -size optimal solution, \mathbf{x}^* is the optimal solution, n is the number of agents, and m is the maximum constraint arity and $m \leq k < n$. Notice that every DCOP that does not have infinite negative costs can be normalized to one with all non-negative rewards. However the analysis is not applicable to DCOPs that include hard constraints.

For the usual case of a binary network (i.e., $m = 2$) the above equation simplifies to:

$$F(\hat{\mathbf{x}}) \geq \frac{k-1}{2n-k-1} F(\mathbf{x}^*) \quad (12.7)$$

Thus, for the constraint network in Figure 12.1, we can immediately conclude that for any 2-optimal solution, $\hat{\mathbf{x}}$, we will have that $F(\hat{\mathbf{x}}) \geq 1/5 \cdot F(\mathbf{x}^*)$ simply because $\frac{k-1}{2n-k-1} = 1/5$ when $k = 2$ and $n = 4$. In fact it is easy to see that this inequality holds for the 2-optimal solution considered above.

Notice that the above inequalities hold for every possible constraint network with every possible reward structure (as long as all rewards are non-negative). For example, if we add a constraint between x_3 and x_4 in our exemplar constraint network, no matter what function we define for that constraint, we are still guaranteed that the value of any 2-optimal solution will be greater than 1/5 of the value of the optimal solution. This is clearly a very strong and general result, but unfortunately, the accuracy of this bound depends on the number of agents, on the arity of the constraints, and on the value of k . Specifically, the bound is more accurate when k is higher, but less accurate when the number of agents in the system grows, and the maximum arity of constraints is high. Clearly, by increasing k it is possible to achieve better bounds; however this would result in an exponential increase in computation and communication required to obtain a k -size optimal solution.

From an algorithmic point of view, the k -size optimal framework assumes that we are able to find a k -size optimal solution. This basically requires that a group of k agents coordinate their choice to find a solution that is optimal for the group. Now, any local hill climbing algorithm is k -size optimal for $k = 1$. Hence, approaches such as DSA [12] and MGM [22] are able to find a 1-optimal

⁷Reward structure here makes reference to the particular values that return the functions of the DCOP.

solution. However, for $k = 1$ we are unable to provide any guarantees as the k -optimal analysis is valid only if $m \leq k$. Therefore, $k = 2$ variants of MGM and DSA, termed MGM-2 and DSA-2, have been devised [22]. Moreover, there are also algorithms to find k -size optimal solutions with arbitrary k [18].

The k -size optimality framework has been recently extended introducing a different criterion for local optimality; in particular, t -distance optimality. This is based on the distance between nodes on the constraint graphs, rather than on the size of the groups. Furthermore, Vinyals et al. recently introduced the \mathcal{C} -optimality framework, which generalizes both k -optimality and t -optimality by providing quality guarantees for local optima in regions that can be defined by arbitrary criteria [36]. Specifically, they propose a new criterion to define regions (i.e., the size-bounded-distance criterion), showing that this criterion outperforms both k -size and t -distance, yielding more precise control of the computational effort required to provide such local optimal solutions. Finally, the \mathcal{C} -optimality framework has been recently used to provide quality guarantees for fixed-point assignments of the max-sum algorithm (i.e., assignments to which the algorithm converged). Specifically, building on the results obtained by Weiss that characterized any fixed-point max-product assignment as a local optima for a specific region (named single loops and trees) [38], Vinyals et al. were able to characterize the quality guarantees for the max-sum algorithm in that region [33]. Thus, if the max-sum algorithm converges, it is possible to provide a worst-case quality guarantee equivalent to 3-optimal solutions of the k -optimality framework.

As mentioned above, Equation 12.6 is valid for every possible constraint network. This is because the bound is the result of a worst-case analysis on a completely connected graph. If we restrict our attention to specific constraint network topologies it is possible to obtain better bounds. For example, for a network with a ring topology, where each agent has only two constraints, we have that $F(\hat{\mathbf{x}}) \geq \frac{k-1}{k+1} F(\mathbf{x}^*)$. This is a much better bound as it does not depend on the number of agents in the system, but applies only to a very specific topology. Similar considerations hold for assumptions about the reward structure. Specifically, better guarantees can be provided assuming some *a priori* knowledge on the reward structure. For example, Bowring et al. show that the approximation bounds can be improved by knowing the ratio between the minimum reward to the maximum reward [5]. In addition, they also extend the k -optimality analysis to DCOPs that include hard constraints. However, while they are able to significantly improve on the accuracy of the bound by exploiting *a priori* knowledge on the reward, the bound is still dependent on the number of agents, and decreases as the number of agents grows; thus, it is of little help for large-scale applications.

6.2 Online Guarantees

Online approaches for providing quality guarantees are complementary to off-line ones, as they usually give accurate bounds but only for specific problem instances [32, 34]. In this respect, the bounded max-sum (BMS) approach is a good representative for this kind of technique [32]. The main idea behind BMS is to remove cycles in the original constraint network by simply ignoring some of the dependencies between agents. It is then possible to optimally solve the resulting tree-structured constraint network, while simultaneously computing the approximation ratio for the original problem instance. The BMS approach uses the max-sum algorithm to provide an optimal solution for the tree-structured problem, hence the name; but any distributed constraint optimization technique that is guaranteed to provide optimal solutions on a tree-structured constraint network could be used. The choice of the max-sum approach is driven by its efficiency in terms of low communication overhead (specifically in the number of messages), low computational requirement, and ease of decentralization. The main issue with this approach is to choose which dependencies should be removed to form the tree-structured constraint network, and to somehow relate the solution quality of the new problem to that of the original loopy one.

To do this, BMS assigns weights to constraints, where the weights quantify the maximum impact that removing any constraint may have on the optimal solution. In other words, the weight of a constraint specifies the worst-case outcome if we were to solve the problem ignoring that constraint. BMS will then remove constraints that have the least impact on the solution quality (i.e., constraints that have smaller weights). This can be done by computing a maximum spanning tree of the original constraint network.

For ease of presentation, we again provide a description of the algorithm under the usual assumptions that each agent controls exactly one variable, and that all constraints are binary. The interested reader can find the description of the algorithm in more general settings in [32]. In this context, the steps of the algorithm are the following:

1. Define the weight of each constraint as: $w_{ij} = \min\{w'_{ij}, w''_{ij}\}$ where $w'_{ij} = \max_{x_j} [\max_{x_i} F_{ij} - \min_{x_i} F_{ij}]$ and $w''_{ij} = \max_{x_i} [\max_{x_j} F_{ij} - \min_{x_j} F_{ij}]$. For example, Figure 12.7 reports a constraint network and the weights that the BMS algorithm would compute. Specifically, for the constraint between x_1 and x_4 we have that $w'_{14} = \max_{x_4} [\max_{x_1} G_{14} - \min_{x_1} G_{14}] = 3$ and $w''_{14} = \max_{x_1} [\max_{x_4} G_{14} - \min_{x_4} G_{14}] = 1$ therefore $w_{14} = 1$.
2. Remove constraints from the original cyclic constraint network by building a maximum weight spanning tree. In this way it is possible to minimize

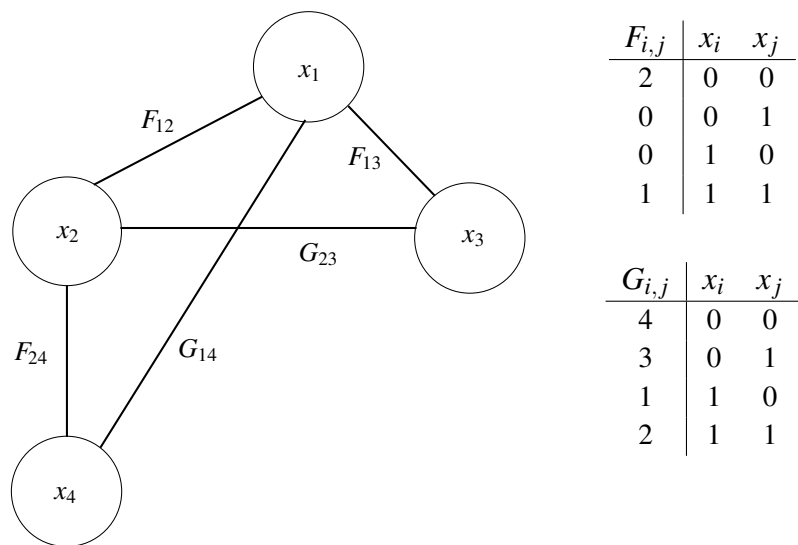


Figure 12.7: Loopy constraint network with two types of constraints.

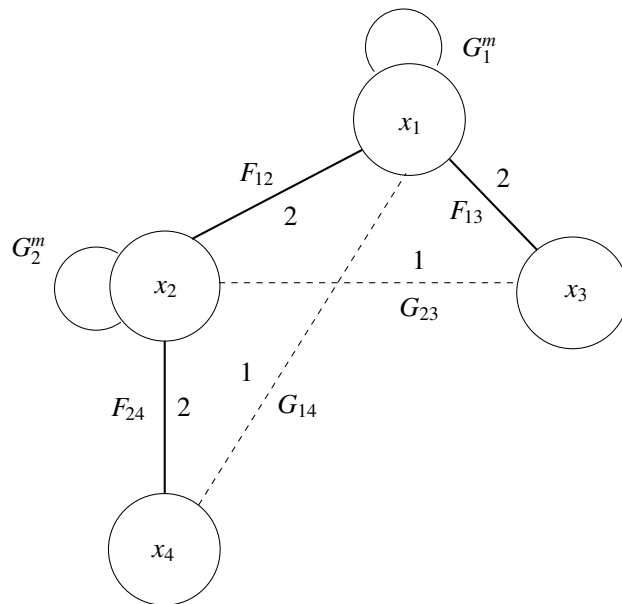


Figure 12.8: Tree-like constraint network formed by the BMS algorithm when run on the loopy constraint network of Figure 12.7. Numbers represent weights for binary constraints.

the sum of the weights of the removed edges, pursuing the objective of

removing constraints that have least impact on the solution.

Moreover, define the sum of the weights of the removed constraints as:

$$W = \sum_{c_{ij} \in C^r} w_{ij} \quad (12.8)$$

where C^r is the set of constraints removed from the constraint network.

For example, in Figure 12.7 we have that $C^r = \{G_{14}, G_{23}\}$ therefore $W = w_{14} + w_{23} = 2$.

3. Run the max-sum algorithm on the remaining tree-structured constraint network. For constraints that have been removed, add unary constraints obtained by minimizing out one of the two variables. The removed variable is the one that yields the minimum weight for that constraint. For example, in our case the assignment we obtain after running max-sum on the spanning tree solves the constraint network shown in Figure 12.8, thus optimizing the global function $G_1^m + F_{12} + F_{13} + G_2^m + F_{24}$ where $G_1^m = \min_{x_4} G_{14}$ and $G_2^m = \min_{x_3} G_{23}$.
4. The resulting variable assignment, $\tilde{\mathbf{x}}$, represents the approximate solution to the original optimization problem, and it is possible to prove that this approximate solution is within a calculated bound from the optimum solution. More precisely:

$$V^* \leq \rho \tilde{V} \quad (12.9)$$

where the approximation ratio $\rho = (\tilde{V}^m + W)/\tilde{V}$, and \tilde{V}^m represents the optimal solution to the tree-structured constraint network. Here V^* represents the value of the unknown optimal solution to the original cyclic constraint network and \tilde{V} is the approximate solution, found using the tree-structured constraint network, but evaluated on the original cyclic constraint network.

All the above steps can be performed using a decentralized approach. In particular, the computation of the maximum spanning tree can be performed in a distributed fashion using various message-passing algorithms, such as, for example, the minimum spanning tree algorithm by Gallager, Humblet and Spira (GHS), modified to find the maximum spanning tree [14].

As previously mentioned, this approach is able to provide guarantees on solution quality that are instance-based. Therefore the algorithm must be run on the specific problem instance in order to obtain the bound. By exploiting knowledge of the particular problem instance, BMS is able to provide bounds that are very accurate. For example, the BMS approach has been empirically evaluated in a mobile sensor domain, where mobile agents must coordinate their movements to

sample and predict the state of spatial phenomena (e.g., temperature or gas concentration). When applied in this domain, BMS is able to provide solutions that are guaranteed to be within 2% of the optimal solution.

Other data dependent approximation approaches with guarantees have also been investigated. For example, Petcu and Faltings propose an approximate version of DPOP [29], and Yeoh et al. provide a mechanism to trade off solution quality for computation time for the ADOPT and BnB-ADOPT algorithms [40]. Such mechanisms work by fixing an approximation ratio and reducing computation or communication overhead as much as possible to meet that ratio.

More specifically, BnB-ADOPT fixes a predetermined error bound for the optimal solution, and stops when a solution that meets this error bound is found. In this approach, the error bound is fixed and predetermined off-line, but the number of cycles required by the algorithm to converge is dependent on the particular problem instance, and, in the worst case, remains exponential. The BMS approach discussed above, in contrast, is guaranteed to converge after a polynomial number of cycles, but the approximation ratio is dependent on the particular problem instance.

Similar considerations hold with respect to A-DPOP [29]. A-DPOP attempts to reduce message size (which is exponential in the original DPOP algorithm in the width of the pseudo-tree) by optimally computing only a part of the messages, and approximating the rest (with upper and lower bounds). In this case, given a fixed predetermined approximation ratio, A-DPOP reduces message size to meet this ratio. Alternatively, given a fixed maximum message size, A-DPOP propagates only those messages that do not exceed that size. As a result of this, the computed solution is not optimal, but approximate. If the algorithm is used by fixing a desired approximation ratio, the message size remains exponential. In contrast, if we fix the maximum message size, the approximation ratio is dependent on the specific problem instance.

7 Conclusions

The constraint processing research area comprises powerful techniques and algorithms that are able to exploit problem structure, and, thus, solve hard problems efficiently. In this chapter we focused on the DCOP framework where constraint processing techniques are used to solve decision-making problems in MAS.

This chapter provides an overview of how DCOPs have been used to address decentralized decision making problems by first presenting the mathematical formulation of DCOPs and then describing some of the practical problems that DCOPs can successfully address. We detailed exact solution techniques for DCOPs presenting two of the most representative algorithms in the literature:

ADOPT and DPOP. We then discuss approximate algorithms, including DSA and MGM, before presenting GDL and the max-sum algorithm. Finally, we presented recent ongoing work that is attempting to provide quality guarantees for these approaches.

Overall, the DCOP framework, and the algorithms being developed to solve such problems, represent an active area of research within the MAS community, and one that is increasingly being applied within real-world contexts.

8 Exercises

1. **Level 1** Consider the coordination problem faced by intervention forces in a rescue scenario. In particular, consider a set of fire fighting units that must be assigned to a set of fires in order to minimize losses to buildings, infrastructure, and civilians. Each fire fighting unit can be assigned to just one fire. However, if more than one unit works on the same fire at the same time, they can extinguish it faster (collaboration has a positive synergy). Furthermore, a fire fighting unit can only be assigned to fires that are within a given distance from its initial position (due to the travel time required to reach the fire). As a result, any particular fire fighting unit can only be assigned to a subset of the fires that exist.
 - Formalize this task assignment problem as a DCOP specifying (i) what the variables represent, (ii) the domain of the variables, and (iii) the constraints.
 - Present an example involving about five fire fighting units and three fires, instantiating each of the features above.
2. **Level 2** Consider the coordination problem described in Exercise 1, but now extended to include two types of intervention forces: fire fighting units and ambulance units. Assume that instead of minimizing losses, we must now assign exactly one fire fighting unit and one ambulance unit to each fire. As before, any particular fire fighting or ambulance unit can only attend a subset of the existing fires. Provide a CSP formulation of this problem.
3. **Level 4** Consider the coordination problem described in Exercise 1, and specifically, a situation with two fire fighting units and two fires, where both units can be assigned both fires. Depending on the units' travel times, the severity of the fires, and the function that defines the losses that result, the best solution could result in both units being assigned to the same fire, leaving the other one uncontrolled. Assume now that we want to have a fair

assignment of fire fighting units to tasks. Can we formalize this as a DCOP? Which approach could be used to tackle this problem?

4. **Level 2** Consider the following constraint network representing a graph coloring problem:

- $X = \{x_1, \dots, x_6\}$
- $D = \{d_1 = d_3 = d_4 = d_6 = \{Red, Blue\}, d_2 = d_5 = \{Red, Blue, Green\}\}$
- $C = \{ \langle x_1, x_2 \rangle, \langle x_1, x_3 \rangle, \langle x_1, x_6 \rangle, \langle x_2, x_3 \rangle, \langle x_3, x_4 \rangle, \langle x_4, x_5 \rangle, \langle x_4, x_6 \rangle, \langle x_5, x_6 \rangle \}$

Assume that every node in the graph is controlled by one agent. Find the computational complexity of DPOP with the following pseudo-tree ordering: $o_1 = \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle$. State whether there is a pseudo-tree ordering that would result in less computational complexity and, if so, present an instance of one.

5. **Level 1** Show a complete execution of DPOP that solves the max-CSP formulation of the constraint network provided in Exercise 4 (use either o_1 or a pseudo-tree ordering of your choice).
6. **Level 1** Give an execution example of MGM where the algorithm finds the global optimum, and another where it gets trapped in a local minimum.
7. **Level 3** The max-sum algorithm is guaranteed to converge to the optimal solution on graphs that do not contain cycles. However, if there are several optimal assignments, the distributed maximization performed in Equation 12.5 is problematic, and may lead to suboptimal solutions. Discuss whether a value propagation phase would solve this problem. Provide an execution example.
8. **Level 4** The DPOP algorithm is based on a pseudo-tree arrangement of the agents. In particular, DFS trees that are a specific class of pseudo-tree are typically used. On the other hand, many constraint optimization approaches [9], including GDL-based techniques, are based on the concept of a junction tree [20]. Discuss the relationship between these two structures, and their impact with respect to computation and communication on DCOP solution techniques. Can you find a junction tree that allows a GDL algorithm to solve a DCOP with significantly less computation (and/or communication) than when using a pseudo-tree? What about DFS trees? We suggest the reader start from [35] where these questions were investigated for unrestricted pseudo-trees.

9. **Level 2** Provide a DCOP problem with at least five variables and a solution that is 3-optimal but not 4-optimal.
10. **Level 3** Consider the bound expressed by Equation 12.9 for the bounded max-sum approach. Assuming that you know the constraint network topology, and the maximum and minimum value for all the functions (but not the actual values of the functions) in a particular DCOP example, modify the bounded max-sum technique to provide a bound in this setting. Can you provide some bounds even before running the max-sum algorithm? Can you extend the analysis assuming you do not know the constraint network topology?
11. **Level 3** Consider the bounded max-sum technique presented in Section 6.2. Elaborate on how this technique can be applied to problems that include hard constraints. In particular, how is the approximation ratio affected by hard constraints? Under which assumptions can this technique still provide useful bounds?
12. **Level 4** Consider the problem of minimizing the running time of a DCOP solution algorithm when agents have heterogeneous computation and communication. In this setting it might be beneficial to delegate computation to agents that have additional computation capabilities, or to minimize message exchange between agents that are connected by poor communication links. Formalize this problem and provide an approach that can take such heterogeneity into account.

References

- [1] S. M. Aji, S. B. Horn, and Robert J. McEliece. On the convergence of iterative decoding on graphs with a single cycle. In *Proceedings of the International Symposium on Information Theory (ISIT)*, pages 276–282, 1998.
- [2] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.
- [3] S. M. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1041–1048, 2005.
- [4] Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. *Artif. Intell.*, 140(1/2):1–37, 2002.

- [5] E. Bowring, J. Pearce, C. Portway, M. Jain, and M. Tambe. On k -optimal distributed constraint optimization algorithms: New bounds and algorithms. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent systems*, pages 607–614, 2008.
- [6] A. Chapman, A. Rogers, N. R. Jennings, and D. Leslie. A unifying framework for iterative approximate best response algorithms for distributed constraint optimisation problems. *The Knowledge Engineering Review*, 26(4):411–444, 2011.
- [7] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1427–1429, 2006.
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.
- [9] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- [10] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171:73–106, 2007.
- [11] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.
- [12] S. Fitzpatrick and L. Meetrens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–293. Kluwer Academic, 2003.
- [13] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.
- [14] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.
- [15] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding for distributed COPs. *Journal Artificial Intelligence Research*, 34:61–88, 2009.
- [16] P. Gutierrez and P. Meseguer. Saving redundant messages in BnB-ADOPT. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1259–1260, 2010.
- [17] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

- [18] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, 2010.
- [19] R. J. Kok and N. Vlassis. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup-2005: Robot Soccer World Cup IX*, 2005.
- [20] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 42(2):498–519, 2001.
- [21] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
- [22] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems*, pages 432–439, 2004.
- [23] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 438–445, 2004.
- [24] P. J. Modi. *Distributed Constraint Optimization for Multiagent Systems*. PhD thesis, Department of Computer Science, University of Southern California, 2003.
- [25] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, (161):149–180, 2005.
- [26] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
- [27] J. P. Pearce and M. Tambe. Quality guarantees on k -optimal solutions for distributed constraint optimization problems. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1446–1451, 2007.
- [28] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), 2007.
- [29] A. Petcu and B. Faltings. A-DPOP: Approximations in distributed optimization. In *Principles and Practice of Constraint Programming*, pages 802–806, 2005.
- [30] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

- [31] A. Petcu and B. Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1452–1457, 2007.
- [32] A. Rogers, A. Farinelli, R. Stranders, and N. R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence Journal*, 175(2):730–759, 2011.
- [33] M. Vinyals, J. Cerquides, A. Farinelli, and J. A. Rodriguez-Aguilar. Worst-case bounds on the quality of max-product fixed-points. In *Neural Information Processing Systems*, pages 2325–2333, 2010.
- [34] M. Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, and J. Cerquides. Divide-and-coordinate: DCOPs by agreement. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 149–156, 2010.
- [35] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming DCOP algorithms via the Generalized Distributive Law. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, pages 1–26, 2010.
- [36] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and M. Bowring. Quality guarantees for region optimal DCOP algorithms. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, 2011.
- [37] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.
- [38] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):723–735, 2001.
- [39] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 591–598, 2008.
- [40] W. Yeoh, X. Sun, and S. Koenig. Trading off solution quality for faster computation in DCOP search algorithms. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pages 354–360, 2009.
- [41] Makoto Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer-Verlag, 2001.

Part V

Development and Engineering

Chapter 13

Programming Multiagent Systems

Rafael H. Bordini and Jürgen Dix

1 Introduction

The first two chapters of this book covered the general ideas of the agent computing paradigm, how it differs from traditional approaches such as object orientation, and what features are expected of multiagent organizations. Chapters 3–12 discussed other very important aspects of agents: communication, learning, planning, coalition formation, and many others. In this chapter, we consider the question of how to effectively *program multiagent systems*: what are the new techniques and principles suitable for multiagent systems as opposed to traditional software engineering methods?

All current trends in computing point toward a vision of the future whereby autonomous software will be needed everywhere, and required to work together with huge numbers of other autonomous software entities regardless of their locations. Traditional software engineering and programming languages were not built with this vision in mind and therefore lack the appropriate methods and tools to deal with these challenges.

Agent-oriented programming was first introduced by Shoham in 1993 (this is discussed below in greater detail). While the first decade saw mainly theoretical approaches, many of which were not yet mature enough for practical programming, the creation of the ProMAS and DALT workshop series (both held with AAMAS since 2003) and related activity helped to change the picture. With those efforts, there was a substantial increase in the numbers of researchers interested in doing research in multiagent programming over the years. ProMAS in par-

ticular was conceived at a Dagstuhl seminar (<http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=02481>), which helped to promote a fresh start for research in this area. As a consequence, a plethora of languages were devised; some of the most prominent appeared in two volumes of contributed chapters [9, 11]. Other Dagstuhl seminars specific to agent programming took place later (for example, <http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=08361>) and are still planned for the future.

Originally agent programming languages were mostly concerned with programming *individual* agents, and very little was available in terms of programming abstractions covering the social and environmental dimensions of multiagent systems as well as the agent dimension. This chapter will focus precisely on these relatively recent developments into multiagent-oriented programming (MAOP).

Both theoretically and practically, some languages are now arguably mature. Many current (multi-)agent programming languages have usable integrated development environments (IDEs) and debugging tools (in particular, tools to inspect the state of an agent or an organization), although much work is still needed in comparison to the best tools used for object-oriented programming. However, the main promise of multiagent orientation as a software development paradigm is in the features that the inspiration of reactive planning systems [31] and a societal view of computing have helped to create.

Some agent languages are based on computational logic (we refer to the CLIMA workshop series <http://centria.di.fct.unl.pt/~clima/> and the collections of papers from recent editions in [25, 26]). While there are similarities to traditional logic programming languages, agent languages provide a combination of theoretical reasoning (i.e., reasoning about beliefs) and practical reasoning (i.e., reasoning about the best action to take at each moment in time) whereas logic programming languages most often deal only with the former. The features of multiagent-oriented programming form a powerful alternative for the high-level conception and implementation of complex distributed autonomous systems.

The best way to find technical papers and currently ongoing research is to look at the proceedings of ProMAS [20], DALT [48], CLIMA, AAMAS (as well as the main AI conferences), LADS [22], and various other workshops. There have also been tutorials at the AAMAS conferences run by IFAAMAS (<http://www.aamas-conference.org/>); at EASSS, the European Agent Systems Summer School, now under the auspices of EURAMAS (<http://www.euramas.org/>); and recently at a DALT Spring School (http://lia.deis.unibo.it/confs/dalt_school/). The interested reader may also appreciate some of the survey papers published in the last few years [8, 23, 29, 45] as well as some journal special issues [10, 12].

1.1 Relation to Other Chapters

Although this chapter is self-contained and does not depend on the rest of the book, some knowledge of Chapter 2 (on organizations) helps to understand the *organizational* level of programming. An important part of this chapter is to provide the program code (in JACAMO, a combination of JASON, CARTAGO and MOISE) of the example given in Chapter 15 about an assembly cell of a manufacturing plant. Thus the reader can compare the agent-oriented software engineering approach to dealing with this example with our solution using a fully-fledged multiagent programming language. This scenario is also used as an example for model checking a multiagent system in Chapter 14.

1.2 Organization of This Chapter

This chapter is organized as follows. In Section 2, we briefly describe the progress from the beginning of agent-oriented programming toward modern agent languages in use today and their important features. Section 3 is devoted to the various levels of abstractions in agent programming: the level of individual agents, the shared environment, and the social level of a multiagent system.

Section 4 is the main part of the chapter. It describes a particular agent programming language, JASON, which is used throughout the chapter to illustrate the paradigm through several examples. We also mention, very briefly, other agent languages (mainly those based on the BDI paradigm and those based on executable logics) and give pointers and references. MAOP is not only about programming agents, it is also meant to deal with the environment and the social/organizational context – two other very important levels of programming in this paradigm. Therefore, we consider these two levels in more detail in Section 5.

Section 6 is the largest in this chapter. It is built around an example defined in Chapter 15 and gives a detailed solution of how to program this example in JASON. To be more precise, we use JACAMO, a combination of three platforms, JASON, CARTAGO, and MOISE to deal with the agent, the social, and the environmental levels.

Finally we conclude with an outlook on the future of multiagent programming and a collection of exercises of different levels of difficulty.

2 From AGENT0 to Modern Agent Languages

In this section we very briefly review the development of agent-oriented programming in the last two decades. While Subsection 2.1 gives a short historical account, Subsection 2.2 highlights the main features of multiagent-oriented programming as opposed to traditional software engineering.

2.1 A Brief History of Agent-Oriented Programming (AOP)

The seminal paper on *agent-oriented programming* (AOP) appeared in the early 1990s [59]. In that paper, Yoav Shoham advocated a new programming paradigm combining a societal view of computation with an “intentional stance” to support practical reasoning of the individual, autonomous components of the system. Although AGENT0, the programming language appearing in that paper, had many of the essential ideas for establishing such a new programming paradigm, it failed to show usefulness and popularity as a programming language, which in a way led some to question the whole paradigm. The work done throughout the 1990s was mostly theoretical without much interest, apart from the odd exceptions, in how the paradigm ought to be used concretely in practical programming. Also missing were serious considerations about the tools that are required for programming agent systems: Is the paradigm useful in practical software development for complex applications in dynamic, unpredictable environments where rational, social, and autonomous behavior is essential?

Apart from AGENT0, the work on programming languages for agents in the 1990s was dominated by Concurrent METATEM [27], Golog variants [32, 43, 57], AgentSpeak(L) [52], and 3APL [35]. All these languages had fundamental historical importance in an area of research that was to see a completely different picture in the naughts, as mentioned in Section 1.

While many approaches emphasized the idea of mental states, there were also approaches that stressed the importance of the action-selection mechanisms in multiagent systems (quite a few of them based on reactive planning): e.g., Müller’s InteRRaP architecture [46], Bryson’s POSH [49], Laird’s SOAR [42], and Subrahmanian et al.’s IMPACT [62].

Various technical issues have been tackled in the last decade. One example is the issue of *declarative goals*. In [18], Braubach and Pokhar studied a number of different types of goals that could be practically included in agent languages to facilitate the programming of rational autonomous behavior. Much work has been done on declarative goals since then, for example [67], as well as on the management of a goal “life-cycle” [63].

Another example of an important issue addressed in the last decade is that originally agent programming languages were mostly concerned with program-

ming *individual* agents, and very little was available in terms of programming abstractions covering the social and environmental dimensions of a multiagent system as well as agents individually. Important work in that direction is [24, 37], and we will discuss this progression throughout this chapter.

2.2 Features of Multiagent-Oriented Programming (MAOP)

The importance of the multiagent-oriented programming paradigm is that by getting inspiration in the areas of multiagent systems and artificial intelligence, a concrete programming paradigm has emerged that is specially suited for the development of highly social, autonomous software, which is difficult to develop with traditional approaches and increasingly needed in computing applications. Obviously the approach does not make hard problems disappear, but it gives the right abstractions and tools for programmers to be able to develop this particularly complex type of software. It does so because of various features that are intrinsic to the architectures that have been developed and made available in concrete platforms for multiagent-oriented software development. We summarize some of these features below.

Reacting to Events × Long-Term Goals: In a dynamic environment, an autonomous agent will have to be attentive to changes and react to them appropriately, because changes in the environment might lead to situations where, for example, the course of action the agent has adopted to achieve a goal will not succeed or perhaps other more important things will require the agent's attention. Yet an agent cannot lose track of the long-term goals it has been asked to achieve by its designer or "owner." In a highly dynamic environment, not reacting to events means losing opportunities for the agent to achieve what is expected of it.

Courses of Action Depend on Circumstances: Agents will be constantly deciding which courses of action to take in order to react to events or to achieve long-term goals or other subgoals. Typically, the best course of action to handle external events or goals depends on the current circumstances (of the agent, other agents, the environment, etc.). So agent languages will typically allow various alternative courses of action to be specified for the same event or goal, and the agent will use its most up-to-date information about the state of itself, other agents, and the environment in order to decide at run-time what needs to be done.

Choosing Courses of Action only When About to Act: Exactly because environments in multiagent applications are very dynamic, the course of action to be used should not be decided too early: things might have changed by

the time the agent is actually about to act. Agent languages often use partially instantiated plans so that not only the details of a plan but also the particular (sub)plan to be used for each (sub)goal is only chosen when the agent is about to act on achieving a particular goal.

Dealing with Plan Failure: Even delaying the decision on particular courses of action might not be enough to ensure that the agent has chosen a suitable course of action in a dynamic environment. While executing a plan, the agent may realize a failure has occurred, so agent languages still need to provide mechanisms to deal with plan failure.

Rational Behavior: In general, agent applications will require that agents behave *rationally*. Of course even for humans the notion of rational behavior can be arguable, but the BDI literature [53] has pointed to very concrete aspects of rationality, and agent programming languages facilitate the task of programmers who want to ensure rational behavior of their autonomous software. One example is that if an agent has an intention (i.e., is committed to the goal of achieving a particular state of affairs) we expect it to reason about how to achieve that intention, and we do not expect the agent to give up before the intention is believed to have been effectively achieved, unless there is good reason to believe it will not be possible to achieve it at all, or the reason why it became an intention in the first place no longer holds.

Social Ability – High-Level Communication, Organization: An essential feature of multiagent systems is that some tasks are only possible if agents *interact*. In order to cooperate or to coordinate their action, agents typically use a high-level form of communication based on the idea of speech acts [3, 58]. Recently, it has become possible to program agents to take part in an agent organization all within the context of multiagent-oriented programming. These are also features that can greatly facilitate the development of complex multiagent systems.

Code Modification at Run-Time: In a platform for multiagent-oriented programming, changing the system program at run-time comes almost for free. First, an agent can have its plan library changed at run-time, which implies that the agent's behavior also changes accordingly. In fact, this is sometimes so simple that it can be done through *speech act-based communication*: thus not only other agents but humans as well can communicate new plans (i.e., know-how or behavior) for the agents. More than that, with some platforms for agent organizations the specification of the social structure and overall social plan and norms that agents ought to follow can be changed on-the-fly

(again by both humans and agents who can reason about their own organization). Needless to say, this means there is the potential for truly adaptive autonomous systems from a practical perspective.

3 Abstractions in the MAOP Paradigm

Abstraction is fundamental in programming and, more generally, in software engineering. Much of the achievement and social impact of current computing systems was only possible by computer scientists endowing programmers with the increasing levels of abstractions that have become available in programming languages. It is precisely the significantly more sophisticated abstractions related to multi-agent systems, in comparison to the “object” abstraction in object-oriented programming, that characterize MAOP as a new programming paradigm – one that has a whole lot of higher-level abstractions to help programmers with the coming challenges of developing more autonomous computational systems for highly dynamic environments. For this reason, this section is dedicated to describing some of the main abstractions related to MAOP, and describing the three clearly distinct levels of a multiagent system, each one leading to specific abstractions that relate to that level.

3.1 Agent Level

MAOP provides abstractions to facilitate the development of software that is both autonomous and social. Autonomy implies making decisions on the most appropriate goals to pursue and the most appropriate courses of action in order to achieve such goals. Social ability implies at least the ability to communicate with other autonomous (software or human) entities in order to cooperate or coordinate. While from the point of view of agent-oriented programming alone social ability is often no more than speech act–based interagent communication, from the point of view of the development of complex multiagent systems the social level involves much more than that. We leave these aspects for later consideration in this section and first concentrate on the features of agent-oriented programming languages, which allow for the development of autonomous software that operates in a social context within a shared environment.

Beliefs are used to represent the information currently available to the agent, regarding the current state of the agent itself and its environment (including other agents sharing the same environment), as well as any past information that the agent may need to recall. The abstraction covering an agent’s informational state is termed *belief* because in dynamic, unpredictable environments (as in most areas

of applications of multiagent systems) it is in general not possible to ensure an agent will have complete and only correct information about its environment.

Agents need to be able not just to represent beliefs but to continuously update them in regard to the perceived state of the environment. Together with the ability to update its beliefs repeatedly comes an important aspect of autonomous agents: the ability to *react* to (believed) changes in the environment. An agent needs to be able to adapt its goals and courses of action to a changing environment, so reacting to changes in beliefs is paramount. More than that, because typically it is not possible to obtain complete information about the given environment, and agents' perception mechanisms might be faulty (or untrustworthy agents might be around), agents need to be able to cope with information that might be neither complete nor guaranteed to be correct.

Perhaps the most important abstraction in agent programming is that of a *goal*. We expect agents to act autonomously and proactively on our behalf, so there needs to be an explicit representation within each agent of what the long-term goals are that it needs to achieve. A goal is typically represented as a property that is currently not believed to be true and that will lead the agent into action in order to make that property true (of the environment, for example).

The most common type of goal in agent languages is that of a *declarative achievement goal*: the agent wishes to bring about a certain state of affairs, which it currently believes not to hold, and is willing to commit itself to acting so as to bring about such a state of affairs. Furthermore, in dynamic environments, it is possible that after completing a course of action that was expected to achieve the desired state of affairs, it might actually turn out that the course of action failed to do so. If that happens, the agent should not just carry on with its other duties; that would be “irrational” from the intentional point of view [16].

This is precisely where the importance of goals for agent programming lies: knowing what property of the world it needs to bring about, and being able to perceive its state, allows an agent to try the relevant courses of action to achieve a goal until it is believed to have been achieved or it is believed not to be possible to achieve (or the reasons that led the agent to adopting the goal in the first place no longer apply). This facilitates the programming of software that can appear to be proactive as well as recovering from failure due to a quickly changing environment. It can also be used to justify the actions of a particular agent in a particular circumstance, and it helps programmers ensure *rational* agent behavior.

Different agent programming languages differ in which goal types they offer to programmers and how they are to be programmed (e.g., by having different syntactic constructs for different types of goals or by having fewer more general constructs and providing particular programming techniques for more complex types of goals). One of the first comprehensive typologies for goals in agent program-

ming was published by Braubach and Pokhar [18], with much work following it, e.g., by van Riemsdijk [67].

Finally, at the agent level there are two more important (and related) notions: *plan* and *intention*. A plan is a course of action that under specific circumstances might help the agent handle a particular event (achieving a long-term goal or reacting to changes in beliefs, for example, about the environment). An intention is an instance of a plan that has been chosen to handle a particular event and has been partially instantiated with information about the event. This *intended means* may contain further goals to achieve, but the particular plan among several possible alternative plans to achieve the same goal will only be chosen at the time the goal is selected to be achieved in a later reasoning cycle. This ensures the agent uses information as up-to-date as possible when committing to particular means to achieve its goals. The agent abstractions will be discussed in more detail in Section 4.1, using JASON as a concrete example.

3.2 Environment Level

When we think of a multiagent system, we tend to think that the environment is “given” rather than designed. Even when that is the case, validation of multiagent systems often requires a simulation of the given environment and, in any case, a notion of environment can be used to ease certain aspects of agent coordination, as is often emphasized in the work on the “Agents and Artifacts” model [47] (which led to the CARTAGO platform discussed in Section 5.2.1), for example. Therefore, we need first-class programming abstractions at the *environment* level of a multiagent system.

A typical abstraction at the environment level is that of an *artifact*: a non-autonomous, non-proactive entity, which however is not an object in object orientation. It needs to transparently encapsulate two other important abstractions connecting agents and their environment: *actions* and *percepts*. Actions are fundamental as they represent the ways agents can change the environment where they are situated, and percepts are the abstractions used to represent the aspects of the environment that agents can perceive or observe. Artifacts can be used to transparently give agents access to software services, for example. They can also be used to create a model of a real-world environment. Whether leveraging objects and services or being used as modeling tools, the important thing is that they facilitate many aspects of software development by automatically interacting at the same degree of abstraction as agents in the multiagent-oriented programming paradigm. The environment abstractions will be discussed in more detail in Section 5.2.1, using CARTAGO as a concrete example.

3.3 Social Level

There are many important programming abstractions at the *social level* in a multi-agent system. Some of the best known are groups, roles, norms, and social plans. Since the concepts related to agent *organizations* are meant to be covered in Chapter 2 of this book, we here only describe them very briefly. An agent organization typically has a *structure*, possibly hierarchical, formed by *groups* of agents, where individual agents might play specific *roles*.

If an agent autonomously chooses to adopt a specific role in an agent organization, it will commit to specific *obligations* that the organization expects of agents playing that role; in addition, agents might acquire permissions to do, or be prevented from doing, certain things as a consequence of agreeing to play a particular role. Such obligations, prohibitions, and permissions are specified by means of social *norms*. Norms can be enforced by *regimentation*, i.e., the system prevents the violation of the norm to even take place, or *sanctions* might be specified so as to punish agents that do not comply with particular norms. Finally, *social plans* can be used to explicitly represent the specific subgoals that each agent in a group is expected to achieve in order for a task that requires the joint work of a team of agents to be accomplished. The social abstractions will be discussed in more detail in Section 5.1.1, using MOISE as a concrete example.

4 Examples of Agent Programming Languages

In this section, we introduce one particular agent programming language, JASON, as a representative of the BDI approach. We illustrate how to model and program agents in this language through various examples. In Section 6, JASON will be used, as part of JACAMO, to solve a non-trivial problem introduced in detail in Chapter 15. While the focus in this section is on individual agent programs, in Section 6 we will deal with the organizational and environmental level of multiagent-oriented programming.

Other approaches to BDI agent programming will be briefly mentioned in Subsection 4.2, with appropriate links to further studies. Finally, Section 4.3 deals with approaches based on executable logics (METATEM, ConGolog, and IndiGolog). METATEM will be discussed also in the next chapter, which deals with verification of multiagent systems.

4.1 JASON

As seen above, agent-oriented programming is based on various abstractions that facilitate the development of software that is both autonomous and social. The

view of agent-oriented programming introduced below is organized according to the main programming abstractions and illustrated by JASON code (i.e., the JASON [14] variant of the AgentSpeak [52] programming language). The illustrative code excerpts are for the simplified Mars rover scenario used in [64]; Figure 13.1 shows the goal-plan tree¹ used in [64]. Later, we show JASON plans that correspond to that goal-plan tree.

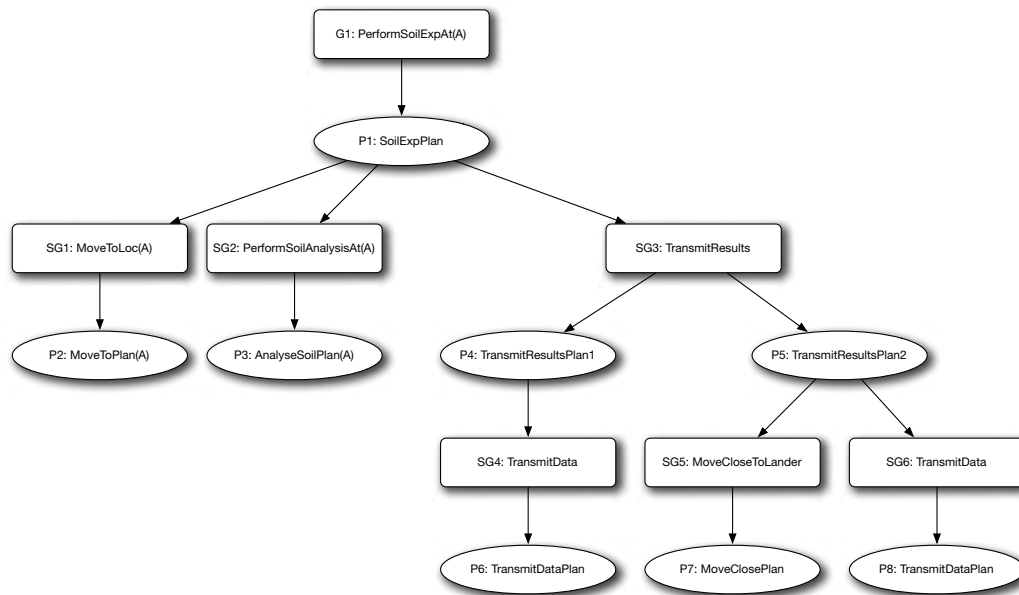


Figure 13.1: A goal-plan tree for a Mars rover scenario [64].

4.1.1 Beliefs

In JASON, a *belief* is represented as a predicate, similar to a Prolog fact. For example,

```
lander_signal(low).
```

However, in JASON beliefs are annotated automatically with the source of the information represented by the belief (the source can be either sensing, communication, or created by the agent program itself). The belief above would actually appear in the belief base as:

¹A goal-plan tree is a structure that shows alternative plans to achieve a goal and the goals that must all be achieved in order for a plan to finish successfully. Such structure is very useful for depicting an agent programmed in a BDI-based agent-oriented programming language.

```
lander_signal(low) [source(percept)]
```

if the information was obtained by perceiving the state of the environment by the agent's own sensing mechanisms. Programmers can also create their own annotations if other meta-level information about individual beliefs are useful in their particular applications. Examples of meta-level information related to a belief are the time it was created and the degree of certainty with which the belief is held.

In JASON, beliefs may also include *reasoning rules*, similar to Prolog, and beliefs can be stored in databases when useful. Of course there are variations for different agent programming languages but the essential aspect of beliefs is to represent part of the system state and, importantly, be able to update such representation as often as possible.

4.1.2 Goals

In JASON, a *goal* is represented also with a predicate, as for beliefs, but it is prefixed with the symbol '!'. So, for example

```
!battery(charged)
```

means that the agent wishes to bring about a state of affairs in which the battery is believed to be charged (which normally implies that the agent believes the battery is not currently charged and therefore this will lead the agent to commit itself to acting so as to bring about such state of affairs).

As mentioned earlier, the best-known type of goal in agent programming refers to the notion of *declarative achievement goals*. This is precisely the type of goal in the example above: the agent currently believes `battery(charged)` is not true and will act so as bring about a state of the world where it does believe so. It is up to the programmer to use the right pattern of plans [39] to ensure that the agent will continue to act rationally towards achieving that goal, even if initial attempts fail to do so.

In the goal-plan tree example, at least by the way the goals were written, they would all seem to be perform goals (i.e., the goal to execute a sequence of actions without further consideration about the state of affairs), although some of them might be better modeled as achievement goals and some even as *maintenance goals*. For example, the goal to be at a particular location is typically an achievement goal, and the plan to transmit results could have used a maintenance goal for the rover to keep sufficiently close to the lander while transmitting data to earth (i.e., the goal to *maintain* a position sufficiently close to the lander).

4.1.3 Plans

An AgentSpeak *plan* has three parts: the trigger, the context, and the body. The trigger part of the plan is used to formulate in which kind of events the plan is to be used (events are additions or deletions of beliefs or goals). The context says in which circumstances the plan is expected to succeed, so that the best course of action might be chosen in different circumstances. The body, finally, has a course of action, containing also further (sub)goals that the agent should commit to in order to handle the particular event. The syntax is as follows:

```
event : context <- body.
```

The examples below show the plans required for an agent to handle the *declarative achievement goal* of positioning itself at some particular location.

```
// already where supposed to be, nothing else to do;
+!at(L) : at(L).
// if not already there, move towards it and
// check if there yet;
+!at(L) : battery(charged) <- move_towards(L); ?at(L).
// if failed to achieve the goal, try it again
-!at(L) : !at(L).
```

The simplest JASON agent implementation that would lead to a goal-plan tree for the goal to do a soil experiment as shown in Figure 13.1 is as follows, where the actions in the plans have been invented so as to keep the example as simple as possible. Note that the preconditions for the plans are not shown in the diagram, so the plans' contexts are all empty although it is not difficult to guess what they should be; also, the '@' symbol is used to label (i.e., to name) a plan in JASON, which is not necessary but included here as it might help with matching the goal-plan tree to the code.²

```
@p1_SoilExpPlan
+!performSoilExpAt(A)
  <- !moveToLoc(A);
    !performSoilAnalysisAt(A);
    !transmitResults.

@p2_MoveToPlan(A)
+!moveToLoc(A) <- move_to(A).
```

²It is not clear in this example whether the authors meant for plans P6 and P8 to be different means of achieving the same goal, but we assume in this code excerpt that this is not the case.

```

@p3_AnalyseSoilPlan(A)
+!performSoilAnalysisAt(A) <- perform_analysis.

@p4_TransmitResultsPlan1
+!transitResults <- !transmitData.

@p5_TransmitResultsPlan2
+!transitResults <- !moveCloseToLander; !transmitData.

@p6_TransmitDataPlan
+!transmitData <- transmit_all_data.

@p7_MoveClosePlan
+!moveCloseToLander <- move_to(lander).

```

4.1.4 Semantics

For most of the BDI-based programming languages that have formal semantics, the semantics has traditionally been given using structural operational semantics [50]. Operational semantics for *AgentSpeak* including some of the extended features made available in JASON has appeared, e.g., in [13, 15, 68], and see also [14]. We do not aim at showing the semantics of an agent programming language in this chapter, but we show a couple of rules of the semantics of JASON to give a flavor of what the semantics of such languages look like. In the references above, full details of the semantics of *AgentSpeak*/JASON can be found, and the references in Section 4.2 can be used to find the semantics of other BDI-based languages (out of the ones discussed in that section, only GOAL and 2APL also have formal semantics).

Operational semantics is given by a transition system where a transition relation on *configurations* of the system is induced by logical rules. A configuration of the system is formally defined as representing a state and abstractly also the architecture of an agent programmed in that particular programming language. As for rules, the part of the rule below the horizontal line states that an agent in a given state (i.e., configuration) can transition to another (the transition relation is represented by a long right arrow) if the conditions above the line are met. Examples of the semantic rules for the *AgentSpeak* language are as follows.

An agent's *reasoning cycle* starts with the selection of a particular event (representing changes in beliefs or goals of the agent) to be handled in that particular cycle. An event is represented by a plan triggering event *te* and an intention *i*. The rule for *event selection* below assumes the existence of a (user-defined) selection function S_E that selects events from a set of events *E* (which is a component of the configuration element *C* representing the agent's current circumstances). The

selected event is removed from E and it is assigned to the ε component of the element of the configuration T that is used to record temporary information needed in later stages of the reasoning cycle. Note how the last component of the configuration keeps track of the various stages a reasoning cycle goes through: from the stage where an event is being selected we go to the one where the relevant plans (plans written to handle that type of event) will be selected. Another rule, **SeleV₂** (not shown here), skips to the intention execution part of the cycle, in case there is no event to handle.

$$\frac{\mathcal{S}_\varepsilon(C_E) = \langle te, i \rangle}{\langle ag, C, M, T, \text{SelEv} \rangle \longrightarrow \langle ag, C', M, T', \text{RelPl} \rangle} \quad (\text{SELEV}_1)$$

$$\text{where: } \begin{array}{ll} C'_E &= C_E \setminus \{ \langle te, i \rangle \} \\ T'_\varepsilon &= \langle te, i \rangle \end{array}$$

The next example we give is from a much later part of the reasoning cycle where the agent has already selected one of its intentions to execute. The formula that appears at the beginning of the body of the plan at the top of the intention will be executed, and there are various rules depending on the type of that particular formula. The rule below is for the case where an *action* is to be executed. The action a in the body of the plan is added to the set of actions A . The action is removed from the body of the plan and the intention is updated to reflect this removal. It is part of the overall agent architecture to then take that action execution request from the A component and to associate that request with the particular agent “effectors” to which the action corresponds.

$$\frac{T_1 = i[\text{head} \leftarrow a; h]}{\langle ag, C, M, T, \text{ExecInt} \rangle \longrightarrow \langle ag, C', M, T', \text{ClrInt} \rangle} \quad (\text{ACTION})$$

$$\text{where: } \begin{array}{ll} C'_A &= C_A \cup \{a\} \\ T'_1 &= i[\text{head} \leftarrow h] \\ C'_I &= (C_I \setminus \{T_1\}) \cup \{T'_1\} \end{array}$$

4.2 Other BDI-Based Languages

There exist many BDI-based programming languages, some of them started as prototypes developed within a PhD project. We list here some languages that we consider more mature, which have been developed over a few years, and that are actively maintained and applied: JADEX, 2APL, AGENTFACTORY, GOAL, and BRAHMS.

JADEX is a Java-based, modular, and standards compliant agent platform that allows the development of goal-oriented agents following the BDI model (see Figure 13.2). JADEX provides a framework including a run-time infrastructure

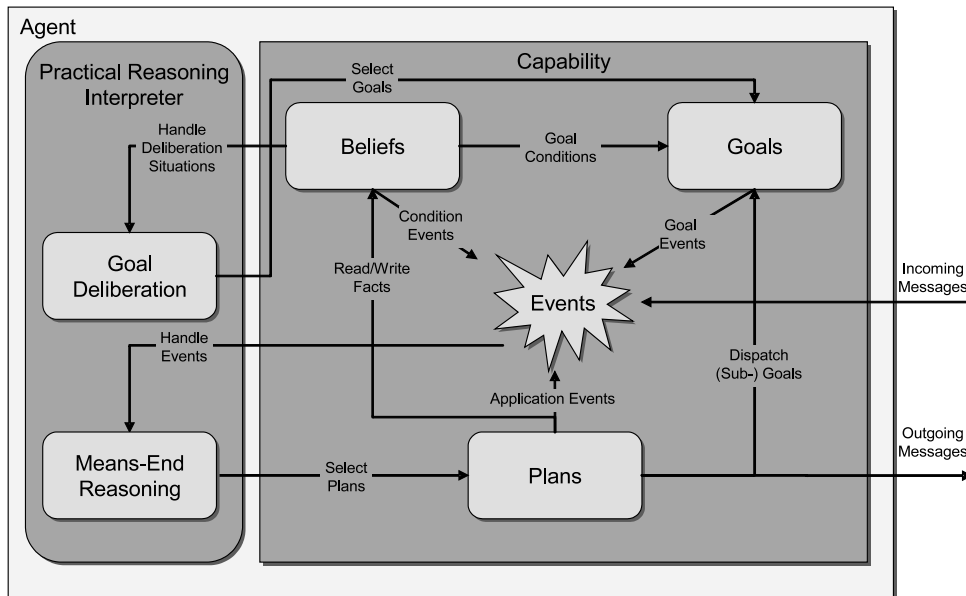


Figure 13.2: The abstract architecture of JADEX.

for agents, the agent platform, and an extensive run-time tool suite. It allows for programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE. The developed software framework is available under GNU's LGPL license, and is continuously evolving (available on SourceForge.net). We refer the interested reader to <http://jadex-agents.informatik.uni-hamburg.de/> and [17, 51].

2APL provides programming constructs both (1) to specify a multiagent system in terms of a set of individual agents and a set of environments, as well as (2) to implement cognitive agents based on the BDI architecture (agent's beliefs, goals, plans, actions, events, and a set of rules through which the agent can decide which actions to perform). 2APL is a *modular programming language* allowing the encapsulation of cognitive components in modules. Its graphical interface, through which a user can load, execute, and debug 2APL multiagent programs using different execution modes and several debugging/observation tools, is shown in Figure 13.3. For further detail we refer the reader to <http://apapl.sourceforge.net/> and [2, 21].

AGENTFACTORY (see Figure 13.4) has at its core a FIPA-standards-based run-time environment (RTE), which provides support for the deployment of heterogeneous agent types, ranging from pure Java-based agents to customizable agent architectures and agent programming languages. While much work in the past has focused on the development of the AGENTFACTORY agent program-

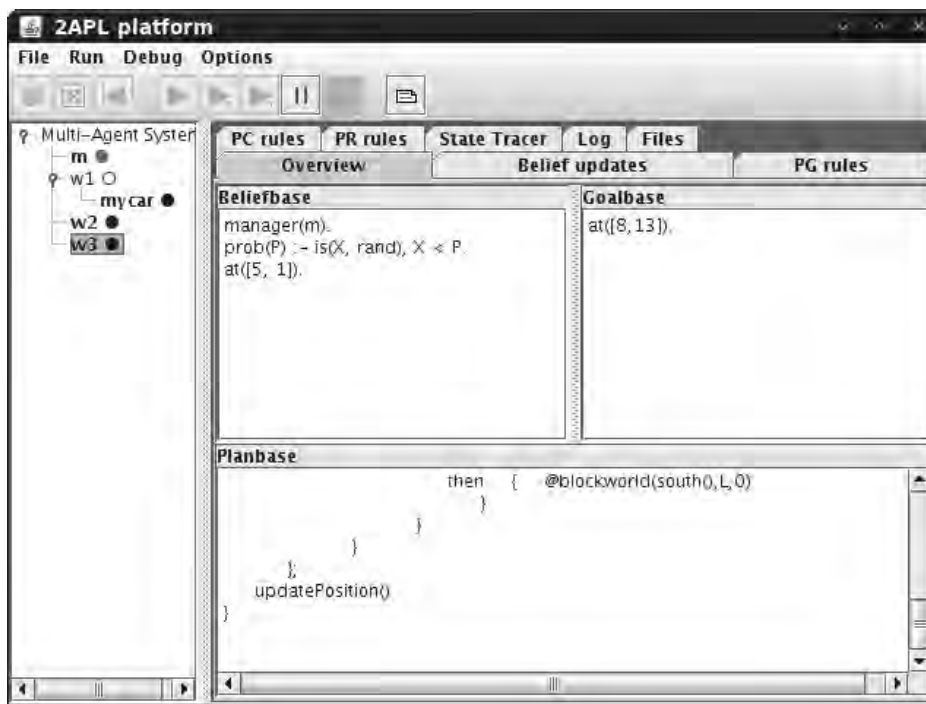


Figure 13.3: A screenshot of the 2APL platform.

ming language (AFAPL), more recent work has resulted in the common language framework, a suite of components for AGENTFACTORY that are intended to help simplify the development of diverse logic-based agent programming languages (APLs). For further references see <http://www.agentfactory.com> and [41, 44].

BRAHMS (see Figure 13.5) can be seen both as a programming language as well as a *behavioral modeling language*. It allows users to model complex agent organizations to simulate people, objects, and environments. Agents can deal with time and can be easily integrated with Java agents. A particularly exciting application is the multiagent system OCAMS that was developed with BRAHMS and is running continually in NASA's ISS mission control: <http://ti.arc.nasa.gov/news/ocams-jsc-award/>. Refer to <http://www.agentisolutions.com/> and [19, 61, 66] for details.

4.3 Approaches Based on Executable Logics

In this section we present two approaches that are tightly connected to logics and their underlying deductive engine. The first one is Concurrent METATEM, a language that can be seen as an *executable specification* in a temporal logic.

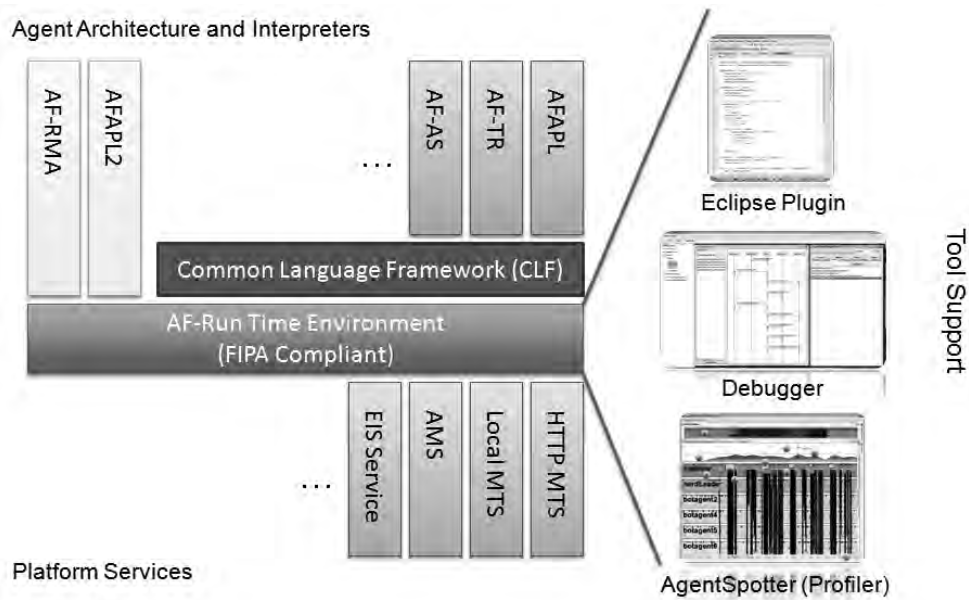


Figure 13.4: The architecture of AGENTFACTORY.

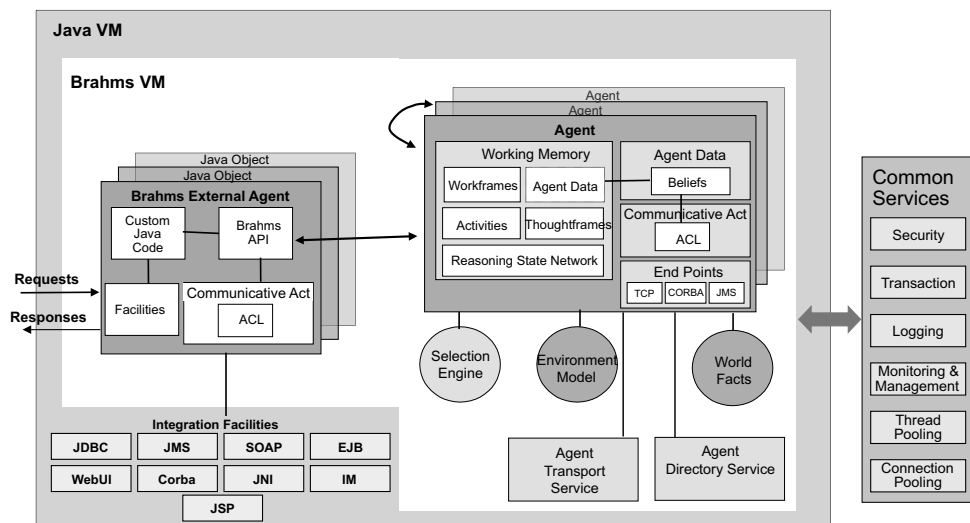


Figure 13.5: The architecture of BRAHMS.

A GOAL agent program is a set of modules that consist of various sections including knowledge, beliefs, goals, a program section that contains action rules, and action specifications. Each of these sections is represented in a knowledge representation language such as Prolog, *answer set programming*, *SQL* (or Datalog), or the *planning domain definition language*. The figure on the right illustrates these sections. For details, refer to <http://mmi.tudelft.nl/trac/goal> and [34, 36].

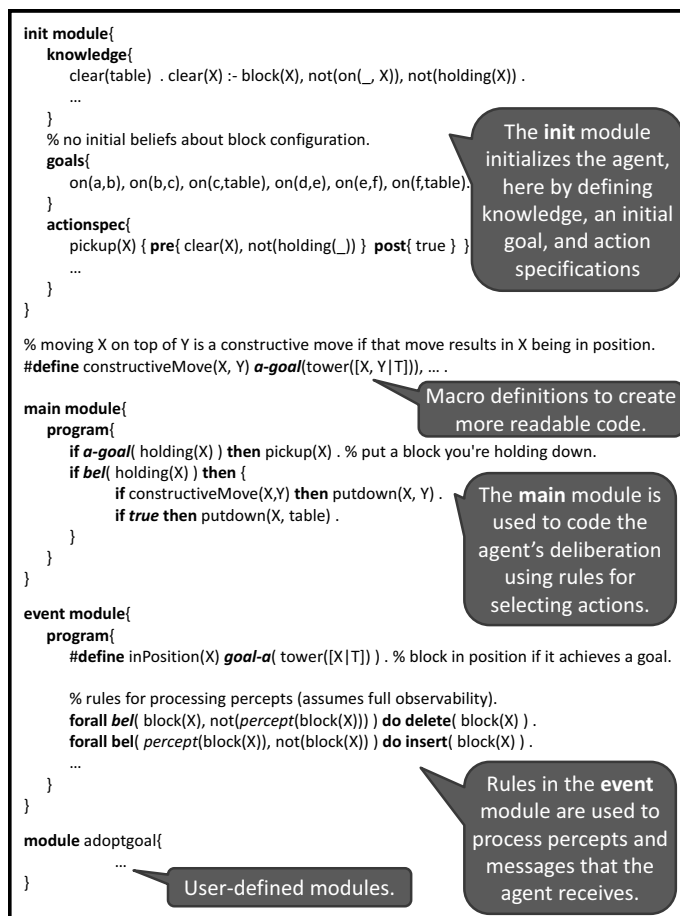


Figure 13.6: The structure of a module in GOAL.

The second one is ConGolog, a language based on the situation calculus of first-order logic (to be more precise it is a dynamic logic combined with ideas from the situation calculus).

We follow [45] to compare Concurrent METATEM and ConGolog on a simple contract-net protocol (Figure 13.7 from [45]), consisting of the following steps. (1) The seller agent may receive a *contractProposal* message from a buyer agent. (2) According to the *amount* of merchandise required and the *price* proposed by the buyer, the seller may *accept* the proposal, *refuse* it, or try to *negotiate* a new price by sending a *contractProposal* message back to the buyer. (3) The buyer agent can do the same (*accept*, *refuse*, or *negotiate*) when it receives a *contractProposal* message back from the seller. (4) If there is enough merchandise in the warehouse and the price is greater or equal to a *max* value, the seller accepts by sending an *accept* message to the buyer and *concurrently ships* the required mer-

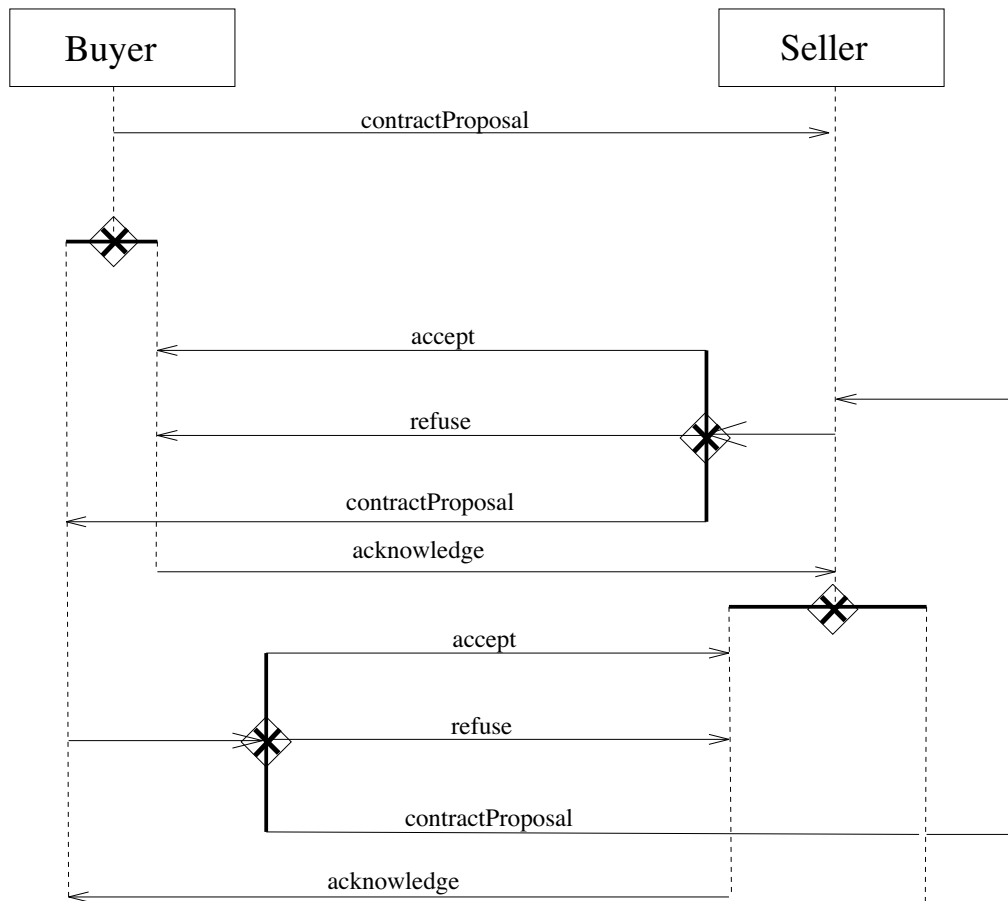


Figure 13.7: The contract-net protocol.

chandise to the buyer (if no concurrent actions are available, answering and shipping merchandise will be executed sequentially). (5) If there is not enough merchandise in the warehouse or the price is lower or equal to a *min* value, the seller agent refuses by sending a *refuse* message to the buyer. (6) If there is enough merchandise in the warehouse and the price is between *min* and *max*, the seller sends a *contractProposal* to the buyer with a proposed price evaluated as the mean of the price proposed by the buyer and *max*. (7) The merchandise to be exchanged are **oranges**, with minimum and maximum price of 1 and 2 euro, respectively. The initial amount of oranges that the seller possesses is 1,000.

4.3.1 METATEM

Concurrent METATEM is the concurrent extension of METATEM, a programming language for multiagent systems based on a first-order temporal logic (with discrete, linear models with finite past and infinite future) [27, 28]. A Concurrent METATEM system contains a number of concurrently executing agents that are able to communicate through message-passing. Each agent executes a *first-order temporal logic specification* of its desired behavior. An agent has two main components: (1) an *interface*, which defines how the agent may interact with its environment (i.e., other agents), and (2) a *computational engine*, defining how the agent may act.

An *agent interface* consists of three components: (1) a unique *agent identifier*, which names the agent; (2) a set of predicates defining what messages will be accepted by the agent – they are called *environment predicates*; and (3) a set of predicates defining messages that the agent may send – these are called *component predicates*.

The computational engine of an agent is based on the METATEM paradigm of *executable temporal logics*. The idea behind this approach is to directly execute a declarative agent specification given as a set of *program rules*, which are temporal logic formulae of the form: “antecedent about past \rightarrow consequent about future.” The intuitive interpretation of such a rule is “*on the basis of the past, do so in the future.*”

Besides the usual temporal operators (see also Chapter 14, Section 3.4) \bigcirc , $\cdot\mathcal{U}\cdot$, we also use $\ominus\phi$ (which can be defined as a macro): it expresses “the current state is not the initial state, and ϕ was true in the previous state”.

The internal knowledge base of the seller agent contains the following *rigid* predicates (predicates whose value never changes): *min-price(orange, 1)*, *max-price(orange, 2)*. It also contains the following *flexible* predicates (predicates whose value changes over time): *storing(orange, 1000)*.

The most important program rule of the seller agent is the following one:

$$\begin{aligned} &\forall \text{Buyer, Merchandise, Req_Amnt, Price} \\ &\ominus[\text{contractProposal}(\text{Buyer}, \text{seller}, \text{Merchandise}, \text{Req_Amnt}, \text{Price}) \wedge \\ &\text{storing}(\text{Merchandise}, \text{Old_Amount}) \wedge \text{Old_Amount} \geq \text{Req_Amnt} \wedge \\ &\text{max-price}(\text{Merchandise}, \text{Max}) \wedge \text{Price} \geq \text{Max}] \implies \\ &[\text{ship}(\text{Buyer}, \text{Merchandise}, \text{Req_Amnt}, \text{Price}) \wedge \\ &\text{accept}(\text{seller}, \text{Buyer}, \text{Merchandise}, \text{Req_Amnt}, \text{Price})] \end{aligned}$$

If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state all the conditions were met to accept the proposal, then accept the Buyer’s proposal and ship the required merchandise.

The other two temporal logic rules look very similar. They formalize (1) *If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state the conditions were not met to accept the Buyer's proposal, then send a refuse message to Buyer*, and (2) *If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state the conditions were met to send a contractProposal back to Buyer, then send a contractProposal message to Buyer with a new proposed price.*

4.3.2 ConGolog and IndiGolog

ConGolog ([32]) and IndiGolog ([33]) are languages extending Golog, a language based on the *situation calculus* introduced by McCarthy. Golog stands for alGOl in LOGic.

Actions are described as in the classical STRIPS approach: they have preconditions that must be satisfied in order to apply the action. The postcondition then describes the change of the world. The evolution of the world is described within the logical language by *fluents*, which are terms in the language. The effects of an action is formalized by successor-state axioms: they describe what the successor state of a given state looks like if an action is applied.

Whereas first-order logic as a specification language using deduction (i.e., a theorem prover) as the underlying procedural mechanism is often too slow and difficult to handle for the non-expert, Golog is a programming language that hides the application of the situation calculus and is thus much more user-friendly. Procedures in Golog actions are reduced to primitive actions which refer to *actions in the real world*, such as picking up objects, opening doors, moving from one room to another, and so on. Golog allows programmers to state procedures of the form

```
while ( $\exists$ block) ontable(block) do remove_a_block end-while
proc remove_a_block ( $\Pi$ x)[pickup(x); putaway(x)] end-proc
```

In general the evaluation of a Golog procedure results in a trace of the primitive actions to be executed. Thus a plan is produced to lead from the initial state to the goal state.

Successor state axioms:

(1) $\mathbf{do}(\mathit{send}(\mathit{Sender}, \mathit{Receiver}, \mathit{Message}), S) \equiv \top$.

This formalizes that *Receiver* receives *Message* from *Sender* in situation $\mathbf{do}(\mathit{send}(\mathit{Sender}, \mathit{Receiver}, \mathit{Message}), S)$, which is reached by executing *send*(*Sender*, *Receiver*, *Message*) in situation *S*.

- (2) **storing**(*Merchandise*, *Amount*, **do**(*A*, *S*)) \equiv
 (*A* = **ship**(*Buyer*, *Merchandise*, *Required-amount*) \wedge
storing(*Merchandise*, *Required-amount* + *Amount*, *S*)) \vee
 (*A* \neq **ship**(*Buyer*, *Merchandise*, *Required-amount*) \wedge
storing(*Merchandise*, *Amount*, *S*)).

This formalizes “The seller has a certain *Amount* of *Merchandise* if it had *Required-amount* + *Amount* of *Merchandise* in the previous situation and it shipped *Required-amount* of *Merchandise*, or if it had *Amount* of *Merchandise* in the previous situation and it did not ship any *Merchandise*”.

```

proc seller-life-cycle
if receiving(Buyer, seller,
             contractProposal(Merchandise, Required-amount, Price), now)
then
  if storing(Merchandise, Amount, now)
     $\wedge$  Amount  $\geq$  Required-amount
     $\wedge$  Price  $\geq$  max-price(Merchandise)
  then ship(Buyer, Merchandise, Required-amount) ||
    send(seller, Buyer, accept(Merchandise, Required-amount, Price))
  else
    if storing(Merchandise, Amount, now)  $\wedge$  Amount  $\geq$  Required-amount
     $\wedge$  min-price(Merchandise) < Price < max-price(Merchandise)
    then send(seller, Buyer, contractProposal(Merchandise,
      Required-amount, (Price + max-price(Merchandise))/2))
    else nil

```

5 Organization and Environment Programming

Originally agent programming languages concentrated on agents and not the *environment* where they were situated nor the *social/organizational* context to which they were supposed to adhere. These are two other separate levels of programming that are essential for MAOP, and will be discussed in this section.

5.1 Organizations

Organizations for multiagent systems and normative systems for agent societies have turned into major research topics in multiagent systems in the last few years. Many different approaches and related frameworks have been developed through multidisciplinary research. As is common in computer science, some research work focuses exclusively on theoretical aspects, and only some frameworks are

worked out so as to become sufficiently practical for actual use in the development of software for multiagent applications. In this section, we concentrate on a few approaches available in the literature that have direct relevance to agent programming. Chapter 2 of this book covers some of the topics in multiagent organizations.

5.1.1 MOISE

MOISE is one of the best-known approaches for modeling and programming multiagent organizations. The MOISE organization modeling language explicitly decomposes the specification of organizations into *structural*, *functional*, and *normative* dimensions [38, 40]. The modeling language is accompanied by a graphical language (examples of MOISE diagrams will appear in Section 6) and XML is used to store the organizational specifications. Those specifications are then managed by an organization management infrastructure at run-time. Because the organization is managed at run-time from its explicit representation, agents can in principle reason about their own organization and change it during the system execution.

The structural dimension specifies the *roles*, *groups*, and *links* (e.g., communication) that exist within the organization. The definition of roles is such that when an agent chooses to play some role in a group, it is accepting some behavioral constraints and rights related to this role. The functional dimension determines how the *global collective goal* should be achieved, i.e., how these goals are decomposed (through *social plans*) and grouped into coherent sets of subgoals (called *missions*) to be distributed among the agents. Such decomposition of a global goal results in a goal tree, called *scheme*, where the leaf-goals can be achieved individually by the agents. The normative dimension binds the structural dimension with the functional one by means of the specification of *permissions* and *obligations* toward missions assigned to particular roles. When an agent chooses to play some role in a group, it accepts these permissions and obligations.

A mission defines all the goals an agent playing a given role commits to when participating in the execution of a scheme. The normative specification relates roles and missions through *norms*. Note that a norm in MOISE is always an obligation or permission to commit to a mission. Goals are therefore indirectly linked to roles since a mission is a set of goals. Prohibitions are assumed “by default” with respect to the specified missions: if the normative specification does not include a permission or obligation for a particular role-mission pair, it is assumed that the role does not grant the right to commit to the mission.

We do not give details of MOISE here although some MOISE diagrams appear in Section 6 where we give an example of a multiagent program written for the JACAMO platform. Further details can be found in the references given above.

5.1.2 Other Approaches

As mentioned earlier, there are many different approaches to agent organizations and normative multiagent systems. We only mention here some of the approaches that can be used in practical software development, namely Electronic Institutions, OperettA, and 2OPL.

One of the best-known and practical approaches to organized multiagent systems resulted from the continued research efforts referred to as Electronic Institutions (EI) [60]. Besides having roles and norms, one distinguishing aspect of EI is that interaction is guided by *scenes* representing agent encounters. In particular, multiagent interaction protocols are specified in a graphical tool called ISLANDER, and electronic institutions written in ISLANDER are run on a software infrastructure called AMELI.

Another relevant tool is OperettA [1], which is based on Eclipse and allows the specification and analysis of OperA organizations. Since Chapter 2 of this book covers OperA in detail, we refer the interested reader to that chapter rather than describe it here.

The 2OPL project (<http://oopluu.sourceforge.net/>) is particularly worth mentioning here because it is based on a normative programming language [65], similar to the one used in JACAMO for the automatic generation of an artifact-based organization management infrastructure [37]. The 2OPL programming language has norms and sanctions as primary programming constructs. It is meant to define the overall organization of a multiagent system without any restrictions on (or support for) how to develop individual agents that will take part in the organization.

5.2 Environments

Sometimes an agent is developed for a whole class of applications, not just for a single one. When an agent should be usable for many applications, it might be situated in different environments where it should do its duties accordingly. From an abstract point of view, it would be desirable if agents could be developed as independently of a particular environment as possible. That would make it possible to develop interesting environments and powerful multiagent systems *independently* of each other. Unfortunately, this is not normally the case as there is no general agreement upon what belongs to shared environments and what belongs to the agent platforms, and how exactly they interact.

In this section we describe two approaches that complement each other:

CARTAGO: an implementation of a general-purpose model for environment programming, based on the notion of artifact (see Section 5.2.1). CARTAGO

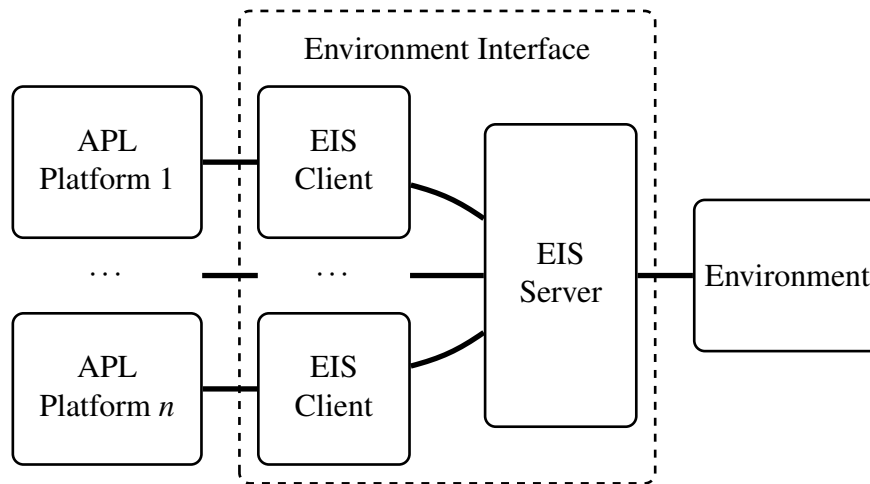


Figure 13.8: Ideal world: Heterogeneous agents acting within a shared environment.

is rooted in the idea that the environment should be seen as a *first-class abstraction* for MAS engineering, as a suitable place to encapsulate functionalities and services to support agents' activities [71].

EIS: an implemented platform for connecting agent systems to existing environments (Subsection 5.2.2). EIS adopts the more classical view that the notion of environment is used to identify the external world (with respect to the system, being a single agent or a set of agents), which is perceived and acted upon by the agents so as to fulfill their tasks [56].

While CARTAGO is used for *creating* environments, EIS is a programming infrastructure for interfacing between agent platforms and existing environments.

Although there are quite a few sophisticated environments around, coupling a particular agent platform to it requires a lot of tedious work. In an ideal world (shown in Figure 13.8), one would like to couple different agent platforms at the same time to such environments in order to perform the required tasks: a *heterogeneous* agent system where agents written in different languages can work together. We refer to [4].

A combination of EIS and CARTAGO is an interesting research line. Currently, there are particular bridges available for connecting agent platforms such as JADEX, JASON, and 2APL to CARTAGO (see [54]). Implementing an EIS bridge for CARTAGO could be used to connect CARTAGO to any kind of agent platform that supports EIS.

We note that there are more frameworks available in the literature than we can address in this chapter. We would like to mention at least Kaminka et al.'s Game-

Bots (<http://gamebots.sourceforge.net/>) and Brom et al.'s **Pogamut** (<https://artemis.ms.mff.cuni.cz/pogamut/>, [30]). These and similar frameworks are used in the area of *serious games*.

5.2.1 CARTAGO

CARTAGO (Common Artifact infrastructure for Agent Open environment) [55] is a framework and infrastructure for programming and executing MAS environments based on the A&A (Agents and Artifacts) meta-model [47].³ The approach allows the design and programming of an environment in terms of a dynamic set of first-class computational entities called *artifacts*, collected in logical localities called *workspaces*, which can be distributed over the network.

The responsibilities and functionalities that can be suitably encapsulated in such a notion of environment can be summarized by the following three different levels of support [71] (see Figure 13.9): (i) *a basic level*, where the environment is exploited to simply enable agents to access the *deployment context*, i.e., the given external hardware/software resources that the MAS interacts with (sensors and actuators, a printer, a network, a database, a web service, etc.); (ii) *abstraction level*, exploiting an environment abstraction layer to bridge the conceptual gap between the agent abstraction and low-level details of the deployment context, hiding such low-level aspects from the agent programmer; and (iii) *interaction-mediation level*, where the environment is exploited to both regulate the access to shared resources, and mediate the interaction between agents. These levels represent different degrees of functionality that agents can use to achieve their goals.

On the one hand, artifacts are first-class abstractions for MAS designers and programmers to shape environment functionalities by defining the types of artifacts – and with that the structure and the computational behavior of the concrete artifact instances that can be instantiated in workspaces. On the other hand, artifacts are first-class entities of the agents' world, which agents perceive, use, and dynamically instantiate and compose as such – analogously to resources and tools used by humans in organization environments (Figure 13.10 shows metaphorically a bakery as an artifact-based environment).

Through its approach to artifacts as first-class abstraction, CARTAGO provides direct support to all the three levels identified above. At the basic level, artifacts can be used to wrap and enable access to resources in the deployment context. At the abstraction level, artifacts can be used to define a new abstraction layer both hiding the low-level details of the deployment context and pos-

³CARTAGO sources and technology are available as an open-source project at <http://cartago.sourceforge.net>

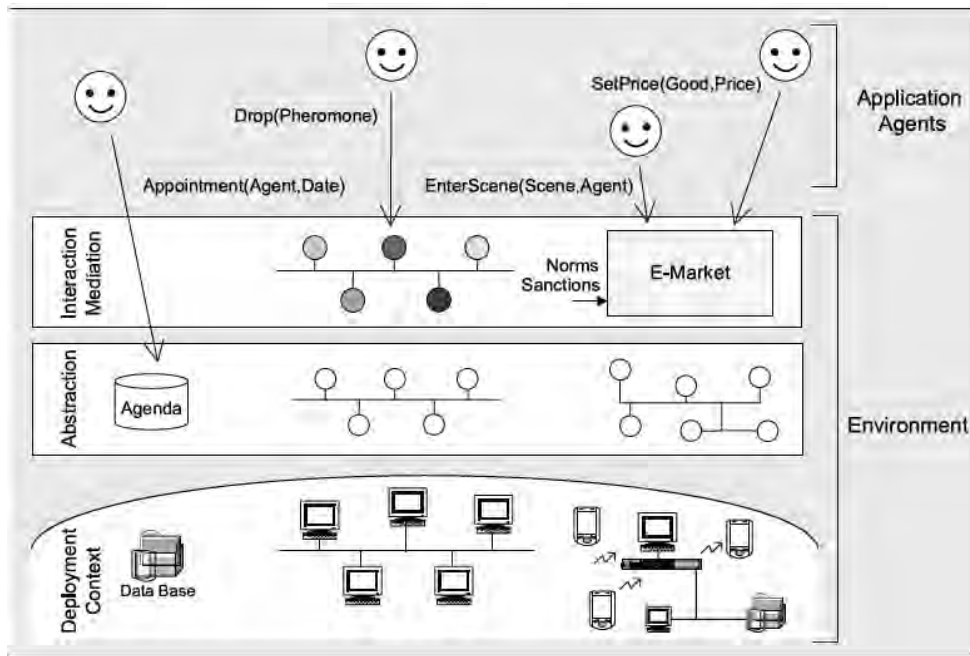


Figure 13.9: Environment support levels [71].

sibly containing computational resources that are fully *virtual*, independent of the deployment context. At the interaction-mediation level, artifacts can be designed to encapsulate and enact coordination mechanisms, so as to realize forms of environment-mediated interaction and coordination.

Figure 13.11 shows the CARTAGO meta-model, including the main concepts that characterize an artifact-based environment. To be used, an artifact provides a usage interface containing a set of *operations* that agents can execute to obtain some functionality. Operations correspond to agent *actions*, so the repertoire of actions available to an agent working within an artifact-based environment is given by the set of operations provided by the overall set of artifacts currently instantiated in the environment (which can be changed dynamically during run-time). To be perceived, an artifact can have one or multiple *observable properties*, as data items that can be perceived by agents as environment state variables, whose value can change dynamically through operation executions. Such executions may generate *observable events* as environment signals that can be relevant for agents using/observing the artifact. By using CARTAGO with BDI agent programming languages, when an agent starts observing an artifact, the artifact's observable properties are mapped into the beliefs base of the agent as beliefs about the current (observable) state of the environment. So changes to the observable properties

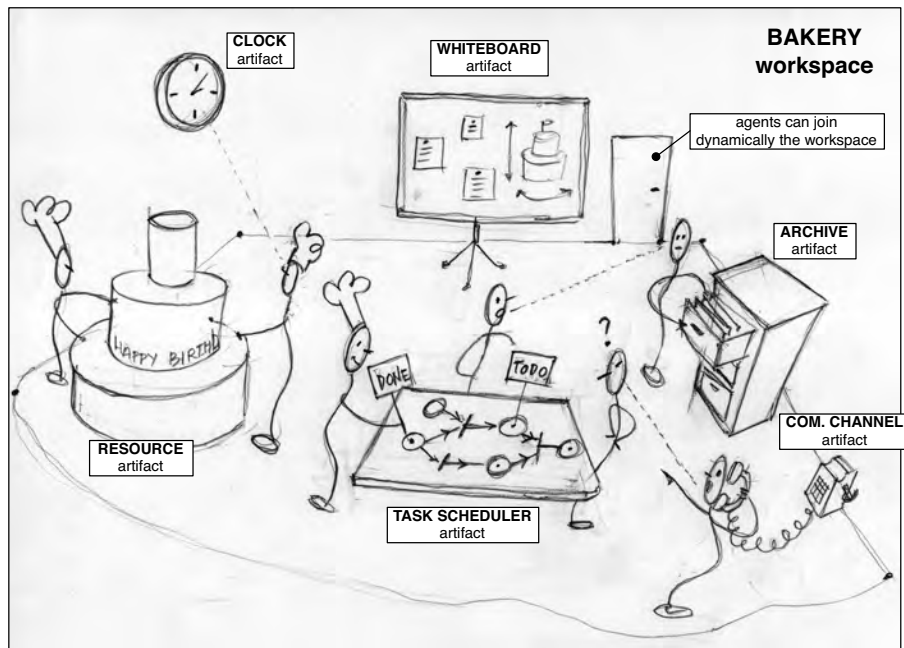


Figure 13.10: A bakery used as a metaphor to frame the notion of workspaces and artifacts as resources and tools used by agents to work.

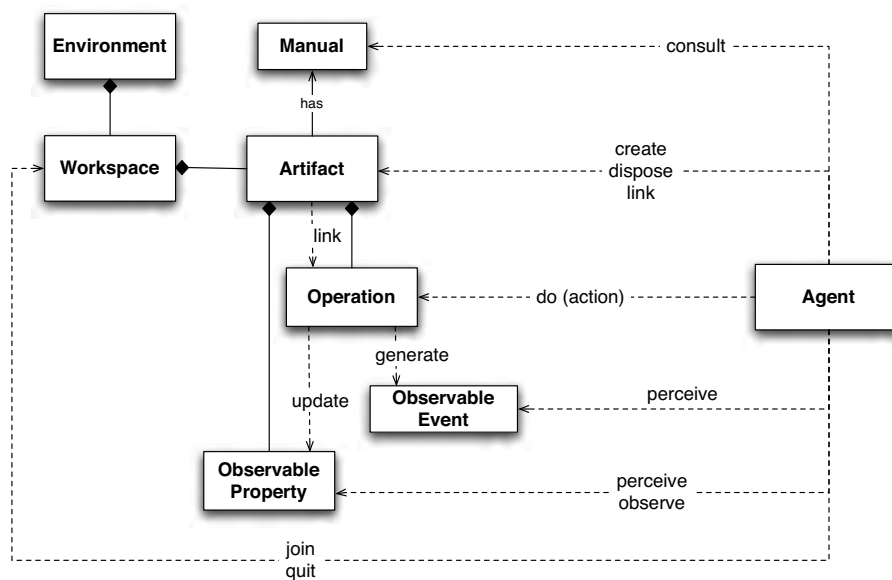


Figure 13.11: CARTAGO meta-model based on A&A.

```

class Counter extends Artifact {
    void init(){
        defineObsProperty("count",0);
    }

    @OPERATION void inc(){
        ObsProperty p = getObsProperty("count")
        updateObsProperty("count",p.intValue()+1);
    }
}

```

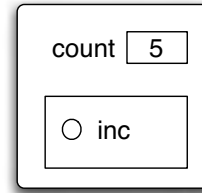


Figure 13.12: Java-based implementation of a *Counter* artifact type having one operation *inc* and one observable property *count*.

```

/* Jason agent creating and using          // Jason agent observing the counter
   the counter */                          !observe

!create_and_use.                           +!observe
                                           <- lookupArtifact("c0",Id);
+!create_and_use : true                    focus(Id).
  <- makeArtifact("c0","Counter",[],Id);
    inc;
    inc[artifact_id(Id)].                  +count(N)
                                           <- println("perceived new value: ",N).

```

Figure 13.13: Snippets of two JASON agents creating and using a shared counter (on the left) and observing the shared counter (on the right).

are then directly perceived as changes in the beliefs related to those properties.

From a programming point of view, CARTAGO provides a Java-based API to program artifacts and a run-time environment to execute artifact-based environments, along with a library of predefined general-purpose artifact types. Besides, it also provides the API and the underlying mechanism to extend existing agent programming languages/frameworks so as to program agents to work within CARTAGO environments. So, by integrating CARTAGO with existing agent programming languages, heterogeneous agents written in different agent programming languages can cooperatively work inside the same workspaces, sharing and co-using the same artifacts.

As a simple example illustrating the CARTAGO API for implementing artifacts (for MAS designers) and for using artifacts (for the agents), Figure 13.13 shows two JASON agents working inside a CARTAGO environment. One agent creates a *Counter* artifact – whose implementation is shown in Figure 13.12 – and uses it by executing the *inc* action (operation) twice. The other agent observes the counter, reacting to changes in the beliefs about its observable state (the *count* observable property).

Criterion	2APL	GOAL	JADEX	JASON
Portability	jar-files	jar-files	everything	jar-files
Perceiving	sense-actions and external events	getting all percepts via a provided method	accessing env-objects or requesting percepts from an env-agent	getting all percepts via a provided method
Acting	invoking methods	invoking a method	manipulating env-object or sending a message to an env-agent	invoking a method
Abstract environment functionality	mapping from ag-names to ag-objects	no special functionality	no abstract environment defined	logging and action-scheduling
Formats	terms/atoms encoded as Java-objects	strings	java-objects	logical literals and structures encoded as Java-objects
Java accessibility	jar-files	jar-files	everything that is in the class-path	jar-files

Table 13.1: Comparison of some agent platforms regarding their environment interface.

5.2.2 EIS

The overall idea of EIS is to set up a standard interface for connecting agent platforms to environments. As a result, agent platforms that support the interface can connect to any environment that implements the interface. This will significantly reduce the effort required from agent and environment programmers: the environment code needed to implement the interface needs to be written only once for each platform.

EIS has been crafted based upon a detailed analysis of some agent languages shown in Table 13.1 (taken from [7]).

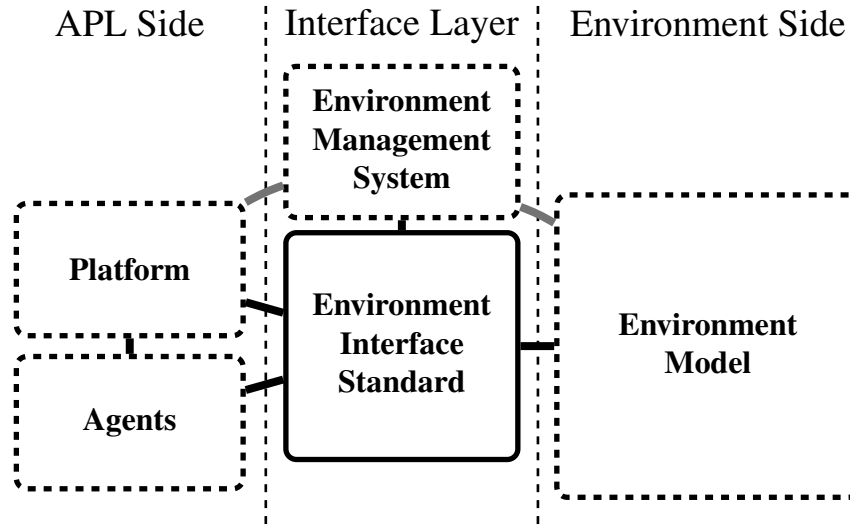


Figure 13.14: Components of EIS. The platform containing the agents is separated from the environment model. The interface layer acts as a link to facilitate the interaction of the components.

We identify five components from a software engineering perspective (see Figure 13.14):

- **Agent:** The objective of defining an environment interface standard is to provide a generic approach for connecting *agents* to environments.
- **Environment model:** It is assumed that an environment contains *controllable entities*. Controllable entities establish the connection between agents and the environment by providing (1) *effectoric* capabilities and (2) *sensoric* capabilities to agents, thus facilitating agent *situatedness*. Such entities may be controlled from outside the environment (by agents) and are capable of performing actions in the environment to change the state of that environment. EIS assumes that each entity has its own repertoire of actions, and does not assume anything particular about how these actions are performed in the environment.

Controllable entities may be linked one-to-one to concrete Java objects at the code level but need not be so. That is, entities may be *implicit* and it is *not* required that entities can be matched to particular Java objects that are part of an environment. Entities are thus primarily used conceptually and refer to abstract containers for actuators and sensors to which agents can connect. The only representation that is obligatory for each controllable entity is an identifier.

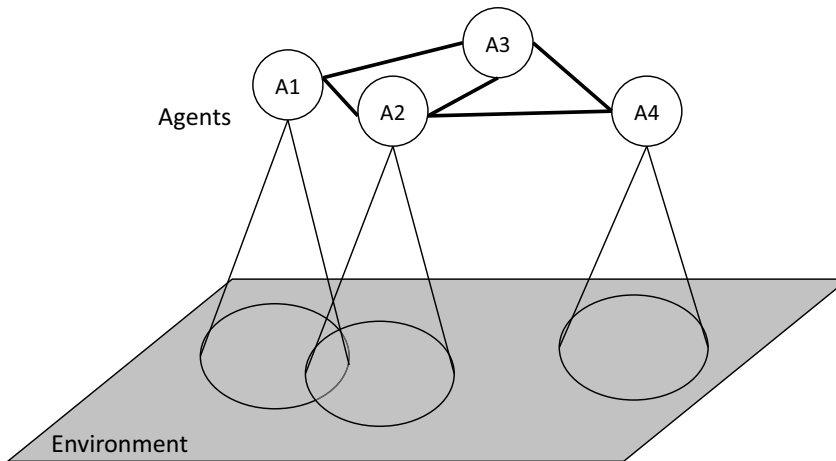


Figure 13.15: Environment-MAS model.

The model of the environment is illustrated in Figure 13.15. It assumes (possibly) intersecting spheres of influence of multiple agents acting in an environment. The *sphere of influence* of an agent is defined by the effecting and sensory ranges of its associated controllable entities. Note that it is *not* assumed that there is a one-to-one relation between agents and controllable entities. Different perspectives may be taken toward these spheres of influence: (i) an action perspective (agents may interfere with each other, they are able to change the same parts of the environment) and (ii) a perception perspective (agents may have different views of the environment).

Controllable entities can be something very simple like thermostats or something quite complex like a robot. In the *Multi-Agent Programming Contest 2008–2010*,⁴ the “cowboys” are the controllable entities. The sensory capabilities are limited to some sort of camera, which provides agents with a limited visual range. The effecting capabilities consist of moving the cowboy in different directions.

Finally, although it is natural to refer to states of the environment, this should *not* be taken to imply that EIS imposes any additional structure on an environment, e.g., requiring it to be a discrete state system. The environment model is generic and can be instantiated to all sorts of specific environments.

- **Platform:** We assume the platform to be responsible for instantiating and

⁴<http://www.multiagentcontest.org>

executing agents. Furthermore, we assume that it facilitates connecting agents with environments, and associating agents to controllable entities in environments through EIS.

- **Environment management system (EMS):** We assume this component to provide all the actions for managing an environment. Such actions might be: initializing an environment using a configuration file, releasing the resources of the environment and terminating its execution, and further actions such as pausing, resuming, and resetting. The environment management system may be run independently of an APL platform.
- **Environment interface standard (EIS):** The environment interface standard is the layer that connects the platform, the environment management system, and the agents with the environment(s).

Figure 13.14 presents the meta-model schematically. It also depicts the relations that need to be supported between the various components. One of the most important relations that should be part of an agent-environment interface is what we call the *agents-entities-relation*. This relation associates agents with *controllable entities* in the environment. The relation is maintained to provide basic bookkeeping functionality. This bookkeeping functionality provides a key role as it determines which agents are allowed to control which entities and also determines which percepts should be provided to which agents.

For further details on EIS, we refer the reader to <http://sf.net/projects/apleis/>, <http://cig.in.tu-clausthal.de/apleis>, [4, 6, 69, 70] and the references therein.

6 Example of Full MAOP in JaCaMo

In Section 4, we have seen examples of agent programs. However, the latest trends in agent programming point to a paradigm coined as multiagent-oriented programming to emphasize the important recent interaction of agent-oriented programming languages with programming approaches developed in the multiagent systems research communities interested in agent organizations and shared environments. It is in the combination of the strengths from each of these three different levels of abstraction in multiagent systems that we are able to harness all the benefits of agent societies as metaphors for developing complex distributed systems.

To demonstrate this, we use a concrete example in JACAMO, the integration of the well-known platforms JASON, CARTAGO, and MOISE. However, JACAMO

is a lot more than three platforms working together. By offering first-class abstractions for the agent (JASON), social (MOISE), and environment (CARTAGO) levels of a multiagent system, it has for the first time in fully working operation unraveled the full potential of MAOP: it allows the development of fully-fledged multiagent systems using very high-level programming. JACAMO can be downloaded from <http://jacamo.sourceforge.net>.

6.1 The Application Scenario

We shall use the running example that appears also in Chapters 14 and 15 to demonstrate the use of a combination of organization-oriented programming with agent-oriented programming and environment-oriented programming. We reproduce here only the main ideas of the scenario, as it is presented in more detail in Chapter 15. Note however that the chosen running example centers mostly on the organization level and, in our design, partly on the environment; agents are very simple and in general they only execute the required action at the right time in orchestration with the team; this is mostly handled by the organization.

The solution presented here should be extended (see Exercises below in Section 8) with an artifact for managing instances of the contract-net protocol suggested by the running example for allocating orders to manufacturing units – we here concentrate on the functioning of a single unit. An implementation of the contract-net protocol in the JASON variant of AgentSpeak can be found in [14] and we later show as an example of artifact, the artifact available with CARTAGO to manage instances of the contract-net protocol. Further, it should be noted that this is one particular software design, and other agent-based solutions can also be implemented. For example, in the absence of social and environment constructs, a purely agent-based solution would potentially require significantly more direct agent-to-agent (speech act-based) communication.

Although the solution shown here does not provide much scope for complex (individual) agent behavior, as this was already partly illustrated earlier in this chapter, it makes sense to use the running example anyway, even if mostly for the purpose of showing the integration of agent programming with organization and environment abstractions.

The scenario used here is explained in detail in Chapter 15, and in particular Figure 15.3 is very illustrative. It shows that the assembly cell of a manufacturing plant is assumed to have two jigs in a rotating table, with two manufacturing robots located at two ends of the table: one that mostly does loading and unloading tasks and another that is able to join separate parts that have been loaded into a jig. To summarize the manufacturing process, and so that this chapter is self-contained, we quote from Section 2 of Chapter 15:

1. *robot1 loads an A part into one of the jigs on the rotating table*
2. *robot1 loads a B part on top of it*
3. *the table rotates so the A and B parts are at robot2*
4. *robot2 joins the parts together, yielding an “AB” part*
5. *the table rotates back to robot1*
6. *robot1 moves the AB part to the flipper*
7. *the flipper flips the part over (“BA”) at the same time as robot1 loads a C part into the jig*
8. *the BA part is loaded on top of the C part*
9. *the table rotates*
10. *robot2 joins the C and BA parts, yielding a complete ABC part*
11. *the table is rotated, and*
12. *robot1 then unloads the finished part.*

Although this process may sound straightforward, it is made more complex by the need to manage a number of concurrent assembly jobs. In other words, we want to be able to exploit parallelism, for instance, having robot2 be assembling one part while robot1 is unloading a different order. On the other hand, we need to respect synchronization requirements such as not moving the table while robot1 or robot2 are operating.

Note that in general in holonic manufacturing there are multiple interchangeable entities so that the process of selecting a table, or an assembly robot, needs some mechanism to manage load-balancing (e.g., using the contract net).

The code we have developed using JACAMO to control a manufacturing cell is available at <http://www.inf.ufrgs.br/~bordini/WeissBookChapter13Ex>. Below, we explain the main points about that solution and include some code excerpts with the intention of illustrating some of the features of fully-fledged multiagent-oriented programming with JACAMO. Note that this is not necessarily the best JACAMO design for a solution to this problem. It was created with the aim of illustrating this approach; in fact, some of the exercises at the end of this chapter suggest other possible design options to be considered and compared.

6.2 Organization Program

By glancing over Figures 13.16 and 13.17, it should be clear how simple⁵ it is to use MOISE to solve the manufacturing coordination problem of the running example. Figure 13.16 shows the structural specification where we specify the relevant roles. As we mentioned, most agents are very simple, with effectively the same JASON code, so we have three roles that just inherit from a “simple controller” role. The same simple controller agent can be used for the robot at one end of the manufacturing unit, which is in charge of operations related to loading the various parts (which we have called the “loader” robot), and for the robot at the other end, which is responsible for joining together two parts (called the “joiner” robot); and we also have a simple controller agent that will play a role in controlling the flipping of joined parts (the “flipper”). The only manufacturing agent that needs a bit more JASON code is the one that rotates the table (called the “rotator”) thereby controlling which of the two robots can operate on each jig. We need one agent playing each of these four roles in order to have a functional “assembly cell group.” Note that MOISE has much more expressive power; we only used the most basic features of MOISE to solve this problem.

In order to run a simulation of the system with the code available at <http://www.inf.ufrgs.br/~bordini/WeissBookChapter13Ex>, that system also includes another agent called “cellmng”, which uses an artifact to simulate the appearance of orders for assembling tasks and an artifact to simulate the operations of the assembly cell, so that we can test if the code appears to work. That agent also creates the MOISE group for the manufacturing unit agents to join, and creates, for each order it accepts, an instance of the `manufacture_ABC` scheme in the functional specification shown in Figure 13.17. This means that in the example code we already took care of allowing two orders to be concurrently processed in a single cell, which is possible since the rotating table has two jigs.

In the specification in Figure 13.16, we define the whole social plan. For readability, we have divided the whole task into three main parts: (1) assembling an AB part, then (2) the complete ABC part, and finally (3) finishing up (by unloading the complete piece). In this manufacturing process most tasks are sequential; the only two tasks that are to be done in parallel are indicated by the parallel horizontal lines in that diagram (see also the legend). It is not difficult to imagine how much interagent communication was saved by delegating the coordination task to the organization rather than doing it all at the level of pure agent-oriented programming. At run-time, the organization will assign goals to agents at the appropriate times, taking into consideration the partial ordering of the goals to be

⁵For the time being, the diagrams need to be manually coded into an XML file in order to run the system, but at the time of writing, work is in progress to use graphical interfaces for automating this task.

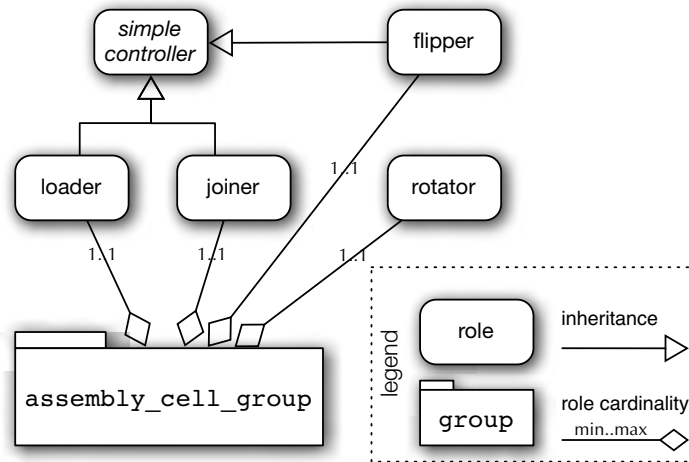


Figure 13.16: MOISE organization: Structural specification.

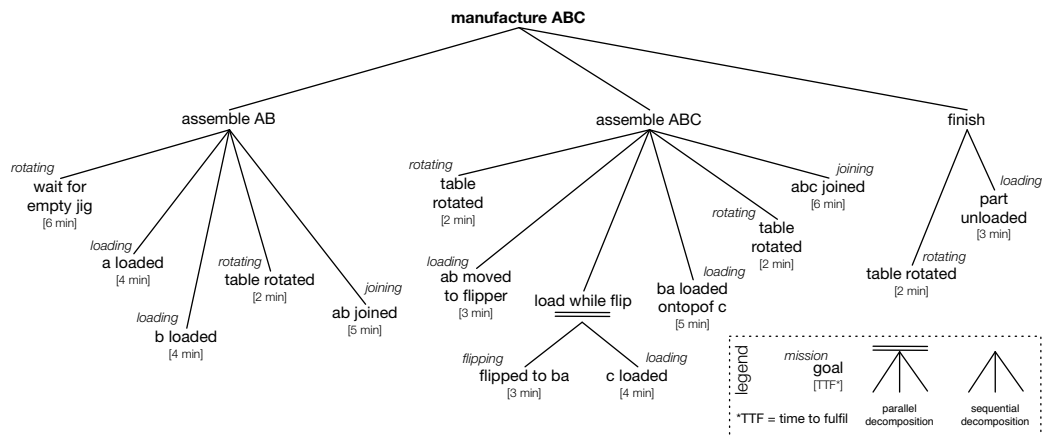


Figure 13.17: MOISE organization: Functional specification.

achieved according to the functional specification.

The careful reader will have also noticed that the diagram in Figure 13.17 annotates each goal to be achieved with one of four different *missions* (namely *loading*, *joining*, *flipping*, and *rotating*). This way, we can easily specify, through the normative specification in Table 13.2, which sets of goals the agents playing each of the four roles will be asked to achieve (by the organization management infrastructure, at run-time).

norm	role	mission
n1	loader	<i>loading</i>
n2	joiner	<i>joining</i>
n3	rotator	<i>rotating</i>
n4	flipper	<i>flipping</i>

Table 13.2: MOISE organization: Normative specification.

6.3 Agent Programs

There are a number of generic JASON plans available with JACAMO that facilitate interaction with MOISE and CARTAGO when programming the agents. Some such plans appear in file `common.asl` in our example code, which is included by all agents that take part in the MOISE organization discussed above. An example of a plan from such a generic library of plans is given below.

```
// obligation to achieve a goal
+obligation(Ag,Norm,achieved(Scheme,Goal,Ag),Deadline) :
    .my_name(Ag) <-
        !Goal[scheme(Scheme)];
        lookupArtifact(Scheme,Id);
        goalAchieved(Goal)[artifact_id(Id)].
```

The plan above says that whenever the agent comes to believe that it has a new obligation toward an organizational goal `Goal` (note the use of JASON higher-order variables here), it just tries to achieve that goal and, if all goes well, the agent (through an ORA4MAS artifact) tells the organization that the goal it was obliged to achieve has been achieved (this is important so that the organization can then delegate further goals to be achieved, possibly by other agents).

In this application, the actual behavior for agents “loader,” “joiner,” and “flipper” is to simply adopt its predetermined role (done by the first plan below) and then do whatever it is asked to do. For example, when the MOISE scheme determines the adoption of goal `!a_loaded`, the agent should just do the action `a_loaded` that activates the loading mechanism in the actual factory. This is possible because the name of such an operation (i.e., an external action for the JASON agent that executes the corresponding artifact operation) in the artifact simulating the manufacturing cell is the same as the goal itself. The second plan below shows that this can be done in a generic way (through the use of the higher-order variable `G` below) for any organizational goal received. The first plan makes the agent join the ORA4MAS workspace so as to take part in the organization;

then it also needs to focus on the ORA4MAS organizational artifact so as to automatically perceive information about the group such as newly created schemes. Finally, it adopts a role in the group (the group and specific role for each of the three agents using this code are specified as initial goals in the JASON project file).

```
// Join the organization and play a role in it
+!join_and_play(GroupName, RoleName)
  <- !in_ora4mas;
      lookupArtifact(GroupName, GroupId);
      focus(GroupId);
      adoptRole(RoleName)[artifact_id(GroupId)].

// Then, just do whatever told by the organization
+!G[scheme(S)] <- G; .print("Doing ", G, " - Scheme ", S).
```

These three agents have only the code above, nothing else. The only agent that requires more complex behavior is the “rotator.” In the MOISE scheme for the manufacturing process, the rotator is assigned two different goals: to wait for an empty jig and to get the table rotated. Below we show the various plans needed to achieve these goals. In the beginning there are two simple Prolog-like rules used to facilitate the plan contexts. They check the number of instances of the manufacturing scheme in MOISE so as to check if there are one or two concurrent orders being manufactured by this cell (each order is handled by one scheme instance; this will be further detailed later in this section). In the code below, note that the name of the scheme that requested the achievement of a particular goal is annotated in the new goal events generated by the agent architecture.

```
// rule to check if we have two concurrent orders (2 Moise schemes)
two_orders :- schemes(L) & .length(L)==2.
// or only one order so far
one_order :- schemes(L) & .length(L)==1.

// 1st organizational goal of the rotator (wait for empty jig)

// avoid conflicts when 2 orders are simultaneously waiting for empty jigs
+!wait_for_empty_jig[scheme(S1)] :
  .desire(wait_for_empty_jib[scheme(S2)]) & S1\==S2 <-
    .wait(500); // wait a bit
    !wait_for_empty_jig[scheme(S1)]. // and try again

// already got an empty jig
+!wait_for_empty_jig[scheme(S)] :
  jig_loader("empty") <-
    reserve_jig(S). // make sure another order doesn't get it too

// will have to wait until the jig at the loader end is empty
+!wait_for_empty_jig[scheme(S)] <-
  .wait({+jig_loader("empty")}); // wait until this event happens
  reserve_jig(S); // make sure empty jig is allocated to this order
  // if there are pending requests to rotate the table
```

```

    if (.desire(table_rotated[scheme(S)])) {
      // might need reconsidering which plan to use for rotating
      .drop_desire(table_rotated[scheme(S)]);
      !!table_rotated[scheme(S)];
    }.

// 2nd organizational goal of the rotator (rotate table)

// Only 1 assembling task, rotate whenever asked
+!table_rotated : one_order <- table_rotated.

// Let it rotate if another job needs it and we're waiting for an empty jig
+!table_rotated :
  two_orders & .desire(wait_for_empty_jig) & not jig_loader("empty") <-
    table_rotated.

// If there are 2 concurrent assembling tasks, wait for both
// to want to rotate before actually rotating

// This is actually the second request to rotate
@tr[atomic] // both goals need to be considered achieved simultaneously
+!table_rotated[scheme(S1)] :
  two_orders & .desire(table_rotated[scheme(S2)]) & S1\==S2 <-
    table_rotated; // one rotation achieves both requests
    .succeed_goal(table_rotated[scheme(S2)]).

// The first attempt just waits, 2nd request releases both
+!table_rotated[scheme(S)] :
  two_orders <-
    .wait(1000); // wait a bit
    !table_rotated[scheme(S)]. // try again

```

The comments in the code above explain all of the details. It will be noted that as a new order can be started at any time during the manufacturing of another, it is a rather difficult synchronization problem that the rotator has to solve.

Finally, the cell manager agent has mostly procedural code to create the simulation artifacts and initialize the organization. Other than that, it has only a few plans, the following one being the longest:

```

// each order generates an instance of the Manufacture scheme
@opl[atomic] // needs to be an atomic operation: changing the no. of schemes
+order(N) :
  formationStatus(ok) [artifact_id(GrArtId)]
  & schemes(L) & .length(L)<=1 <- // no more than 1 order under way
  // wait until empty jig is correctly positioned at loader robot
  .concat("order", N, SchemeName);
  makeArtifact(SchemeName, "ora4mas.nopl.SchemeBoard",
    ["src/manufacture-os.xml", manufacture_schm, false, true], SchArtId);
  focus(SchArtId); // get all info about this Moise scheme
  addScheme(SchemeName) [artifact_id(GrArtId)].

```

This plan accepts at most two concurrent manufacturing orders, and creates the necessary ORA4MAS scheme artifact to handle a new (simulated) manufacturing order. Focusing on the scheme allowed the cell manager to automatically perceive the state of the scheme; for example, it needs to know when the order has been

completed so that a new one can be accepted. This plan also needs to add the newly created scheme to the (well-formed) MOISE group.

6.4 Environment Program

We mentioned a few environment artifacts above when describing our implementation. Besides those, it makes sense to use an artifact to manage instances of the *contract-net protocol* (CNP), required in the scenario as a mechanism for load balancing (i.e., selecting the best assembly cell for a particular part assembling request). While in most cases artifacts required for particular tasks have to be developed using the CARTAGO API, in this case we do not need to do that as CARTAGO already offers an artifact-based implementation of CNP management. Of course CNP can also be managed directly by agents, as in the example given in [14, Section 6.3]. However, there are many advantages of using the artifact-based implementation in this case. For example, it reduces the amount of direct agent-to-agent communication required, and, most importantly, allows the use of CNP in open multiagent systems: when agents join a CARTAGO workspace, they will be able to automatically perceive the available CNP instances and join in if they so wish.

Even though it is not necessary to program the artifact in this case, we show the code of one of the artifacts (the task board) just to illustrate the environment side of the system, and to show how artifacts are at a different level of abstracts as normal objects in Java. The observable properties and operations automatically become percepts/actions available to all agents that enter the shared workspace. In the code for the task board⁶ below, agents use the `announce` operation on this artifact when they wish to start a new CNP instance for a particular task. This artifact will then create another artifact to manage that particular instance of the CNP, with an observable property with the task description (which again is accessible to any agents joining the workspace at run-time). It is in that newly created artifact that agents will be able to bid, and the agent being awarded the contract will be announced there too.

```
public class TaskBoard extends Artifact {
    private int taskId;

    void init() {
        taskId = 0;
    }

    @OPERATION void announce
        (String taskDescr, int duration, OpFeedbackParam<String> id){
        taskId++;
        try {
```

⁶Available at <http://cartago.sourceforge.net>.

```

        String artifactName = "cnp_board_"+taskId;
        makeArtifact(artifactName, "c4jexamples.ContractNetBoard",
            new ArtifactConfig(taskDescr,duration));
        defineObsProperty("task", taskDescr, artifactName);
        id.set(artifactName);
    } catch (Exception ex) {
        failed("announce_failed");
    }
}

@OPERATION void clear(String id) {
    String artifactName = "cnp_board_"+taskId;
    this.removeObsPropertyByTemplate("task", null, artifactName);
}
}

```

Note that in this section we only included excerpts, although all the important parts of the code were covered. Still, we strongly encourage the reader to look at the complete (fully commented) code and run the system. The working example for one manufacturing cell can be downloaded from <http://www.inf.ufrgs.br/~bordini/WeissBookChapter13Ex>, and we leave as an exercise to use the CNP artifacts for extending to multiple cells.

7 Conclusions

The type of programming made available in practice for the first time with JACAMO is effectively very new. Although agent-oriented programming has been around for many years, it was only with the combination of agent-oriented programming with organization-oriented programming and environment-oriented programming that the true potential of a programming paradigm inspired by multi-agent systems was unraveled.

Of course there are many open issues in this programming paradigm. There are specific issues at the various levels, for example, the issue of encapsulation of agent code has some proposed solutions but more is required in the way of experimentation so that the best approaches can be determined. The combination of the three levels of abstraction, being rather recent, is also likely to require much further research still. Equally, we should expect much progress in practical programming tools in order to make the approach usable in industry.

However, with the trends in computing toward a vision of the future where autonomy and large-scale interaction will be required by so much software, it is reasonable to expect that MAOP will play an important role in the mainstream computing industry in that future.

Acknowledgments

We thank Alessandro Ricci for providing us with some of the material that went into Section 5.2.1. Section 5.2.2 was mainly based on [5, 7] and we thank the authors for allowing us to use a few excerpts from those papers. We also thank Koen Hindriks, Maarten Sierhuis, Lars Braubach, Alexander Pokahr, Mehdi Dastani, and Rem Collier for providing us with some material that went into Section 4.2. Tristan Behrens, Michael Köster, and Federico Schlesinger proofread parts of this chapter and gave us useful comments.

Rafael Bordini acknowledges the support of CNPq grant 307924/2009-2. Jürgen Dix acknowledges that this work was partly funded by the NTH School for IT Ecosystems. (NTH [Niedersächsische Technische Hochschule] is a joint university consisting of Technische Universität Braunschweig, Technische Universität Clausthal, and Leibniz Universität Hannover.)

8 Exercises

1. **Level 1** Discuss how the programming techniques presented in this chapter address each of the required features of autonomous systems presented in Subsection 2.2.
2. **Level 1** Create a mind map of all the programming abstractions at the three different levels of a multiagent system, available in full MAOP as well as related concepts and ideas.
3. **Level 1** Using JASON, give the Mars rover agent used as an example in Section 4.1 further know-how (for example taking panoramic pictures and performing spectrometric analysis of rock samples) and write a complex plan with a particular mission for the rover using both the old and new know-how.
4. **Level 1** Develop a simple simulated environment and perform experiments with your JASON code for the Mars rover.
5. **Level 2** The solution presented in Section 6 for the manufacturing scenario concentrated on the functioning of a single manufacturing unit. Extend the provided code with an artifact for managing instances of the contract-net protocol – as suggested in the description of the scenario in Chapter 15 – for allocating incoming orders to the various manufacturing units. Remember that you can use the contract-net artifact that is available in CARTAGO's own library.

6. **Level 2** Re-implement the agent code of the manufacturing example in other agent-oriented programming languages; compare the different programming styles.
7. **Level 2** Propose an alternative mechanism for allocating the assembling requests to the various manufacturing units and run simulations to evaluate the performance of both mechanisms (including overall productivity of the factory, communication overhead, etc.).
8. **Level 3** Redesign the whole manufacturing system so that the levels of abstraction in JACAMO take responsibility for different aspects of the application. For example, avoid coordination at the organization level and leave it for direct agent communication, or avoid using contract-net at the level of the environment. Compare the performance of the system (as in the exercise above) and, most importantly, compare also the elegance of the new version of the code with the original solution.
9. **Level 3** Propose a methodology for designing multiagent systems applications that assumes that the programming platform has first-class abstractions at the three main levels of abstractions of a multiagent system.
10. **Level 3** Design and implement a multiagent system for a complex application using JACAMO.
11. **Level 4** In the version of JACAMO used in this book, there are no programming constructs to create interaction protocols. Propose a general mechanism for this and integrate it within the JACAMO platform as a concrete implementation.
12. **Level 4** Propose an approach to formally verify JACAMO systems, in such a way that properties to be verified could refer to social constructs such as roles as well as individual mental attitudes and artifact states.
13. For the next few exercises, we refer to the multiagent contest site <http://www.multiagentcontest.org>. There, a whole platform, contest scenarios, as well as several dummy agents to start with (in several agent languages) are available. The following exercises can be based on your favorite agent language.
14. **Level 1** In this exercise you learn how to start the server, the monitor, and some dummy agents. Please note that a Unix-Shell, preferable a bash, is needed. For Windows you can use MSYS <http://www.mingw.org/wiki/MSYS>.

- (a) Download and extract the MASSim package from <http://multiagentcontest.org/teaching-downloads>.
- (b) Start the server with `massim/scripts/startServer.sh`, choose one simulation and press <Enter>.
- (c) Start the Mars-Monitor with `massim/scripts/startMarsMonitor.sh`.
- (d) Start the dummy-agents with `javaagents/scripts/startAgents.sh`.
- (e) Read the scenario description.

For more information consult the documentation of the MASSim package.

- 15. **Level 2** Write a reactive agent that does random walks and recharges its battery whenever the energy is low. Use the subsumption architecture for the implementation.
- 16. **Level 2** Modify the agent by adding a belief and a goal base to it. Implement a modified Dijkstra algorithm that computes the shortest path to an unexplored node. The agent should now be able to analyze the topology and store the information observed in the belief base by repeatedly walking to the nearest unexplored node.
- 17. **Level 2** Add a communication mechanism to the agents. Design a communication protocol that allows the agents to efficiently exchange the information about the map topology.
- 18. **Level 2** Implement a strategy that allows the agents to explore the map as fast as possible.
- 19. **Level 2** Implement a strategy for the agent team that has two phases: first, explore the map; and second, conquer the zone with the highest value.
- 20. **Level 3** Extend your agent team with the concept of roles. Each two agents should implement one of the five roles (Explorer, Sentinel, Repairer, Saboteur, Inspector).
- 21. **Level 3** Your agent team has to achieve at least these achievements: (1) probe 5 nodes, (2) analyze 10 edges, (3) inspect 5 agents, (4) attack the opponent's agents at least 5 times successfully, and (5) the zone value should be greater than 20.
- 22. **Level 2** Develop a method for the agent team of when and how to buy new equipment for the agents.

23. **Level 2** Design a strategy for defending your own zones and attacking the zones of the opponents.
24. **Level 2** Implement agent teams for the gold-miners scenario.
25. **Level 4** Implement agent teams for the cows-and-cowboys scenario.

References

- [1] Huib Aldewereld and Virginia Dignum. OperettA: Organization-Oriented Development Environment. In Mehdi Dastani, Amal El Fallah-Seghrouchni, Jomi Hübner, and João Leite, editors, *LADS*, volume 6822 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.
- [2] Natasha Alechina, Mehdi Dastani, Brian Logan, and John-Jules Ch. Meyer. Reasoning about Agent Deliberation. *Autonomous Agents and Multi-Agent Systems*, 22(2):356–381, 2011.
- [3] J. L. Austin. *How to Do Things with Words*. Oxford University Press, London, 1962.
- [4] Tristan Behrens. *Towards Building Blocks for Agent-Oriented Programming*. PhD thesis, Department of Computer Science, Clausthal University of Technology, Germany, 2012.
- [5] Tristan Behrens, Rafael Bordini, Lars Braubach, Mehdi Dastani, Jürgen Dix, Koen Hindriks, Jomi Hübner, and Alexander Pokahr. An Interface for Agent-Environment Interaction. In Rem Collier, Jürgen Dix, and Peter Novák, editors, *Proceedings of the 8th International Workshop on Programming Multi-agent Systems (ProMAS)*, volume 6599 of *LNCS*, pages 170–185, Heidelberg, Germany, 2011. Springer Verlag.
- [6] Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. The Multi-agent Programming Contest from 2005–2010. *Annals of Mathematics and Artificial Intelligence*, 59(3–4):277–311, 2010.
- [7] Tristan Behrens, Koen Hindriks, and Jürgen Dix. Towards an Environment Interface Standard for Agent Platforms. *Annals of Mathematics and Artificial Intelligence*, 61(1–2):3–38, 2011.
- [8] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J. Gómez-Sanz, João Leite, Gregory M. P. O’Hare, Alexander Pokahr, and Alessandro Ricci. A Survey of Programming Languages and Platforms for Multi-agent Systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

- [9] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors. *Multi-agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005.
- [10] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni. Preface – Special Issue on Programming Multiagent Systems. *International Journal of Agent-Oriented Software Engineering*, 1(3/4), 2007.
- [11] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors. *Multi-agent Programming: Languages, Tools and Applications*. Springer, 2009.
- [12] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni. Preface. *Autonomous Agents and Multi-Agent Systems*, 23(2):155–157, 2011.
- [13] Rafael H. Bordini and Jomi Fred Hübner. Semantics for the JASON Variant of AgentSpeak (Plan Failure and some Internal Actions). In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 635–640. IOS Press, 2010.
- [14] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-agent Systems in AgentSpeak Using JASON*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.
- [15] Rafael H. Bordini and Álvaro F. Moreira. Proving BDI Properties of Agent-Oriented Programming Languages. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):197–226, 2004.
- [16] Michael E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
- [17] L. Braubach and A. Pokahr. Addressing Challenges of Distributed Systems Using Active Components. In F. Brazier, K. Nieuwenhuis, G. Pavlin, M. Warnier, and C. Badica, editors, *Intelligent Distributed Computing V - Proc 5th International Symposium on Intelligent Distributed Computing (IDC 2011)*, volume 382 of *Computational Intelligence*, pages 141–151. Springer, 2011.
- [18] Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal Representation for BDI Agent Systems. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *ProMAS*, volume 3346 of *Lecture Notes in Computer Science*, pages 44–65. Springer, 2004.
- [19] William J. Clancey, Maarten Sierhuis, Charis Kaskiris, and Ron van Hoof. Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System. In Ingrid Russell and Susan M. Haller, editors, *Proc. FLAIRS Conference*, pages 7–11. AAAI Press, 2003.

- [20] Rem Collier, Jürgen Dix, and Peter Novák, editors. *Programming Multi-agent Systems, Revised Selected and Invited Papers of ProMAS 2010*, volume 6599 of *Lecture Notes in Computer Science*. Springer, 2011.
- [21] Mehdi Dastani. 2APL: A Practical Agent Programming Language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
- [22] Mehdi Dastani, Amal El Fallah-Seghrouchni, João Leite, and Paolo Torroni, editors. *Languages, Methodologies, and Development Tools for Multi-agent Systems, Second International Workshop, LADS 2009, Torino, Italy, September 7-9, 2009, Revised Selected Papers*, volume 6039 of *Lecture Notes in Computer Science*. Springer, 2010.
- [23] Mehdi Dastani and Jorge J. Gómez-Sanz. Programming Multi-agent Systems. *Knowledge Eng. Review*, 20(2):151–164, 2005.
- [24] Mehdi Dastani, Davide Grossi, John-Jules Ch. Meyer, and Nick A. M. Tinnemeier. Normative Multi-agent Programs and Their Logics. In John-Jules Ch. Meyer and Jan Broersen, editors, *KRAMAS*, volume 5605 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2008.
- [25] Jürgen Dix and Michael Fisher. Where Logic and Agents Meet. *Annals of Mathematics and Artificial Intelligence*, 61(1):15–28, 2011.
- [26] Jürgen Dix and Joao Leite. Special Issue: Selected Papers of CLIMA 2010. *Annals of Mathematics and Artificial Intelligence*, 62(1/2), 2011.
- [27] Michael Fisher. Temporal Semantics for Concurrent MetateM. *Journal of Symbolic Computation*, 22(5/6):627–648, 1996.
- [28] Michael Fisher. A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, 1997.
- [29] Michael Fisher, Rafael H. Bordini, Benjamin Hirsch, and Paolo Torroni. Computational Logics and Agents: A Road Map of Current Technologies and Future Trends. *Computational Intelligence*, 23(1):61–91, 2007.
- [30] Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondrej Burkert, Radek Píbil, Jan Havlíček, Lukás Zemčák, Juraj Simlovic, Radim Vansa, Michal Stolba, Tomáš Plch, and Cyril Brom. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In Frank Dignum, Jeffrey M. Bradshaw, Barry G. Silverman, and Willem A. van Doesburg, editors, *AGS*, volume 5920 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2009.
- [31] Michael P. Georgeff and Amy L. Lansky. Reactive Reasoning and Planning. In *AAAI*, pages 677–682, 1987.

- [32] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog: A Concurrent Programming Language Based on the Situation Calculus. *Journal of Artificial Intelligence*, 121(1-2):109–169, 2000.
- [33] Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque, and S. Sardinia. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In R.Ĥ. Bordini, M. Dastani, J. Dix, and A. El Fallah-Segrouchni, editors, *Multi-agent Programming: Languages, Tools and Applications*, pages 31–72. Springer, 2009.
- [34] Koen V. Hindriks. Modules as Policy-Based Intentions: Modular Agent Programming in GOAL. In Mehdi Dastani, Amal El Fallah-Seghrouchni, Alessandro Ricci, and Michael Winikoff, editors, *ProMAS*, volume 4908 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2007.
- [35] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent Programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
- [36] Koen V. Hindriks and Tijmen Roberti. GOAL as a Planning Formalism. In Lars Braubach, Wiebe van der Hoek, Paolo Petta, and Alexander Pokahr, editors, *MATES*, volume 5774 of *Lecture Notes in Computer Science*, pages 29–40. Springer, 2009.
- [37] Jomi Fred Hübner, Olivier Boissier, and Rafael H. Bordini. From Organisation Specification to Normative Programming in Multi-agent Organisations. In Jürgen Dix, João Leite, Guido Governatori, and Wojtek Jamroga, editors, *CLIMA XI*, volume 6245 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2010.
- [38] Jomi Fred Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting Multi-agent Organisations with Organisational Artifacts and Agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
- [39] Jomi Fred Hübner, Rafael H. Bordini, and Michael Wooldridge. Plan Patterns for Declarative Goals in AgentSpeak. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 1291–1293. ACM, 2006.
- [40] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing Organised Multiagent Systems using the MOISE. *IJAOSE*, 1(3/4):370–395, 2007.
- [41] Howell R. Jordan, Jennifer Treanor, David Lillis, Mauro Dragone, Rem W. Collier, and Gregory M. P. O’Hare. AF-ABLE in the *Multi-Agent Programming Contest* 2009. *Annals of Mathematics and Artificial Intelligence*, 59(3-4):389–409, 2010.
- [42] John E. Laird. Extending the SOAR Cognitive Architecture. In Pei Wang, Ben Goertzel, and Stan Franklin, editors, *AGI*, volume 171 of *Frontiers in Artificial Intelligence and Applications*, pages 224–235. IOS Press, 2008.

- [43] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. Log. Program.*, 31(1-3):59–83, 1997.
- [44] David Lillis, Rem W. Collier, Mauro Dragone, and Gregory M. P. O’Hare. An Agent-Based Approach to Component Management. In *Proc. 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 529–536, 2009.
- [45] Viviana Mascardi, Maurizio Martelli, and Leon Sterling. Logic-Based Specification Languages for Intelligent Software Agents. *Theory and Practice of Logic Programming*, 4(4):429–494, 2004.
- [46] Jörg P. Müller, Michael Wooldridge, and Nicholas R. Jennings, editors. *Intelligent Agents III, Agent Theories, Architectures, and Languages, ECAI ’96 Workshop (ATAL), Budapest, Hungary, August 12-13, 1996, Proceedings*, volume 1193 of *Lecture Notes in Computer Science*. Springer, 1997.
- [47] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A Meta-model for Multi-agent Systems. *Autonomous Agents and Multi-Agent Systems*, 17(3), 2008.
- [48] Andrea Omicini, Sebastian Sardina, and Wamberto Weber Vasconcelos, editors. *Declarative Agent Languages and Technologies VIII - 8th International Workshop, DALT 2010, Toronto, Canada, May 10, 2010, Revised, Selected and Invited Papers*, volume 6619 of *Lecture Notes in Computer Science*. Springer, 2011.
- [49] Samuel J. Partington and Joanna Bryson. The Behavior Oriented Design of an Unreal Tournament Character. In *IVA*, pages 466–477, 2005.
- [50] Gordon Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, September 1981.
- [51] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI Reasoning Engine. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 149–174. Springer, 2005.
- [52] Anand S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In Walter van de Velde and John W. Perram, editors, *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World (MAAMAW)*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer, 1996.
- [53] Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In Victor R. Lesser and Les Gasser, editors, *Proc. First International Conference on Multiagent Systems (ICMAS)*, pages 312–319. The MIT Press, 1995.

- [54] Alessandro Ricci, Michele Piunti, Daghan L. Acay, Rafael H. Bordini, Jomi Fred Hübner, and Mehdi Dastani. Integrating Heterogeneous Agent Programming Platforms within Artifact-based Environments. In *Proc. 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 225–232. IFAAMAS, 2008.
- [55] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. Environment Programming in CARTAGO. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-agent Programming: Languages, Platforms and Applications*, Vol. 2, pages 259–288. Springer, 2009.
- [56] Stuart Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach (2nd ed.)*. Prentice Hall, 2003.
- [57] Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the Semantics of Deliberation in Indigolog – from Theory to Implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2-4):259–299, 2004.
- [58] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
- [59] Yoav Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.
- [60] Carles Sierra, Juan A. Rodriguez-Aguilar, Pablo Noriega Blanco-Vigil, Josep-Ll. Arcos-Rosell, and Marc Esteva-Vivancos. Engineering Multi-agent Systems as Electronic Institutions. *UPGRADE European Journal for the Informatics Professional*, V(4), August 2004.
- [61] Richard Stocker, Maarten Sierhuis, Louise A. Dennis, Clare Dixon, and Michael Fisher. A Formal Semantics for Brahms. In *Proc. 12th International Workshop on Computational Logic in Multi-agent Systems (CLIMA)*, volume 6814 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2011.
- [62] V.S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Özcan, and Robert Ross. *Heterogenous Active Agents*. MIT Press, 2000.
- [63] John Thangarajah, James Harland, David N. Morley, and Neil Yorke-Smith. Operational Behaviour for Executing, Suspending, and Aborting Goals in BDI Agent Systems. In Omicini et al. [48], pages 1–21.
- [64] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & Exploiting Positive Goal Interaction in Intelligent Agents. In *AAMAS*, pages 401–408. ACM, 2003.

- [65] Nick A. M. Tinnemeier, Mehdi Dastani, and John-Jules Ch. Meyer. Roles and Norms for Programming Agent Organizations. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *Proc. AAMAS*, pages 121–128. IFAAMAS, 2009.
- [66] Bart-Jan van Putten, Virginia Dignum, Maarten Sierhuis, and Shawn R. Wolfe. OperA and Brahms: A Symphony? In *Proc. 9th International Workshop on Agent-Oriented Software Engineering (AOSE)*, volume 5386 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2008.
- [67] Birna van Riemsdijk, Mehdi Dastani, and John-Jules Ch. Meyer. Semantics of Declarative Goals in Agent Programming. In *Proc. 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 133–140. ACM, 2005.
- [68] Renata Vieira, Álvaro F. Moreira, Michael Wooldridge, and Rafael H. Bordini. On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. *Journal of Artificial Intelligence Research*, 29:221–267, 2007.
- [69] Website, *Multi-Agent Programming Contest*. <http://www.multiagentcontest.org> (April 2011).
- [70] Website, *EIS. Environment Interface Standard*. <http://cig.in.tu-clausthal.de/eis> (April 2011).
- [71] Danny Weyns, Andrea Omicini, and James J. Odell. Environment as a First-class Abstraction in Multi-agent Systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb 2007. Special Issue on Environments for Multi-agent Systems.

Chapter 14

Specification and Verification of Multiagent Systems

Jürgen Dix and Michael Fisher

1 Introduction

As we have seen from the previous chapter, there are many ways to implement a multiagent system. Although certainly not trivial, the task of producing a working multiagent system is relatively straightforward. However, producing a multiagent system that *always* “works” is much more challenging. But how can we assess this? How can we describe *exactly* what we want our system to do? And then how can we *ensure* that any system we build actually conforms to this description? A further aspect that is increasingly becoming important is how we can assess that a multiagent system built *by someone else* conforms to our requirements.

In this chapter we will address these important problems. Specifically, we will show how formal logics and logical procedures can form the basis for the comprehensive analysis of multiagent systems with respect to their formal requirements. While we aim to focus on the analysis, through *verification*, of systems, we must naturally explore ways of describing our requirements in a formal way. Thus, we begin by considering the *formal specification* of agents and multiagent systems. From this we move on to the *formal verification* of these systems, examining the many different ways that designs, models, and programs can be exhaustively assessed with respect to a formal specification.

Although this can be quite a technical topic, we will present only the basic ideas and exhibit them through simple examples. In addition, there are many

existing verification systems that we will point the reader toward as we proceed through the chapter. As this area is still very much at the forefront of research activity, many of the tools are prototypes under active development. As such, they are not so refined or stable; nevertheless, we encourage you to try some of these (leading edge) tools and aid their development.

1.1 Why Logic, Specification, and Verification?

We might ask, at this point, why is verification needed *at all*? As multiagent systems grow in popularity they are beginning to be used in safety-critical areas, such as aerospace or factory processes. It is clearly vital that such systems behave as expected in all situations, especially if human lives are at risk. Yet before we even reach such safety-critical applications there is a class of *business-critical* applications. Here, any failure in our agents might not be life-threatening but could well compromise the viability of the business, for example, through failures involving finance, security, or privacy. Even if we do not consider the multiagent systems we are dealing with to be involved in *any* critical scenarios, it is important that our multiagent software works as expected, even in “non-critical” applications. Software failure can have severe effects, such as *bad publicity*, *legal aspects*, *product recall*, or *software revision and re-testing*.

Here are three famous (but not specifically agent-based) events that might have been avoided with better specification and verification methods:

AT&T Telephone Network Outage (1990): There was a 9-hour network outage in New York City of large parts of the US telephone network [90]. It cost several 100 million US\$ and the cause was the incorrect interpretation of a break statement in the C programming language [73].

Pentium FDIV BUG (1994): Problems occurred with the *floating point division unit* (FDIV) of Pentium chips [91, 130]. Under certain circumstances, a flaw in this unit resulted in incorrect results. This cost an estimated 500 million US\$ and resulted in a serious image loss for the company. (Following this, chip designers invested heavily in formal verification techniques.)

Ariane 5 Disaster (1996): This is the famous crash of the Ariane-5 rocket [56, 127]. The source of the crash is believed to be in the data conversion from a 64-bit floating point to a 16-bit signed integer. It cost over 500 million US\$ and had a very negative impact on the image of space reliability.

These, along with several other less high profile problems, have led to companies increasingly employing formal verification techniques to build confidence, find bugs early, improve efficiency, etc. Yet another important aspect is that verified

software is likely to inspire more confidence in the public and so lead to increased “trust” in computational solutions and so greater opportunities for uptake/use of software in general. In the particular case of multiagent systems, this is especially important. Since *autonomy* is a central aspect of agent-based systems, public confidence will be quickly eroded if we cannot guarantee that the autonomous choices made by an agent are both safe and secure.

Consequently, we require some *specification* to describe our requirements and some *verification* process to match these requirements against a system to be analyzed. We use *logic* as our basis for both specification and verification. But why?

1. Formal logic provides a *clear, concise, and unambiguous* notation for describing systems and scenarios. Compare this to a *functional documentation sheet*, which is often written in natural language and prone to all sorts of misunderstandings.
2. The *formal properties* of logical descriptions are well understood. For example, the expressive capabilities of various logics (what can be described at all, what can be *easily* described in the logic, and what cannot) have been comprehensively explored.
3. Along with a formal logic comes a range of logical procedures that we can utilize in verification, such as *decision procedures, proof systems, model checkers*, etc. Again, most of these have well-established complexity and completeness results and often have implementations that we can take advantage of.
4. Finally, one aspect of formal logics that we find particularly useful when considering multiagent systems is the *flexibility* of logic. There are *very many* different logics, capturing many different aspects. Logics have been developed to capture time, space, belief, desire, wishes, cooperation, intention, probability, etc. [15, 43, 122]. Indeed, we can design our own logics to capture relevant, new activities.

These four properties make logic a suitable candidate. In particular, they help to devise an appropriate logic providing a *level of abstraction* close to the key concepts of the multiagent system. In addition, we can combine logics together to represent multifaceted systems. As we have seen already in this book, agents are often multifaceted and their description typically requires combinations of logics. For example, the BDI theory combines several modal logics (for beliefs, desires, and intentions) with a temporal (or dynamic) logic describing the underlying system evolution.

1.2 Limits and Relation to Other Chapters

This chapter builds on some notions introduced in Chapter 13, in particular on the running example introduced there. This example is assessed in order to show several features of verification in general. The next chapter, Chapter 15, also builds on this example and extends it slightly to illustrate specific features of the software engineering approach in agent systems. Finally, the precise notions of the logics involved in this chapter can be found in Chapter 16.

1.3 Organization of This Chapter

In Section 2 we introduce the basic terminology to be used throughout this chapter, specifically the logics for agent systems and specification languages. In Section 3 we describe how to bridge the gap between specifications and implementations. We start by presenting methods for formal verification (as opposed to just testing) in Section 4. Then we turn to approaches that have been specifically developed for the formal verification of agents and multiagent systems. There are, as yet, few¹ run-time verification tools explicitly developed for agents and so we primarily consider the *deductive* verification of agents in Section 5, the *algorithmic* verification of agent *models* in Section 6, and the direct *algorithmic* verification of agent *programs* in Section 7.

2 Agent Specification

In this section we first provide a description and the basic terminology to talk about formal, concise, and relevant specifications of agent-based and multiagent systems. While Section 2.1 deals with logics and specifications in general, we introduce in Section 2.2 several variants of temporal logics. We briefly describe some approaches based on dynamic logics in Section 2.3. An important aspect is to combine temporal logics with other logics, such as logics of knowledge and belief; we elaborate on this in Section 2.4. Finally, the running example from the last chapter is discussed in Section 2.5.

2.1 Logics of Agency and Specification Languages

Many of the logics described in Chapter 16 are used in some form or another for specifying agents or multiagent systems. Rather than surveying all the different varieties, we will outline one style, which turns out to be quite common. This

¹For one such, see [2].

typical approach involves a logical basis that is, in turn, a combination of logics [42, 52]. Primarily this is a logic describing the core abstract “state” of an agent, combined with a logic showing how such abstract states change or evolve dynamically.

As an example, let us imagine that our agent is primarily concerned with accessing, selecting, and distributing information. We might decide that the best way to formalize such an agent is in terms of a *logic of knowledge*. This will allow us to describe what the agent knows, what it knows that it knows, what it knows that others know, and so on. A standard approach to representing such knowledge is to use a multimodal logic of the **S5** style. Here, the modal necessitation operator, K_i , can be parametrized by an agent, i . Thus, $K_{Jürgen} \varphi$ means that “Jürgen knows φ ”, while $K_{Michael} \psi$ means that “Michael knows ψ ”. With such a logic we can describe a variety of interesting agent behaviors concerning knowledge, for example

$K_{Jürgen} \text{raining} \dots\dots\dots \text{Jürgen knows it is raining}$

$K_{Jürgen} K_{Jürgen} \text{raining} \dots\dots\dots \text{Jürgen knows that he knows it is raining}$

$K_{Jürgen} \neg K_{Jürgen} \text{warm} \dots\dots\dots \text{Jürgen knows that he doesn't know it is warm}$

$K_{Jürgen} K_{Michael} \text{warm} \dots\dots\dots \text{Jürgen knows that Michael knows it is warm}$

We can also consider schemata of the form $K_{Jürgen} \Phi \rightarrow K_{Michael} \Phi$ for all formulae Φ . This means that whatever Jürgen knows, Michael knows and so *Michael knows at least as much as Jürgen*.

This gives quite a strong mechanism for describing static agent knowledge [40], but it is not yet enough. We need to add a more *dynamic* dimension, allowing our agent “state” to evolve or change to a new “state.” This might be due to some action occurring, some time passing, or any other dynamic event. Thus we need a logic that captures such aspects; the primary candidates for this are *dynamic logic* [59] or *temporal logic* [36, 47, 62, 115].

Continuing our example, let us use a simple linear temporal logic [51] to describe dynamic change. Recall that typical connectives in such a logic are “ \bigcirc ”, meaning “at the next moment in time,” and “ \Diamond ”, meaning “at some future moment in time.” Such operators allow us to navigate between states at distinct moments in time. Once we combine this logic with our logic of knowledge (technically a *fusion* of the two logics [42]), we can describe statements such as

$\bigcirc K_{Jürgen} \text{warm} \dots\dots\dots \text{in the next moment, Jürgen will know it is warm}$

$K_{Michael} \Diamond \text{raining} \dots\dots\dots \text{Michael knows it will eventually be raining}$

However, while the above represents an interesting combination for describing how the knowledge within an agent evolves [40], it is typically still not enough to be able to describe truly *autonomous* agents. For this we need some way to capture the *reasons* for an autonomous agent making the choices it does; in other words, some *motivation* for doing what it does.

This leads us to a very common structure for agent theories, and so for agent specification languages, comprising

1. a logical dimension describing the underlying dynamic/temporal nature of the agents, for example, *dynamic logic* or *temporal logic*;
2. a logical dimension describing the *information* the agent has, for example, a *logic of belief* or *logic of knowledge* (as above); and
3. a logical dimension describing the *motivations* an agent has, for example, a logic of *goals*, *desires*, *wishes*, or *intentions*.

For example, the logical basis for the BDI approach comprises a temporal logic, a modal logic of belief (for the information dimension), and modal logics of desire and intention (for the motivational dimensions) [106]. Alternatively, the KARO framework (for Knowledge, Abilities, Results, and Opportunities) [88, 123] combines a dynamic logic basis with a modal logic of knowledge (information) and a modal logic of wishes (motivation).

2.2 Approaches Based on Temporal Logics

We have already mentioned some simple temporal logic statements above. In any temporal logic, one can talk about properties that are true at certain time intervals. The \mathcal{L}_{LTL} -formula $\bigcirc(\varphi \wedge \psi)$, for instance, expresses the fact that both φ and ψ hold in the next moment; $\varphi \mathcal{U} \psi$ states that φ is true at least until ψ becomes true, which will eventually be the case. The additional operators \Diamond (*sometime from now on*) and \Box (*always from now on*) can be defined as macros by $\Diamond\varphi \equiv \top \mathcal{U} \varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$, respectively. The standard Boolean connectives $\top, \perp, \vee, \rightarrow$, and \leftrightarrow are defined in their usual way.

In the next few subsections we define several variants of temporal logic that are increasingly more expressive. While **LTL** talks about linearly ordered time-points (so there is only one possible execution of the system), **CTL*** deals with branching time and thus needs new quantifiers E, A to talk about *all possible temporal paths* (in addition to talking about time-points on such paths). In **CTL*** we can model many different executions of a system. **ATL*** finally extends **CTL*** by replacing E, A by *cooperation modalities*. This allows us to deal with abilities (or strategies) of groups of agents.

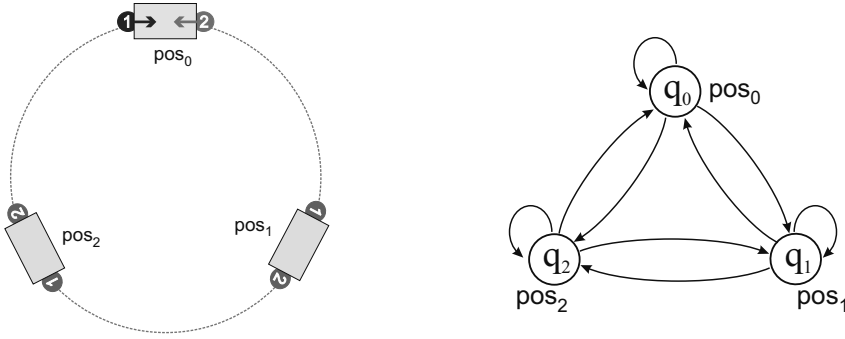


Figure 14.1: Two robots and a carriage: a schematic view (left) and a transition system \mathcal{M}_0 that models the scenario (right).

In this section we introduce the logics that we investigate later more thoroughly. We do not give precise semantics here (we refer to Chapter 16 and to [18]).

We illustrate these logics with the following example.

Example 14.1 (Robots and Carriage) *Two robots push a carriage from opposite sides (Figure 14.1). As a result, the carriage can move clockwise or counter-clockwise, or it can remain in the same place – depending on who pushes with more force (and, perhaps, who refrains from pushing). We identify 3 different positions of the carriage, and associate them with states q_0 , q_1 , and q_2 . The arrows in transition system \mathcal{M}_0 indicate how the state of the system can change in a single step. We label the states with propositions $\text{pos}_0, \text{pos}_1, \text{pos}_2$, to refer to the current position of the carriage.*

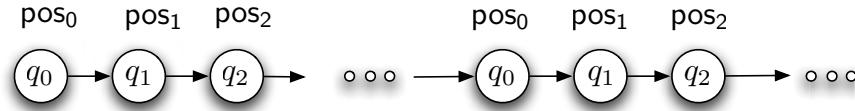
As we are investigating the precise complexity of the model checking problem later, we need to be a bit more precise. The transition system above is also called a *Kripke model*.

Definition 14.1 (Kripke Model, Path) *A Kripke model (or unlabeled transition system) is given by $\mathcal{M} = \langle St, \mathcal{R}, \Pi, \pi \rangle$ where St is a non-empty set of states (or possible worlds), $\mathcal{R} \subseteq St \times St$ is a serial transition relation on states, Π is a set of atomic propositions, and $\pi : \Pi \rightarrow 2^{St}$ is a valuation of propositions. A path λ (or computation) in \mathcal{M} is an infinite sequence of states that can result from subsequent transitions, and refers to a possible course of action. For $q \in St$ we use $\Lambda_{\mathcal{M}}(q)$ to denote the set of all paths of \mathcal{M} starting in q and we define $\Lambda_{\mathcal{M}}$ as $\bigcup_{q \in St} \Lambda_{\mathcal{M}}(q)$. The subscript “ \mathcal{M} ” is often omitted when clear from the context.*

2.2.1 LTL

Definition 14.2 (Language \mathcal{L}_{LTL} [99]) The language \mathcal{L}_{LTL} is given by all formulae generated by the following grammar, where $p \in \Pi$ is a proposition: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathcal{U} \phi \mid \bigcirc\phi$.

The logic is termed *linear-time* because formulae are interpreted over infinite *linear* orders of states. It allows us to reason about a particular computation of a system: there is always exactly one *next* time moment. A model is an infinite sequence of states, such as



This describes a computation that consists of the same 3 states occurring again and again: the carriage moves forever in a cycle from position 0 via position 1 to position 2.

What about the formulae $\bigcirc pos_1$, $\diamond pos_2$, and $\square \diamond pos_2$? They are all true in this model (we evaluate these formulae in the first state of the series).

There are several important classes of property that can be expressed easily in **LTL**:

Reachability: A particular state is reachable from the present state.

Safety: A (bad) property will never be satisfied at any future state.

Liveness: A (good) property will eventually be satisfied by some state in the future.

Deadlock freedom: A dead-end state will never be reached.

The formula $\square \diamond pos_1 \rightarrow \diamond pos_2$ expresses the statement: “If it is infinitely often the case that pos_1 is true then at some point in the future pos_2 will be true.” Formulae such as this, especially when they incorporate subformulae such as $\square \diamond pos_1$, are very useful for representing *fairness* properties [50].

Fairness: If something is attempted a certain (either finite or infinite) number of times, then it will eventually occur a certain (finite or infinite) number of times.

LTL can be viewed in many ways [48], for example, as a decidable fragment of first-order logic (see one of the exercises).

There is one important observation to make here: models of **LTL** are infinite paths (see one of the exercises where we show how to make finite paths infinite).

by adding loops). How can we represent such an infinite path? Clearly we need a finite representation of it. In fact, we use the Kripke model \mathcal{M} (see Definition 14.1) and define

$$\mathcal{M}, q \models_{\text{LTL}} \varphi$$

to mean that φ is true *on all resulting paths starting in q in the model \mathcal{M}* . Note again that **LTL** formulae are always evaluated on one individual path: in the definition just presented we consider, in addition, all possible paths the system *could* take.

Clearly, $\bigcirc \text{pos}_1$ is not true at q_0 in \mathcal{M} as there is a path from q_0 where pos_1 is *not* true (using the transition that leads into q_0 or the one that leads into q_2).

2.2.2 CTL and CTL*

The logic **CTL*** [38] explicitly refers to patterns of properties that can occur along a particular temporal path, *as well as* to the set of possible time series, and thus extends **LTL**. This extra dimension is handled by *path quantifiers*: E (*there is a path*) and A (*for all paths*) where the A quantifier is defined as the macro: $A\varphi \equiv \neg E\neg\varphi$. Hence, the language of **CTL***, $\mathcal{L}_{\text{CTL}^*}$, essentially extends \mathcal{L}_{LTL} by adding the ability to quantify over different paths.

Definition 14.3 (Language $\mathcal{L}_{\text{CTL}^*}$ [38]) *The language $\mathcal{L}_{\text{CTL}^*}$ is given by all formulae generated by the following grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid E\gamma$ where $\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \gamma \mathcal{U} \gamma \mid \bigcirc\gamma$ and $p \in \Pi$. Formulae φ (respectively, γ) are called state (respectively, path) formulae.*

For example, $E\Diamond\varphi$ states that there is at least one path on which φ holds at some (future) moment in time.

We now consider the formulae $E\Diamond\text{pos}_1$, $A\Diamond\text{pos}_1$, and $\neg A\Box\Diamond\text{pos}_2$ and evaluate them in our Kripke model and in state q_0 . While the first one is true (from state q_0 it is possible to reach position 1 in the future via a computation), the second is not (because this is not possible for *all paths*). The third formula is also true (as its negation is false).

Finally, we mention that there is a fragment² of **CTL***, called **CTL** [23], which is strictly *less expressive* but has significantly *better computational properties*. The language is restricted so that each temporal operator must be directly preceded by a path quantifier. For example, $A\Box E\bigcirc p$ is a \mathcal{L}_{CTL} -formula whereas $A\Box\Diamond p$ is not. The complexity of these logics is investigated in Section 4.3.

²To be precise, it is not just a fragment, i.e., a subset, of the language. In order to make sure that all operators are definable, the definition of the language requires some care.

So **CTL** formulae are directly evaluated in a Kripke model and we can express “on all paths it is always true that there exists a path such that ...”: $A\Box E\Box \text{pos}_1$. This cannot be expressed in **LTL**.

2.2.3 ATL and ATL*

The logic **ATL*** [6, 7] (Alternating-time Temporal Logic) is a generalization of **CTL***. In \mathcal{L}_{ATL^*} the path quantifiers E, A are replaced by *cooperation modality* $\langle\langle A \rangle\rangle$ where $A \subseteq \mathbb{A}gt$ is a team of agents. Thus, formula $\langle\langle A \rangle\rangle\gamma$ expresses the property that

the team of agents, A , has a *collective strategy* to enforce γ .

Thus we can formulate statements of the form *it is possible that a certain group of agents is able to bring about a certain formula ϕ* . This is to be understood as, whatever all other agents are doing, this group can make sure that ϕ holds. The recursive definition of the language syntax is given below.

Definition 14.4 (Language \mathcal{L}_{ATL^*} [6]) *The language \mathcal{L}_{ATL^*} is given by all formulae generated by the following grammar: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle A \rangle\rangle\gamma$ where $\gamma ::= \phi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \gamma \mathcal{U} \gamma \mid \bigcirc\gamma$, $A \subseteq \mathbb{A}gt$, and $p \in \Pi$. Formulae ϕ (respectively, γ) are called state (respectively, path) formulae.*

For example, the formula $\langle\langle A \rangle\rangle\Box\Diamond p$ expresses the statement that coalition A can *guarantee* that p is satisfied infinitely many times (ever and ever again in the future).

The semantics for \mathcal{L}_{ATL^*} is defined over a variant of transition systems where transitions are labeled with combinations of actions, one per agent.

Definition 14.5 (Concurrent Game Structure) *A concurrent game structure (CGS) is a tuple $\mathcal{M} = \langle \mathbb{A}gt, St, \Pi, \pi, Act, d, o \rangle$, which includes a non-empty finite set of all agents $\mathbb{A}gt = \{1, \dots, k\}$, a non-empty set of states St , a set of atomic propositions Π and their valuation $\pi : \Pi \rightarrow 2^{St}$, and a non-empty finite set of (atomic) actions Act . Function $d : \mathbb{A}gt \times St \rightarrow 2^{Act}$ defines non-empty sets of actions available to agents at each state, and o is a (deterministic) transition function that assigns the outcome state $q' = o(q, \alpha_1, \dots, \alpha_k)$ to state q and a tuple of actions $\langle \alpha_1, \dots, \alpha_k \rangle$ for $\alpha_i \in d(i, q)$ and $1 \leq i \leq k$, which can be executed by $\mathbb{A}gt$ in q . We also write $d_a(q)$ instead of $d(a, q)$.*

So, it is assumed that all the agents execute their actions synchronously: the combination of the actions, together with the current state, determines the transition to the next state of the system.

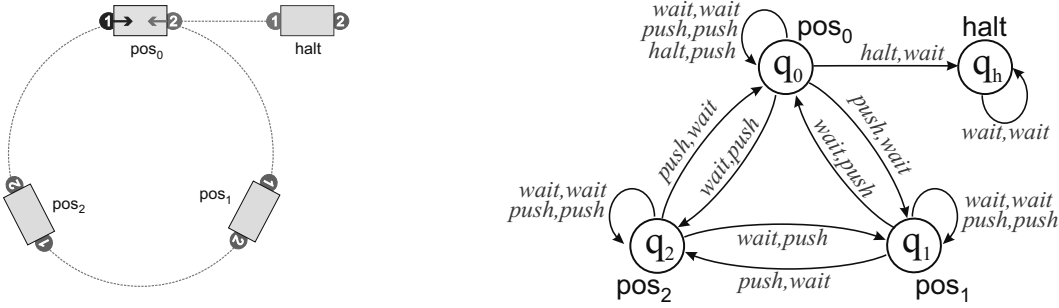


Figure 14.2: Two robots and a carriage: a refined version of our example and a concurrent game structure (CGS).

Example 14.2 (Robots and Carriage, ctd.) Consider the modified version of our example, as shown in Figure 14.2. What about the following formulae?

1. $\text{pos}_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg \text{pos}_1$
2. $\text{pos}_0 \rightarrow \langle\langle 1, 2 \rangle\rangle \bigcirc \text{pos}_1$
3. $\text{pos}_0 \rightarrow \neg \langle\langle 2 \rangle\rangle \bigcirc \text{pos}_1$

The first one expresses that when one is in position 0, then agent 1 alone can ensure that position 1 will never be reached in the future. Indeed, agent 1 should not push while in position 0, but should do so in position 2 (otherwise it might end up in position 1 if agent 2 does not push). The appropriate strategy is $s_1(q_0) = \text{wait}$, $s_1(q_2) = \text{push}$ (the action that we specify for q_1 is irrelevant). The second formula above says that both agents can make sure that they reach position 1 in the next step. Indeed, agent 1 should push and agent 2 should refrain from doing so. Clearly they need to work together. The final formula above says that, in position 0, it is not possible for agent 2 on its own to ensure that the carriage ends up in position 1 in the next time-point. Indeed, the next position of the carriage depends on the action carried out by agent 1.

Can cooperation modalities be compared to *diamonds* or rather to *boxes*? The answer is “neither”! They are essentially *combinations of both*. Their structure can be described by “ $\exists\forall$ ” (we ask for the *existence* of a strategy of the proponents that is successful against *all* responses of the opponents).

As for **CTL**, we define the language \mathcal{L}_{ATL} that restricts \mathcal{L}_{ATL}^* in the same way as \mathcal{L}_{CTL} restricts \mathcal{L}_{CTL}^* : each temporal operator must be directly preceded by a cooperation modality.

As an example, the formula $\langle\langle A \rangle\rangle \Box \Diamond p$ is *not* a formula of \mathcal{L}_{ATL} (it contains two consecutive temporal operators). \mathcal{L}_{ATL} does not allow us to express abilities related to, for example, fairness properties, where we often need this ‘ $\Box \Diamond$ ’ combination. Still, many interesting properties are expressible. We can express the fact that agent a has a strategy that permanently takes away the ability to enforce $\bigcirc p$ from coalition B : $\langle\langle a \rangle\rangle \Box \neg \langle\langle B \rangle\rangle \bigcirc p$. This is clearly quite powerful.

As is the case for **CTL** and **CTL***, the choice between \mathcal{L}_{ATL}^* and \mathcal{L}_{ATL} reflects the trade-off between expressiveness and practicality.

2.3 Approaches Based on Dynamic Logic

In an attempt to make modal logic more applicable to the specification of computer programs, *dynamic logic* was introduced [58, 103]. Dynamic logic extends modal logic by allowing modal operators to be parameterized by programs. So, in dynamic logic, “ $[\alpha]\psi$ ” means that the formula ψ is true in every state in which program α terminates while “ $\langle\alpha\rangle\varphi$ ” means that φ is true in at least one state in which program α terminates.

Along with the usual propositional operators for combining formulas, dynamic logic includes operators to combine programs. This gives the ability to build complex programs from a simple set of atomic programs. Typical program operators are “ \cup ”, “ $;$ ”, and “ $*$ ”, representing non-deterministic choice, sequential composition, and non-deterministic iteration, respectively. Often, dynamic logics also include the test operator, “ $?$ ”. The informal descriptions of these four operators are:

- $\alpha \cup \beta$ — is a program that does α or β non-deterministically,
- $\alpha; \beta$ — is a program that does α followed by β ,
- α^* — is a program that does a non-deterministic (unbounded) number of α ’s sequentially,
- $p?$ — is a program that aborts if p is false, but continues if p is true.

Dynamic logic has proved very popular and useful across many areas. In addition to application, much research has gone into categorizing the relative power and complexity of dynamic logics that are allowed differing sets of program operators [59]. The semantics of dynamic logic, particularly simple versions such as *propositional dynamic logic* (PDL), are straightforward, essentially following that of multimodal logics. Along with the simplicity of this semantics for PDL, axiomatizations of PDL tend to be equally as simple as decision procedures, at least for basic dynamic logics.

Perhaps one of the most appealing aspects of dynamic logic is its close links to Hoare logic, and partial correctness assertions in general [95]. Thus, $\{p\}\alpha\{q\}$ in Hoare logic can be expressed as $p \Rightarrow [\alpha]q$ in PDL, while termination of a program

α can be expressed by $\langle\alpha\rangle\top$. These aspects make dynamic logic a viable alternative to temporal logic in providing the basis for agent specification formalisms.

2.4 Combinations

As we have said already, the key logical foundation for agent specification is the *combination* of logical domains [78]. Typically, formalisms for agent specification consist of a temporal/dynamic basis combined at least with a logic of information (e.g., knowledge or belief) and usually at least one logic of motivation (e.g., goals, intentions, desires, wishes). In the following subsections we highlight useful and popular combinations.

2.4.1 BDI

As the BDI approach to representing and implementing rational agents [107] is the predominant one within the area, this is described in detail in several other chapters. We will not recap all this but just mention how the *formal* BDI framework fits in with this section. As described above, the core of any agent formalism is some dynamic base, and there are variations of the BDI approach using either dynamic logics or temporal logics. On top of this, we usually need a logical framework for the *information* the agent has and, again, the BDI approach uses *beliefs* captured logically by **KD45** modal logic. For rational agents that must have some explicit *motivation* for making their choices, an additional logical dimension for this is required. Indeed, in the BDI approach, there are *two* varieties of motivation: *desires*, representing long-term goals; and *intentions*, representing goals that the agent is actively undertaking. Both of these are formally represented by (distinct) **KD** modal logics. The combination of all these logical dimensions provides a logical basis for specifying BDI agents [106], in particular, a basis upon which the key aspect of *deliberation* can be described. In the BDI approach, deliberation consists of two aspects: deciding which *desires* will be selected to become *intentions*; and deciding the best way (plan) to achieve these *intentions*.

2.4.2 KARO

The KARO approach (*Knowledge, Abilities, Results, and Opportunities*) [88, 123] is based on dynamic logic. Essentially it is a formal system aimed at specifying and reasoning about the behavior of rational agents.

In the basic framework [123], an agent has *knowledge*, usually expressed through an **S5** modal logic. The dynamic logic basis provides the action language, thus allowing analysis of whether the agent is *able* to perform a certain

action or has the *opportunity* to perform it. Beyond this the *motivational* aspect of a rational agent is described via a **KD** modal logic of *wish*.

2.4.3 Dynamic Epistemic Logic

Dynamic epistemic logic (DEL) [120] is a combination of **S5** multimodal logic (providing an informational dimension) together with a standard dynamic logic (providing the underlying dynamic framework). A typical use is where some action, for example, a *public announcement*, forces a dynamic change in the knowledge of agents [119, 121]. Note that there are no explicit *motivational* attitudes here.

2.5 Sample Specifications

We here provide some examples of formal specifications of agent behavior, particularly relating to some of the examples considered in Chapter 13.

Consider a simple *contract-net* protocol between agents and begin with just the *seller* agent. A naive requirement for this seller might be that the seller will accept the first proposal it receives, e.g.,

$$received(offer) \Rightarrow \bigcirc accept(offer).$$

Of course, it may well be that the *offer* is not acceptable, so

$$(received(offer) \wedge acceptable(offer)) \Rightarrow \bigcirc accept(offer)$$

and, quite possibly, the acceptance will take some time:

$$(received(offer) \wedge acceptable(offer)) \Rightarrow \Diamond accept(offer).$$

However, this is quite a strong requirement. More likely, we will require the agent to accept *one* of the reasonable offers, and so, using some additional first-order syntax,

$$\begin{aligned} & [\exists O_1. received(O_1) \wedge acceptable(O_1)] \\ & \Rightarrow \\ & [\exists O_2. received(O_2) \wedge acceptable(O_2) \wedge \Diamond accept(O_2)]. \end{aligned}$$

An alternative way to capture such a specification is to record the incoming offers within the agent's *knowledge*, and then

$$K_{Seller} best(offer) \Rightarrow \Diamond accept(offer).$$

More typically, within an agent working in the real world, we will not have *certainty* about information or environmental constraints, and so will use *belief*. In addition, since we often do not know what might *stop* the agent from accepting an offer, we often require that the seller agent *intends* to accept the best offer. So:

$$B_{Seller} best(offer) \Rightarrow I_{Seller} accept(offer).$$

Separately, we might require that

$$(I_{Seller} accept(x) \wedge no_problem_with(x)) \Rightarrow \Diamond accept(x).$$

Now, we can also specify the multiagent interaction, for example

$$send(seller, offer) \Rightarrow \Diamond K_{Seller} received(offer)$$

and by combining all the specifications together, we can (ideally) describe the chain of steps to achieve

$$\exists O. \Diamond accept(O).$$

Now we consider the scenario outlined in Chapter 15 and Section 6 of Chapter 13 involving two robots collaborating in a manufacturing plant. As in the contract-net example above, there are a number of different aspects we might require. For example, *robot₁* should ensure that

$$\Diamond on(A, table)$$

and also

$$\Diamond on(B, A).$$

Clearly, we eventually require that

$$\Diamond unloaded(completed_ABC)$$

but this specification can be decomposed to multiple steps, for example

$$\begin{aligned} \left[\begin{array}{l} K_{robot_1} in_front_of(robot_1, C) \wedge \\ K_{robot_1} in_front_of(robot_1, AB) \wedge \\ table_rotates \end{array} \right] &\Rightarrow \Diamond \left[\begin{array}{l} K_{robot_2} in_front_of(robot_2, C) \wedge \\ K_{robot_2} in_front_of(robot_2, AB) \end{array} \right] \\ \left[\begin{array}{l} K_{robot_2} in_front_of(robot_2, C) \wedge \\ K_{robot_2} in_front_of(robot_2, AB) \end{array} \right] &\Rightarrow \bigcirc K_{robot_2} in_front_of(robot_2, ABC) \\ \left[\begin{array}{l} K_{robot_2} in_front_of(robot_2, ABC) \wedge \\ unload(ABC) \end{array} \right] &\Rightarrow \Diamond unloaded(ABC) \end{aligned}$$

And so on. An important aspect to notice here is that the overall system requirement is dependent on the appropriate actions of the environment. Specifically, that “*table_rotates*” is true at relevant times. Typically, we might require $\Box \Diamond \text{table_rotates}$, i.e., that the table rotates infinitely often.

This leads us to consider the cooperation between the robots required in order to achieve the (completed and) unloaded part “ABC”. Intuitively, we might expect a requirement such as

$$\langle\langle \text{robot}_1, \text{robot}_2 \rangle\rangle \Diamond \text{unloaded}(ABC).$$

However, as we have seen, this is not the case and we must take into account the movements of the table. Actually, we can consider the table to be a separate agent since it can choose to rotate when it likes. Given this, we might require

$$\langle\langle \text{robot}_1, \text{robot}_2, \text{table} \rangle\rangle \Diamond \text{unloaded}(ABC)$$

and so the two robots and the table can together ensure that the “ABC” part is completed and unloaded. Finally, we note that the table can, in principle, stop the part being made:

$$\langle\langle \text{table} \rangle\rangle \Box \neg \text{completed}(ABC)$$

since the table can choose to move at exactly the wrong places.

3 From Specifications to Implementations

We have seen how a logical formalism can be used to specify agent behavior. But there remains a gap between such a specification and an actual implemented agent system. As in standard formal methods there are a number of ways to bridge this gap.

3.1 Toward Formal Verification

In subsequent sections we will consider how to generate an agent implementation from an agent specification using formal development methods such as refinement, synthesis, or direct execution. However, as we will see in Section 4, the most likely use of formal specifications is as a formal requirement that we can measure implementations against. In most cases such implementations will be developed by informal approaches, such as traditional software engineering methods. Once we have an agent, or multiagent system, developed in this way, it is important that we are able to assess how well this implementation matches our formal requirement. Automated and effective techniques for achieving this will

be the main focus of our discussion in Section 4 and beyond. Before that, however, we look at several ways to generate an agent implementation from a logical specification.

3.2 Refinement

Given some logical specification of agent behavior, φ_S , then we might choose to *refine* this to a new, perhaps more detailed, specification. As we move toward a “real” implementation we would like to increasingly be specific about the agent’s behavior. So, while φ_S might be quite vague and high level, we would aim xxxxxxxx for the refined specification, say φ_R , to describe behaviors that still correspond to some of those in φ_S but likely remove some of those we now consider irrelevant. Thus, it is typical (and expected) that $\vdash \varphi_R \Rightarrow \varphi_S$. So, we know that all implementations satisfying φ_R will also still satisfy φ_S , though there may well be some implementations allowed by φ_S that are now disallowed by φ_R . Two things are important here:

1. whatever logical properties we established for φ_S can, because we know that $\varphi_R \Rightarrow \varphi_S$, also be established for φ_R ; and
2. φ_R is more detailed, more deterministic, or at least closer to a possible implementation of the agent.

Thus, we can develop a series of refinements, $\varphi_{R_1}, \varphi_{R_2}, \varphi_{R_3}, \dots$, successively moving us toward an implementation in a formally defined way [19, 89]. While this approach is well-known in traditional formal methods, it can also be used for agent specifications. Yet, there still remains the problem of getting from a logical specification, say φ_{R_i} , to an actual agent implementation; we will consider these aspects in the subsequent sections.

3.3 Synthesis

Ideally, we would like to automatically *synthesize* an agent program directly from an agent specification. This sounds ideal, especially if we can guarantee that the agent will definitely implement its specification. This is, of course, a very appealing direction in traditional formal methods but has some underlying difficulties [84]. A typical approach is to synthesize a finite-state automaton from a logical (usually temporal) specification [100, 101]; and though in some cases this can be automatic and effective, in many situations the complexity of undertaking this is very large. Thus, the underlying synthesis problem even for a system of two very simple agents may well be quite complex (2-EXPTIME). So, as yet, such approaches are impractical. However, current work looking at both reduced

scenarios and at *bounded* search for an implementation [98, 109] have promise both in the non-agent and agent cases.

3.4 Specifications as Programs

A formal specification essentially characterizes a set of models of the entity being specified. In the case of agents, a logical agent specification describes a set of agent executions that satisfies the specification. So, if we have some process for extracting one (or more) of these models/executions from the specification, then this effectively gives us a way of implementing the formal specification.

Of course, you have seen this before. Logic programming, primarily in the form of PROLOG [114], provides a mechanism for trying to build a model (execution) of a set of Horn clauses. Indeed, we could use many other methods for model-building from a set of Horn clauses [75]. If we wish to do something similar for agent specifications, then we must invoke suitable model-building procedures for the logics underlying these specifications. Fortunately, the basis for many agent specifications is *linear temporal logic*, and the models of this logic are linear sequences of states, which correspond directly to program executions (see Section 2.2 earlier).

How do we go about building models from temporal specifications and then extending this to agent specifications? An obvious first step is to extend the *resolution* approach, which is central to logic programming for the temporal logic case. Unfortunately, this is quite complex and, sometimes, gives counterintuitive results. For example, if we build a model/execution for a set of temporal Horn clauses using an extension of SLD-resolution for such clauses, then we might generate the first parts of the model/execution from the future all the way back to the present.

In spite of this, such temporal languages have been developed, most notably TEMPLOG [1, 14] and CHRONOLOG [92, 93], both of which execute a subset of temporal Horn clauses using TSLD-resolution, an extension of SLD-resolution. Perhaps more relevant to agent implementation, CHRONOLOG(Z) is a more recent version, which has been shown to be useful in parallel contexts [80].

An alternative approach, and one which builds the underlying temporal models in the correct order, i.e., from the beginning onward, is the METATEM approach [13, 49]. Since this was described in detail in Chapter 13, we just recap the essential features. Again we wish to execute logical specifications in order to build a model that satisfies the specification. In the basic METATEM case [13], a temporal specification is executed, but contrary to the TEMPLOG approach, a lightweight *forward chaining* procedure is used to build an execution sequence that is a model for the temporal specification. A particular feature of the way models are built, called the “*Imperative Future*” approach [13], is that they are

built *from the beginning*; the model is constructed step by step, starting from the initial state. In the basic case this is *complete* in that the temporal specification for an agent can be executed if, and only if, the specification is satisfiable.

As we have seen, however, a temporal specification on its own is not enough and, consequently, METATEM has been extended and developed over the years. The basic specification is extended with *beliefs*, which provide the information the agent decides upon. An interesting aspect of these beliefs is that a form of resource-bounding can be captured by considering the depth of nesting allowed in reasoning about such beliefs [44]. In addition, motivations are developed. Indeed two varieties are considered: the temporal “ \diamond ” modality, which provides a *very* strong motivation since the semantics of “ $\diamond g$ ” require that g will *definitely* happen; and the combination “ $B\diamond$ ”, where “ B ” is the belief operator, which provides a weaker motivation for the agent. Overlaying all this is a framework, called Concurrent METATEM, which takes a set of such agents, each executing its own formal specifications asynchronously, and allows them to communicate, cooperate, and self-organize [48]. See Chapter 13 for more details.

4 Formal Verification

Once we decide to analyze a system with respect to a formal property, there are a number of ways to achieve this. One particularly popular approach is to carry out *testing* [8, 16, 61]. Here, the system/program is executed under a specific set of conditions and the execution produced is compared to an expected outcome. The skill in testing is to carry this out for enough different conditions so that the developer can be relatively confident that the program/system is indeed correct.

While testing is, of course, very useful, it only examines a subset of all the possible executions. What if we want to be sure that the logical specification is met, whichever way the program/system executes? Assessing whether this is the case or not is the core of *formal verification*.

4.1 What Is Formal Verification?

The Latin origin of “verification” is *veritas facere*: “making something true.” A more recent dictionary definition is

Verification: *additional proof that something that was believed (some fact or hypothesis or theory) is correct* [31]

Moving on to “formal verification,” we find,

Formal Verification: *the act of proving or disproving the correctness of a system with respect to a certain formal specification or property* [30]

As described above, we essentially want to examine *all* possible executions of our system/program in order to assess whether they all satisfy our formal requirements. If there is only a finite set of different executions, then we might be able to enumerate them all and check their properties; if there is an *infinite* number of possible executions, then we must do something more sophisticated. We will briefly describe some alternative verification approaches below before specifically looking at these in an agent context in Sections 5, 6, and 7.

4.2 Deductive Verification

If we have a complex system, with an infinite (or at least very large) number of possible executions, then a typical approach is to use some logical description to capture the behavior of our system. This logical formula, say “*Sys*”, is likely to have been devised from the formal semantics of the system/program. If we then have our formal specification of our requirements, say *Req*, given in the same logic, then the aim of deductive verification is to *prove* $\vdash \text{Sys} \Rightarrow \text{Req}$. If this is proved, then all executions, characterized by *Sys*, satisfy the required property, *Req*.

Of course, logical proof can be difficult. If we are lucky, *Sys* and *Req* can be described in a quite simple logic and the formula $\text{Sys} \Rightarrow \text{Req}$ can be decided in a fast and automated way. More likely, either the proof process cannot be fully automated or, even if it can, it is likely to be *very* slow. In the former case, it is often essential to invoke human intervention and so utilize semi-automated theorem provers such as *Isabelle* [96] and *PVS* [94]. In the latter case, more sophisticated heuristics and abstractions are typically used.

4.3 Algorithmic Verification

As described above, if we want to establish some property of all executions of a system, and if there is only a finite number of such executions, then an obvious approach is to enumerate the executions and check the property on each in turn. While this is a gross simplification, it is essentially the basis of the *model checking* approach to algorithmic verification, which has been so successful and influential [10, 12, 22, 129]. Here, a mathematical *model* of the system in question is produced such that the model captures all relevant system executions. (Such a model is typically generated from an *operational semantics* for the system.) Then the *model checker* exhaustively checks that all paths through the model (and, therefore, running through the system) satisfy the requirement. Within this context the formal requirement is often given in a form of *temporal logic*, as introduced in Section 3.4. If all paths satisfy the logical requirement, then the system is reported as being correct with respect to its specification. If, however, a path fails

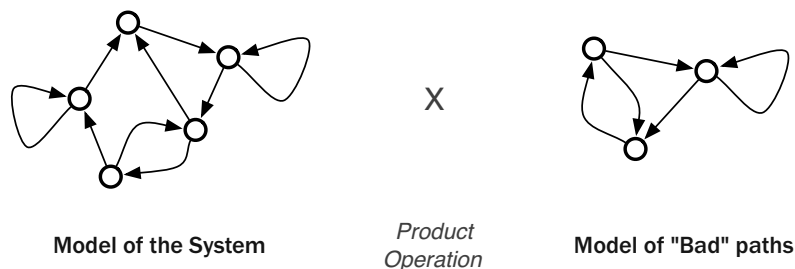


Figure 14.3: Automata-theoretic view of model checking.

to satisfy the specification, then this gives us an execution that violates the formal requirement.

The typical way of visualizing this is in terms of *finite-state automata*, in particular *Büchi automata*. The essential idea here is to capture *all* the possible executions of the system to be verified as a Büchi automaton (a finite-state automaton with infinite runs) and generate a separate Büchi automaton describing all *bad* runs, i.e., executions that do *not* satisfy the property being verified. Then we take the synchronous *product* of these two Büchi automata [113, 125] (see Figure 14.3). If the product automaton is empty, then there is no sequence that is a legal run of the system while at the same time satisfying the “bad” property. However, if the product automaton is non-empty, then there is indeed a sequence that is a legal run of the system while at the same time satisfying our “bad” property. This highlights a failing run of the system.

The model-checking approach has been extremely successful, not only in analyzing hardware systems [70] and protocols [64], but increasingly in software systems [12, 126]. While the basic idea is quite simple, the success of the technology is, to a large part, due to the improvements in implementation and efficiency that have occurred over the last 25 years. As well as a characterization in terms of automata [113], *on-the-fly* [53], *symbolic* [87], and SAT-based [102] techniques have all improved the efficacy of model checkers.

The “on-the-fly” approach will be particularly interesting with respect to our later descriptions, and so we will say a little more about this here. Recall from Figure 14.3 that the basic automata-theoretic view of model checking involves constructing the product of two Büchi automata. In many practical cases, this product turns out to be *much* too large to realistically construct. So, rather than constructing the actual product automaton, the idea with the “on-the-fly approach” is to explore paths through this product automaton without actually constructing it! This is achieved by exploring the two automata in parallel (see Figure 14.4).

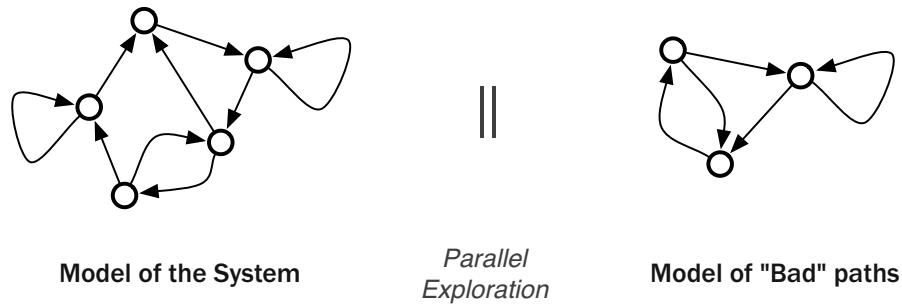


Figure 14.4: “On-the-fly” exploration of the product automaton.

To see how this works, recall that a run of the product automaton must, simultaneously, be a run of each of the automata separately. Thus, we explore the “system” automaton, ensuring that every transition we take is mirrored by a transition in the “bad” automaton. We keep exploring this pair synchronously until either a path has been found that satisfies both, or until the exploration of the “system” automaton can go no further. In the former case, we have found our “bad” path; in the latter case, we roll back our execution to any previous choice point in the “system” automaton and continue exploration. If we have explored all possible paths through the “system” automaton and none of them have yielded a run of the “bad” automaton, then we can assert that no execution has the “bad” property. Notice that in order to be able to achieve this, the model-checking implementation needs to have (a) a way to synchronously step through two representations, and (b) a mechanism for backtracking the execution. The predominant model checker exhibiting this technology is the SPIN model checker [64], though we will see how this approach is used in Section 4.4.

Finally, while we have described model checking as a technique for analyzing *finite-state* systems, there has been considerable work in providing coherent *abstraction mechanisms* to reduce infinite-state systems down to a finite-state form suitable for model checking (see [20, 21]).

4.4 Program Verification

An important development, in recent years, has involved refining the model-checking approach. Traditionally a “model” of the executions of the system is built and then that model is explored and checked with respect to the property. However, if the system we are to verify is a program, then why not use the program itself as the model? In this approach, often termed “software model check-

ing” or “program model checking,” a logical property is directly checked against the program code [65, 66, 126].

In essence this approach is closely related to the “on-the-fly” technique from Figure 14.4 above. If we recall what is needed for this, it is (a) a way of synchronously stepping through a program at the same time as checking a property, and (b) a mechanism for backtracking execution of the program. So, as long as we have implementation technology that allows these two, we can implement program verification. The program to be checked is run and the execution is dynamically assessed against the requirement. Once checked, the program is forced to explore an alternative execution path, which is again checked. And so on. This has led to the development of model checkers for various high-level languages such as JAVA and C [63, 126, 128]. In particular, the JAVA PATHFINDER system implements this approach for model checking JAVA programs [126, 128]. To allow it to achieve (a) it utilizes a modified JAVA virtual machine that can backtrack, and to achieve (b) it uses synchronous *listener* threads. We will see in Section 7 how JAVA PATHFINDER forms the basis for a model-checking system for JAVA-based rational agent programs.

4.5 Runtime Verification

Once we have the idea that a form of model checking can be invoked directly on the program, by forcing it to run numerous times, then this leads us to thinking about *run-time verification* [60]. The idea here is to use (lightweight) formal verification technology to check executions *as they are being created*. In this way, errors are also spotted at run-time. Referring back to Figure 14.4, we see that all the possible program executions are checking against a parallel automaton looking for “bad” runs. Now we can take this automaton and use it to check the current execution as it is being created. In this way we can monitor the execution and recognize when a quite complex error condition has occurred (see Figure 14.5).

5 Deductive Verification of Agents

As we saw earlier, the essence of deductive verification is to provide a logical description capturing the full behavior of our agent, say “Ag.” Then, if we wish to verify some property of our agent, such as *the agent will eventually terminate*, we describe this property as another logical formula, *Req*, and then attempt to *prove* $\vdash Ag \Rightarrow Req$. If we succeed with this proof, then *Req* is true for all possible behaviors of the agent.

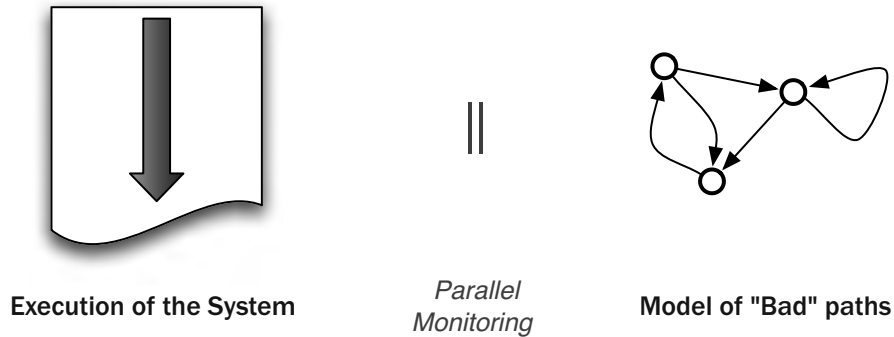


Figure 14.5: A general view of run-time model checking.

5.1 The Problem

While this deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should “Ag” be described in, and how do we actually generate “Ag”?
2. What logic should *Req* be described in, and can we be sure this is sufficient to allow us to say what we want?
3. Given *Ag* and *Req*, will it be *possible* to prove $\vdash Ag \Rightarrow Req$? And will we be able to automate this proof process?
4. If we fail to prove $\vdash Ag \Rightarrow Req$, then what does that mean?

Some of these are, of course, quite difficult and fundamental questions, but let us start to describe answers to some of the above, beginning with (1). For any formal method, we need some variety of formal *semantics* that provides a formal (often logical) representation of all the behaviors of the agent. If agents are described in terms of enhanced finite-state machines, then this is fairly straightforward. If, however, we have an agent program, then we require a semantics for the agent programming language. In the case of deductive verification we consider here, we specifically need a *logical* semantics for the agent programming language. As with traditional formal methods, other varieties of formal semantics, notably *operational semantics*, are more popular. Indeed, there are few agent languages with logical semantics; some exceptions are [4, 33, 46, 116, 117]. In general, it is easier to develop an operational semantics for an agent programming language

(especially since such an operational semantics can form the basis for language implementation) than to develop a logical semantics; consequently, much more verification work has occurred via operational semantics. Nevertheless, there are quite a few areas where agent verification based on some form of proof has been achieved, and we will mention a selection of these below.

Concerning some of the other questions mentioned above, the decision about what logical basis to use must clearly be driven by the requirements of both the logical semantics (i.e., what logic the semantics is provided in) and the formal requirements (i.e., what logic allows us to state the questions we wish to ask). As we saw earlier, some logical basis combining a temporal/dynamic dimension with at least a knowledge/belief dimension and probably a motivational dimension is often used.

5.2 Direct Proof

Some examples of using direct proof methods for agent logics are given below.

2APL, 3APL: In [4] the authors consider a fragment of 3APL and define a series of propositional dynamic logics that can be used to prove safety and liveness properties of programs in this fragment under different deliberation strategies. This is done by relating the operational semantics of programs to models in the appropriate logic. It requires, among others, the axiomatization of fully interleaved strategies.

METATEM: As described earlier, METATEM is a little unusual, having no explicit motivational dimension but using combinations of temporal and belief operators to achieve such “goals.” Consequently, we might prove some (simple) properties of METATEM programs using deductive proof methods for temporal logics of belief [35, 45].

However, as we will see in Section 6, this is non-standard – and true “BDI-like” agents usually require a logic with some motivational dimension, such as *intentions* or *goals*. Although Schild [110] showed how Rao and Georgeff’s BDI logic [106] could be translated to the μ -calculus [76], this target is quite complex and intricate.

IMPACT: Agents are specified in IMPACT through *agent programs*. These have the form of rules and are treated as clauses with negation in logic programming. Therefore, the semantics is given by the well-known fix-point semantics (the least Herbrand model in the case of Horn clauses or stable semantics in the case of rules with negation-as-failure). Whereas the basic language of IMPACT does not allow

us to formalize *mental attitudes* and *temporal* and *probabilistic reasoning*, these features have been investigated elsewhere (see [33, 34, 116]) and can be modeled with annotated logic programs.

Golog and SITCALC: The Cognitive Agent Specification Language (CASL) [112] is, like GOLOG, based on the *situation calculus*, but is extended with knowledge and goal operators. Alongside this, the authors described CASLve, a verification environment for CASL, which translates a CASL specification into a problem for the PVS verification system [94].

5.3 Use of Logic Programming

If our agent language is based on *logic programming*, then there are likely to be several advantages concerning the questions highlighted in Section 5.1. First, since it is traditional that *declarative* as well as *operation* and *fixed-point* semantics are provided for logic programming languages, then generating a logical formula describing the full agent behavior is likely to be more straightforward. A second potential advantage is that as the underlying execution mechanism is essentially deductive (often some variety of SLD-resolution), then we might use the execution system itself to carry out the deductive verification we are interested in. In some cases this can be expressive and efficient.

However, it is important to note that often not all the aspects we might wish for from “BDI-like” languages are present. Here we list three approaches, where agent attitudes are translated to some variant of computational logic.

Abductive logic programming [72]: Here, standard logic programs are extended with *abducible* predicates. These are predicates whose values can be set in such a way as to explain certain observations. Thus, given a program and a set of observations, an *abduction* process is used to suggest which of our abducible predicates explain the observations. This is particularly useful in “intelligent” agent computation, where agents often have only partial knowledge of their environment and so must work out what is the most reasonable explanation for the things it perceives. Importantly, for our purposes, an *abductive* proof procedure is used as part of this process.

KGP and SCIFF: The \mathcal{KGP} agent approach is based on logic programming but extended with specific agent aspects: **K**nowledge, **G**oals, and **P**lans [108]. Abductive logic programming is used via the *SCIFF* procedure for interaction verification [3]. *SCIFF* was originally developed to verify the compliance of agents to interaction protocols; it uses (1) abducibles to represent hypotheses about agent

behavior, (2) CLP constraints, and (3) existentially quantified variables in integrity constraints.

Action logics: In a series of papers [11, 54, 55], the authors tackle the problem of specifying and verifying systems of communicating agents and interaction protocols (e.g., verification of a priori conformance to the agreed upon protocol). This applies to the case where protocols are specified with finite-state automata or when the policies can be implemented in DYLOG, a computational logic. The last approach [55] is based on a dynamic linear-time temporal logic.

5.4 Example

Recall the example of two robots working together to manufacture an artifact, introduced in Chapters 13 and 15. We considered some of the requirements of such a scenario in Section 2.5. Now, if we wish to apply *deductive verification* to assess some of these requirements, we need a logical description of the system in question. Typically, this would contain logical representations of all the steps of the robots, for example

$$\left[\begin{array}{l} K_{robot_1} in_front_of(robot_1, A) \wedge \\ K_{robot_1} in_front_of(robot_1, B) \wedge \\ do(robot_1, load(A, B)) \end{array} \right] \Rightarrow \bigcirc in_front_of(robot_1, AB)$$

Once we have a suitable specification of the system (say *Sys*), possibly comprising formulae such as the above, then we can verify this with respect to some of the formal requirements (say *Req*) from Section 2.5 in the way described earlier, i.e.,

$$\vdash Sys \Rightarrow Req$$

Of course, we require suitable, preferably automated proof systems for the relevant logics. For example, the above will need at least a proof in *temporal logics of knowledge* [40].

6 Algorithmic Verification of Agent Models

In this section we consider the *algorithmic model-checking* problem for the logics introduced in Section 2.2: given a model \mathcal{M} , a state q in it, and a property Φ wrt. a logic \mathcal{L} , we determine whether the model satisfies the property:

$$\mathcal{M}, q \models_{\mathcal{L}} \Phi.$$

This variant of model checking is called *local*, because we evaluate the formula in a given state q . Global model checking, on the other hand, is the problem of computing all states q , such that the above relation holds (where \mathcal{M} , a property Φ in a logic \mathcal{L} are given).

In Section 6.1, we deal with the problem of how exactly to measure the input of a model-checking problem. Subsection 6.2 extends the logics introduced in Section 2.1 along two dimensions: taking into account the past history of the agent; and taking into account that the perceptions of an agent are not perfect. In Subsection 6.3 we introduce an approach with a highly compact representation of a model. Finally, in Subsection 6.4 we present a table highlighting the complexities of the logics considered so far.

6.1 The Representation and Size of the Model

How do we measure the size of a given model? Should we simply consider the number of states? Should we assume the model is given *explicitly* and we just count the number of symbols that are necessary to represent it?

Maybe an *implicit*, for example, *symbolic*, representation is more appropriate? We might have available operations to extract information from the model without unfolding (or unraveling) it.

Let us illustrate these problems in an area that most computer scientists know quite well.

Example 14.3 (Explicit versus implicit representation) *We here consider the famed primality problem: checking whether a given natural number n is prime. A very simple and well-known algorithm uses \sqrt{n} -many divisions (starting with 2, then 3, etc., until \sqrt{n}) and thus runs in less than linear-time when the input is represented in unary.*

But a symbolic representation of n needs only $\log(n)$ bits and thus the above algorithm runs in exponential time: \sqrt{n} is exponential as a function of $\log(n)$.

This does not necessarily imply that the problem itself is of exponential complexity. In fact, the famous and deep result of Agrawal, Kayal, and Saxena shows that the primality problem *can* be solved in polynomial time: there is an ingenious algorithm that runs in time polynomial in $\log(n)$.

We distinguish between the following approaches for measuring the complexity of the model-checking problem in multiagent systems:

Explicit: The input size is given by the number of transitions in the model and the length of the formula. Thus, we assume the model is given *explicitly*. However, many realistic multiagent systems are characterized by an immensely huge state space that cannot be handled efficiently.

Implicit: We assume that the *transition function is implicitly encoded* in a sufficiently small way. The input size can then be viewed as a function of the number of states and the number of agents (and the length of the formula).

Highly compact: For many systems, some symbolic and thus very compact representations are possible. The model can be defined in terms of a *compact high-level representation*, plus an *unfolding procedure* that defines the precise relationship between representations and explicit models of the logic. Of course, unfolding a higher-level description to an explicit model involves usually an exponential blowup in its size.

We are now ready to tackle the questions at the beginning of this subsection. Taking only the number of states into account would give a misleading measure. Let n be the number of states in a concurrent game structure \mathcal{M} , let k denote the number of agents, and d the maximal number of available decisions (moves) per agent per state. Then, $m = \mathbf{O}(nd^k)$.

Thus, if we consider explicit models, the size of the input is measured as nd^k . If we consider, however, implicit models, then the size of the input is viewed as a function of n and k . Therefore, many model-checking algorithms (e.g., from [7]) are *polynomial* in nd^k but they run in *exponential* time if the number of agents is a parameter of the problem (implicit models).

6.2 (Im-)Perfect Information, (Im-)Perfect Recall

When we introduced the logics \mathbf{CTL}^* and \mathbf{ATL}^* , we made two simplifying assumptions:

Perfect Information: *Agents have perfect information about the current state.*
All states can be distinguished and all agents know the current state.

However, often agents do not perceive their environment perfectly. Some are able to distinguish certain states, whereas others might do so for different states. This requires that each agent only knows an equivalence relation of the set of states. And then the strategy of an agent has to be compatible with this relation. We refer to Example 14.4.

Imperfect Recall: *Agents base their decisions only on the current state.* This means that whenever an agent gets back to this state, its decision must be the same. Thus a strategy is defined as $s_a : St \rightarrow Act$ where $s_a(q) \in d_a(q)$, a *memoryless* strategy.

A more flexible (and powerful) method would be to base the decision not only on the current state, but *on the whole history of events until now*. A

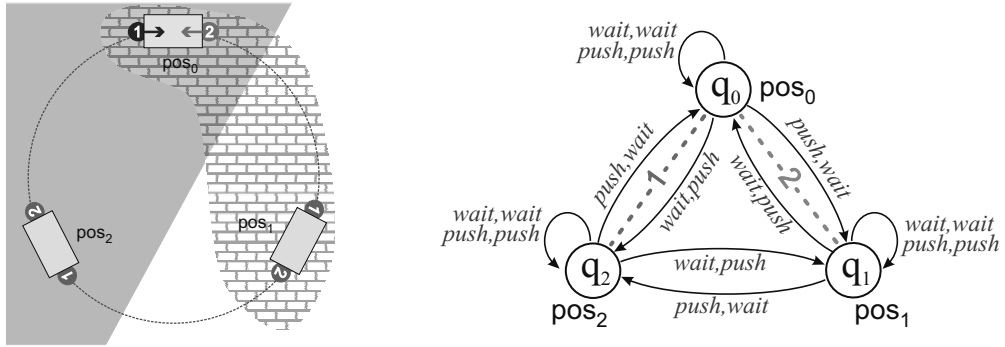


Figure 14.6: Two robots and a carriage: a schematic view (left) and an imperfect information concurrent game structure \mathcal{M}_2 that models the scenario (right).

history is a finite sequence of states of the system. Thus $s_a : St^+ \rightarrow Act$ where $s_a(q) \in d_a(q)$. This is then called *perfect recall*.

IR, Ir, iR, ir: Combining the two dimensions mentioned above gives us four different logics. **ATL_{IR}**: ATL with perfect information and perfect recall; **ATL_{Ir}**: ATL with perfect information and imperfect recall; **ATL_{iR}**: ATL with imperfect information and perfect recall; and **ATL_{ir}**: ATL with imperfect information and imperfect recall.

We note that **ATL_{Ir}** and **ATL_{IR}** are equivalent, which does not hold for the **ATL^{*}** version. Consider Figure 14.2 and the formula $\langle\langle 1, 2 \rangle\rangle(\Diamond pos_1 \wedge \Diamond halt)$. Does this formula hold in state q_0 ? In order to make it true, we have to first go to q_1 and then back to q_0 to finally switch to the halting state. But this is only possible when we have perfect recall! This shows that **ATL_{IR}^{*}** and **ATL_{Ir}^{*}** are different.

But the formula just considered does not belong to the language of ATL. How can one show that **ATL_{Ir}** and **ATL_{IR}** are equivalent? One can show by induction on the structure of a formula that the following holds. If a formula Φ is true in a model, then there is a strategy that leads to a certain (infinite) path. So there must be a prefix that contains for the first time a state twice. Thus it is of the form q_0, \dots, q, \dots, q . Then there must be a strategy (maybe a different one) that makes Φ true in this prefix. This implies that perfect recall is not needed because all this depends on the first prefix: no history is needed (as in our counterexample).

Example 14.4 (Robots and Carriage, ctd.) We refine the scenario from Examples 14.1 and 14.2 by restricting the perception of the robots. We assume that robot 1 is only able to observe the color of the surface on which it is standing, and robot 2 perceives only the texture (cf. Figure 14.6). As a consequence, the

first robot can distinguish between position 0 and position 1, but positions 0 and 2 look the same to it. Likewise, the second robot can distinguish between positions 0 and 2, but not 0 and 1. We also assume imperfect recall.

With these observational capabilities, no agent can make the carriage reach or avoid any selected states single-handedly. E.g., we have that $\mathcal{M}_2, q_0 \models_{ir} \neg\langle\langle 1 \rangle\rangle \Box \neg \text{pos}_1$. Note, in particular, that strategy s_1 from Example 14.2 cannot be used here because it is not uniform (indeed, the strategy tells robot 1 to wait in q_0 and push in q_2 but both states look the same to the robot). The robots cannot even be sure to achieve the task together: $\mathcal{M}_2, q_0 \models_{ir} \neg\langle\langle 1, 2 \rangle\rangle \Box \text{pos}_1$ (when in q_0 , robot 2 considers it possible that the current state of the system is q_1 , in which case all the hope is gone). So, do the robots know how to play to achieve anything? Yes, for example they know how to make the carriage reach a particular state eventually: $\mathcal{M}_2, q_0 \models_{ir} \langle\langle 1, 2 \rangle\rangle \Diamond \text{pos}_1$, etc. – it suffices that one of the robots pushes all the time and the other waits all the time. Still, $\mathcal{M}_2, q_0 \models_{ir} \neg\langle\langle 1, 2 \rangle\rangle \Diamond \Box \text{pos}_x$ (for $x = 0, 1, 2$): there is no memoryless strategy for the robots to bring the carriage to a particular position and keep it there forever.

6.3 Modular Interpreted Systems

Most real multiagent systems are characterized by a huge state space that is impossible to represent explicitly. Thus it would be better to consider a *compact high-level representation* of this model accompanied by an unfolding procedure that defines the relationship between representations and models of the logic. Of course, one cannot escape the overall complexity problem: unfolding a high-level description to an explicit model will remain an exponential blowup in its size.

A trivial example is a system whose state space is defined by just r binary attributes. The number of global states in the system is then $n = 2^r$. A more refined approach from [77] defines “high-level descriptions” in terms of *concurrent programs*, which can be used for simulating binary variables, but also for processes or agents acting in parallel.

A concurrent program P is composed of k concurrent processes, each described by a labeled transition system $P_i = \langle St_i, Act_i, \mathcal{R}_i, \Pi_i, \pi_i \rangle$, where St_i is the set of local states of process i , Act_i is the set of local actions, $\mathcal{R}_i \subseteq St_i \times Act_i \times St_i$ is a transition relation, and Π_i, π_i are the set of local propositions and their valuation. The behavior of program P is given by the *product* of P_1, \dots, P_k (viewed as a labeled transition system) under the assumption that processes work asynchronously, actions are interleaved, and synchronization is obtained through common action names.

While concurrent programs seem to be sufficient for reasoning about purely temporal properties of systems, they are not for reasoning about agents’ strategies

and abilities. For the latter kind of analysis, we need to allow for more sophisticated interferences between agents' actions (and enable modeling agents that play synchronously).

A modular interpreted system (MIS) is defined as a tuple $\mathcal{M} = \langle \text{Agt}, \text{env}, \text{Act}, \mathcal{I} \rangle$, where $\text{Agt} = \{a_1, \dots, a_k\}$ is a set of agents, env is the environment, Act is a set of actions, and \mathcal{I} is a set of symbols called *interaction alphabet*. Each agent has the following internal structure: $a_i = \langle St_i, d_i, out_i, in_i, o_i, \Pi_i, \pi_i \rangle$.

The unfolding of a MIS \mathcal{M} to a concurrent game structure is naturally induced by the synchronous product of the agent (and the environment) in \mathcal{M} , with interaction symbols being passed between local transition functions at every step. The unfolding can also determine indistinguishability relations as follows: $\langle q_1, \dots, q_k, q_{\text{env}} \rangle \sim_i \langle q'_1, \dots, q'_k, q'_{\text{env}} \rangle$ iff $q_i = q'_i$, thus yielding a full *iCGS*.

ATL model checking for such higher-order representations was first analyzed in [118] over a class of *simple reactive modules*, based on the synchronous product of local models. However, such reactive modules do not allow us to model interference between agents' actions.

Note. This section is taken from [18], and we refer to the work of Jamroga and Agotnes for further details [68]. It is inspired by interpreted systems [40], reactive modules [5], and are in many respects similar to ISPL specifications [104]. A recent application is [74].

6.4 MC Complexity for LTL, CTL, ATL, and MIS

Theorem 14.1 (MC LTL has the same complexity as CTL* [24, 37]) *The LTL and CTL* model-checking problems are PSPACE-complete, and can be done in time $2^{\mathcal{O}(|\Phi|)} \mathcal{O}(|\mathcal{M}|)$, where $|\mathcal{M}|$ is given by the number of transitions.*

What does Table 14.1 tell us? The variables n and m stand for the number of states and transitions, respectively, and k is the number of agents in the model; l is the length of the formula, and n_{local} is the number of local states in a modular interpreted system.

The *P*-completeness results in the first two columns look good only at first sight: they are calculated with respect to the explicit model, which is huge. Therefore the fact that the problem is polynomial in the (huge) size of the model is not surprising.

The third column gives an overview of the complexity results for modular interpreted systems, as discussed in Subsection 6.3. The *PSPACE*-completeness result corresponds to the *P*-completeness results of the first two columns. The drop in complexity from *EXPTIME*- to *P*-completeness from **ATL**_{IR,IR} to **ATL**_{ir} looks astonishing at first. Why is the complexity going down for a more difficult

Logic \ Input	m, l	n, k, l	n_{local}, k, l
LTL	P -complete [24]	P -complete [24]	$PSPACE$ -complete [77]
CTL	P -complete [24]	P -complete [24]	$PSPACE$ -complete [77]
CTL*	P -complete [24]	P -complete [24]	$PSPACE$ -complete [77]
ATL_{Ir,IR}	P -complete [7]	Δ_3^P -compl. [69, 79]	$EXPTIME$ -compl. [118]
ATL_{Ir}	Δ_2^P -compl. [69, 111]	Δ_3^P -compl. [69]	$PSPACE$ -compl. [67]
ATL_{IR}	Undecidable [†]	Undecidable [†]	Undecidable [†]
ATL*_{Ir}	$PSPACE$ -complete [111]	$PSPACE$ -complete	$EXPTIME$ -hard
ATL*_{ir}	$PSPACE$ -compl. [111]	$PSPACE$ -complete	$PSPACE$ -complete
ATL*_{IR}	$2EXPTIME$ -compl. [7]	$2EXPTIME$ -compl.	$EXPTIME$ -hard
ATL*_{ir}	Undecidable [†]	Undecidable [†]	Undecidable [†]

Table 14.1: Overview of the complexity results: most are completeness results.

problem? The reason is that the *representation* of a concurrent game structure by a MIS can be in general more compact than that of an *i*CGS. In the latter case, the MIS encodes the epistemic relations *explicitly*, while for CGS, the epistemic aspect is completely ignored.

The results for **ATL_{Ir}** and **ATL_{ir}** in the first two columns are interesting. When we compare model-checking agents with *perfect information* versus those with *imperfect information*, the first problem appears to be much easier against explicit

models measured by their number of transitions. Then, we get the same complexity class against explicit models measured using the number of states and agents. Finally, model checking imperfect information turns out to be *easier* than model checking perfect information for modular interpreted systems. Why is that so?

The *number of available strategies* (relative to the size of input parameters) is the crucial factor here. It is *exponential in the number of global states*. For uniform strategies, there are usually fewer of them but still exponentially many in general. Thus, the fact that perfect information strategies can be synthesized incrementally has a substantial impact on the complexity of the problem. However, measured in terms of local states and agents, the number of all strategies is *double exponential*, while there are “only” exponentially many uniform strategies – which settles the results in favor of imperfect information.

A comment on the two rows that state that the model-checking problems for ATL_{ir} and ATL^*_{ir} are undecidable: This has been open for some time (although it has been stated without proof) and a (complicated and not very insightful) proof has only recently been presented in [32, 57]. The main point is that the perfect recall helps to distinguish histories that are, in the case of ATL_{ir} , indistinguishable. These can be then used to encode arbitrary runs of Turing machines and thus to encode undecidable problems like the halting problem.

MCMAS. An important system here is MCMAS [82, 83, 86], which builds on work on model checking temporal logics of knowledge and the efficient, symbolic verification of interpreted systems [81, 97, 105]. MCMAS has been evaluated on real systems, in particular, to verify properties of underwater autonomous vehicles [39].

6.5 Model Checking Agent Language Models

As should be clear from the preceding sections, there has been a great deal of work on the algorithms for, and complexity of, model checking within temporal, strategic, and epistemic logics. However, while such logics neatly capture the dynamic and informational dimensions we are interested in, they say little about *motivations* such as goals or intentions. In particular, the model-checking techniques above are not appropriate either for the BDI model or for programming languages based on this model.

So, what are we to do? As we see in Chapter 13, the vast majority of agent programming languages are based on the BDI approach, or at least some variation of it. An obvious, and viable, route is to utilize the operational semantics for the programming language in question. Recall that an operational semantics describes the configurations the system/program can be in and gives rules for transforming

between these configurations. In this way, it provides an abstract view of the potential execution (i.e., sequence of configuration changes) of any program. Now, given a specific program, we can work through the program and by examining the operational semantics, can build a model of all the potential configurations that the particular program can generate. This model can then be checked against a logical requirement.

This approach has been used often, and in the following some selected examples are given.

To PROMELA and SPIN: In [133] simple agent programs were verified via a translation to SPIN. In [17], AgentSpeak programs were translated to the PROMELA language, and then the SPIN model checker is used to verify its properties. Note that subsequent work translated to JAVA and used JPF (see Section 7.4).

GOAL: In [71], the operational semantics of the GOAL agent programming language is used to describe all the possible executions of a specific GOAL program. The on-the-fly algorithmic verification techniques are used to explore all these potential executions. This provides quite an efficient verification mechanism for GOAL programs.

Rewriting: Given that the formal semantics of an agent language is often given in terms of rewrite rules (especially if it is an operational semantics), then an alternative way to tackle verification would be to base it on some underlying *rewrite* system. This clearly has some link to the use of an underlying logic programming system (as in Section 5.3) as well as a link to the *model-checking* approaches based on operational semantics that we consider here.

The predominant rewrite system is MAUDE, which provides an efficient and flexible rewriting basis [25]. Indeed, the operational semantics of several agent languages have been translated to MAUDE input [41, 124].

In [9], a programming language is defined that facilitates the implementation of coordination artifacts, which are used to regulate the behavior of individual agents. This language provides constructs inspired by social and organizational concepts and allows different operational semantics (*vis-a-vis* the scheduling mechanism of such constructs). They show that a particular semantics can be prototyped in MAUDE. As an example, they define certain properties, enforcement and regimentation, and verify them using the MAUDE LTL model checker.

Throughout this section we have assumed that the models analyzed, and often produced through the language's operational semantics, *exactly* match the agent

program executions. However, as an operational semantics is likely to be imperfect, this is far from straightforward. So, in the next section, we will consider how to verify the *actual* agent program being used, rather than a *model* of its executions.

7 Algorithmic Verification of Agent Programs

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then *algorithmically verifying* this model with respect to some requirement. Yet, as we described in Section 4.4, a very appealing approach to verification is to verify the actual program rather than a model of it. But, is this possible for agent programs? If so, how does this work? And will it work for many different agent programs? We will consider all these questions in this section.

7.1 General Problem

So, we wish to verify an agent program by exploring its executions directly, rather than building a model (typically a finite-state automaton) and checking that. Once we have an operational semantics then, in principle, we should be able to achieve such *program checking*. However, this is far from simple to implement! Consequently, the only agent program verification system (the one we will describe in this section) takes advantage of sophisticated program verification systems for non-agent programs. Specifically, it extends the JAVA PATHFINDER system for checking JAVA programs.

Recall how program verification works, based on the “on-the-fly” model checking in Figure 14.4. In the particular case of JAVA PATHFINDER, a modified JAVA virtual machine has been developed that allows both the parallel checking of properties and the backtracking of system executions. Now we outline the MCAPL framework [29, 85], which comprises the AIL semantic toolkit (Section 7.2), the MCAPL interface, and the AJPF model checker (Section 7.4).

Note: Agent program verification is a leading-edge research technique; and while the system is available for use/experimentation at [85], it is perhaps not as sophisticated as other tools you will see in this book. Since the need is obvious, an increasing user community will lead to more refined tools that can be used in the future. In addition, it is well-known that program model checking is significantly slower than standard model checking applied to models of the program execution. Thus, verifications in AJPF take minutes and hours, rather than seconds,

with tools such as SPIN or NUSMV. In spite of this, agent program verification is clearly very useful.

7.2 AIL Semantic Toolkit

Let us recall what we do when we write an operational semantics for our favorite agent programming language. We decide on the essential *configurations* in the system; for example, in a BDI-like language we might record the current beliefs, current intentions, suspended intentions, applicable plans, etc. Then we define allowable transitions between these configurations, corresponding to how the language works. A basic transition could be

$$\frac{\text{add_belief}(b)}{\langle \text{Beliefs}, \text{Intentions}, \dots \rangle \longrightarrow \langle \text{Beliefs} \cup \{b\}, \text{Intentions}, \dots \rangle}$$

where the set of beliefs is updated with the new belief, “b,” to generate a new configuration. We must generate many, usually more complex, rules in order to provide the operational semantics of our language.

Then there are two particular ways in which we might use the operational semantics. The first is to provide an implementation. Since such an operational semantics essentially describes a language interpreter, then the language can be implemented just by encoding the operational semantic rules. Then, as we have seen in previous sections, we might use the operational semantics as the basis for a model checker. However, every time we tackle a new language, we must go through this process again, defining configurations, transitions, practical implementation, and model-checking procedures. A particularly awkward aspect is defining how the model-checking procedure accesses/evaluates beliefs, intentions, etc., within the agent execution. Finally, since many agent languages are actually very similar, then there is surely scope for some re-use of the above aspects.

The *Agent Infrastructure Layer* (AIL) is essentially a toolkit that aids the development of all the above aspects for BDI-like, JAVA-based, agent programming languages [29].

The idea is as follows. When you have an idea for a new agent programming language, you can access the AIL toolkit to build an operational semantics for the language. Once such a semantics is built, the AIL toolkit naturally provides a JAVA implementation (since the semantic elements are all objects/classes within JAVA) and also provides ways in which a special model checker (called AJPF) can access the components of the semantics. Although AIL provides a wide range of “ready made” semantic components and rules corresponding to typical BDI language features, the developer still has the capability to write new semantic rules (so long as they respect the interfaces and interactions required).

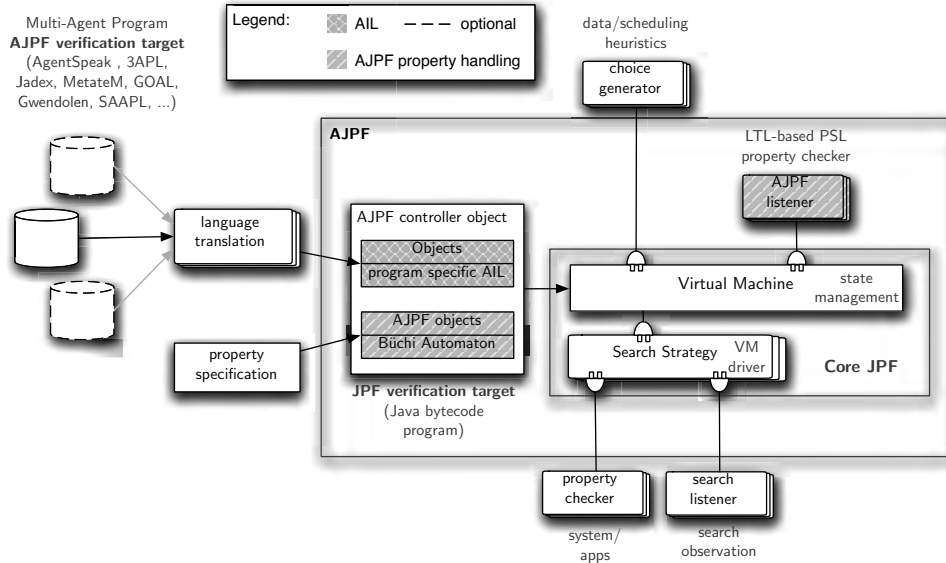


Figure 14.7: AJPF architecture [29].

Thus when we run a program in our new agent programming language, we run it in an AIL-based interpreter, which utilizes special AIL data structures to store the agent's internal configuration (typically, beliefs, intentions, plans, etc.). Importantly, AIL also provides support for describing the agent's *reasoning cycle* within the operational semantics. Such a reasoning cycle defines how the agent's practical reasoning progresses, depending on its current internal configuration. AIL provides support for constructing reasoning cycles, along with a number of rules that typically appear in the operational semantics of agent programming languages. More details of the AIL toolkit are given in [29] and a schematic diagram is given in Figure 14.7.

7.3 Multiple Semantic Definitions

By using a common semantic base, we are able to define the formal semantics for many agent programming languages. For example, in [27], the AIL is used to provide semantics for GOAL [26], SAAPL [132], and GWENDOLEN [28]. Not only can such agents be developed and verified separately, but the fact that the semantics for all three are built on a common basis means that *heterogeneous* multiagent systems can be verified. Thus, in [27], a system comprising GOAL, SAAPL, and GWENDOLEN agents communicating together is verified. Since we have not yet explained how agents built using AIL semantic definitions are verified, we will turn to this next.

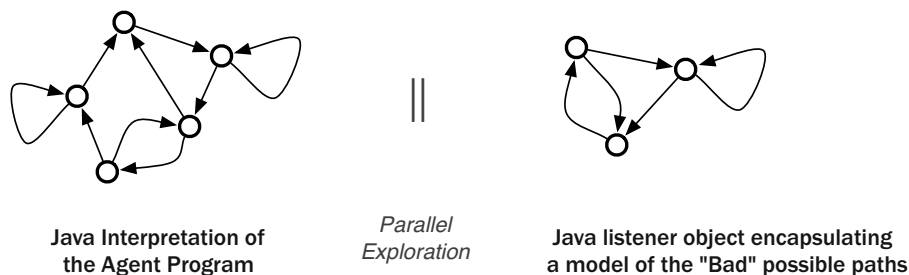


Figure 14.8: AJPF “On-the-fly” exploration.

7.4 Model Checking AIL Through MCAPL/AJPF

The AIL toolkit collects JAVA classes that can be verified through AJPF, an extended version of the JAVA PATHFINDER model checker for JAVA programs [126], mentioned in Section 4.4. When a language interpreter that has been developed using AIL is executed, then the interpreter communicates with the AJPF model checker. In particular, the interpreter will notify AJPF each time a new state is reached that is relevant to the verification, while at the same time AJPF can, through the AIL structures, access all the internal details of the agent’s execution.

Since AJPF is based on the JPF JAVA model checker, it exhaustively explores the execution of the agent, backtracking if necessary through the underlying virtual machine. In parallel, a JAVA listener object “watches” for important steps through the execution (where “important” is defined within the AIL semantic definitions) and tries to match its internal automaton to the execution it is seeing; a modified version of Figure 14.4 is provided in Figure 14.8.

Thus, not only does AIL make it easier to develop agent programming language interpreters, but it also provides easy access to sophisticated model-checking capabilities. Importantly, the program that is model checked is the program that is run.

7.5 Example

Recall again the example of two robots working together to manufacture an artifact, introduced in Chapters 13 and 15. We considered some of the requirements of such a scenario in Section 2.5. Now, if we wish to apply *algorithmic verification* techniques to assess some of these requirements, we need either a model of all possible executions in the system, or a program for the system. If we have a model, for example, generated through the operational semantics, then we can use traditional model checking as in Section 6. Alternatively, if we have a program,

such as the one described in Chapter 13, we might apply program verification techniques as described above. Note again, however, that such direct program verification is particularly slow.

Finally, in this section, we note that the MCAPL (i.e., AIL+AJPF) framework is increasingly used for verifying non-trivial agent-based systems. As well as the heterogeneous agent system from [27], the ORWELL normative agent language has been verified through AIL. On a more practical level, in [131], this approach is used to verify key parts of the control for an unmanned air vehicle.

8 Conclusions

As agents are being found to be useful in more and more application areas, the need for formal specification and verification techniques specifically devised for agents becomes more acute. Agents are not only being used in “harmless” software for INTERNET search and user interfaces, but are increasingly used in business-critical areas. Here, the viability of businesses depends on the reliability and effectiveness of the agent systems. Yet it is *safety*-critical applications for which agent reliability will have the most impact. Sophisticated autonomous, pervasive, and ubiquitous systems are being developed and deployed, with many incorporating some form of “intelligent decision making” encapsulated within an agent. Systems such as robots, space probes, intelligent homes, medical monitoring, and unmanned vehicles typically involve agents of some form or other. The critical nature of all of these, often with human life at risk, means that it is vital to have techniques for comprehensively analyzing the reliability of the underlying agent software.

As we have seen in this chapter, the use of *formal logics* is central to research involved in providing sophisticated analysis tools. The flexibility and range of logics available allows us to specify the properties we require of our agents; and the variety of techniques available allows us to verify (often automatically) the properties of our agent systems. While there is a great deal of important and interesting research that continues to be produced, it should be clear that these areas are still under active investigation. There are increasingly practical verification tools for agents, such as MCAPL [29, 85] and MCMAS [82, 86], but these essentially remain prototypes. In spite of this, however, they are beginning to be tried out in industrial situations. This is because the speed of development of the autonomous, pervasive, and autonomic systems described above is increasing, yet there are concerns about the reliability of the agents within them. For example, many aerospace companies are developing unmanned air vehicles (UAVs) for use in civilian applications, yet few have a clear idea about the reliability of the “in-

telligent” decision-making agents that are often at the heart of such vehicles. So, the need for comprehensive analysis, preferably through formal verification, is acute [131].

As much of the work described in this chapter, particularly that on agent verification, is leading-edge research, there are clearly many open research issues. We have already mentioned some of these within the text, but will recap a few of them here. While there has been significant work on the model checking of temporal logics, and indeed of temporal logics of knowledge, we have seen that this is not enough. Rational agents incorporate explicit representations of the *motivations* for their choices, and these are typically captured through goals, intentions, desires, etc. So, it is crucial to be able to model check combinations of time, knowledge, *and* goals. There has been some work in this direction, but much more is required.

Another obvious direction, especially when *real* systems are being targeted, is to address the uncertain and continuous nature of real-world interactions. If our agents must deal with environmental sensors, then such sensors will never be infallible or precise. If our agents deal with physical processes or control systems, then these are typically represented as continuous systems. And so on. So, if we wish to verify the behavior of agents in such real systems, then we are likely to need to incorporate *probabilistic* and *hybrid* verification. Again, while some work on this has been carried out, there is much left to do.

Acknowledgments

We thank Nils Bulling, Wojtek Jamroga, and Thomas Agotnes for the use of some material from joint papers, in particular Example 14.1, which went into Section 2.2 and Section 6. Nils Buling also proofread parts of this chapter and helped us to improve an earlier version. Jürgen Dix acknowledges that this work was partly funded by the NTH School for IT Ecosystems. (NTH [Niedersächsische Technische Hochschule] is a joint university consisting of Technische Universität Braunschweig, Technische Universität Clausthal, and Leibniz Universität Hannover.)

9 Exercises

1. **Level 1** State a formula in **LTL** that expresses deadlock freedom.
2. **Level 1** Show that LTL can be seen as a fragment of first-order logic. Transform a **LTL** formula into first-order logic extended by the natural numbers and including the binary predicate “ \leq ”.

3. **Level 1** The models of **LTL** are infinite paths. Show that we can also handle Kripke models that contain finite paths.
4. **Level 1** We transform **LTL** formulae into **CTL** formulae as follows: each temporal operator is preceded by A. Is the transformed formula equivalent (in **CTL**) to the original one? Prove or disprove.
5. **Level 1** Prove that $\bigcirc p \Rightarrow \Diamond p$ is valid, by using an LTL proof tool such as TSPASS: <http://www.csc.liv.ac.uk/~michel/software/tspass>
6. **Level 2** Try the online Mocha demonstration at <http://mtc.epfl.ch/cgi-bin/mocha-trial.cgi>
7. **Level 2** Specify a simple contract-net protocol in terms of either linear temporal logic or linear temporal logic combined with **S5n** modal logic (to represent agent knowledge).
8. **Level 2** Prove formally that **ATL_{lr}** and **ATL_{lr}** are equivalent.
9. **Level 2** Work out the differences between the * versions and their restricted versions for **ATL**, **CTL**, and **LTL**. Show that the * versions are really more expressive by giving example formulae and explaining how these cannot be expressed in the restricted variety.
10. **Level 3** Agent specifications typically require multidimensional logics; for example, temporal/dynamic logics combined with knowledge/belief (for information) and goals/intentions (for motivation). There are often *interactions* between these dimensions. So, describe the intuitive effect of the following interactions:
 - a) $\bigcirc Kp \rightarrow K\bigcirc p$,
 - b) $Bq \rightarrow Kq$,
 - c) $Dr \rightarrow Ir$,
 - d) $Is \rightarrow Bs$.
11. **Level 3** Check the *undecidable* entries in Table 14.1 by finding the appropriate references, and work through the proofs.
12. **Level 3** Check the last column in Table 14.1, and work through the proofs (by consulting the appropriate literature) for modular interpreted systems.

13. **Level 4** Consider the cognitive agent language CASL and the verification environment for it. Try to formalize the example in this chapter (or the one in Chapter 13) in this language, and verify it using the PVS verification system.
14. **Level 4** Choose a BDI-based agent programming language that you have an operational semantics for. Then implement (at least the core of) this semantics within the AIL. Define the semantic configurations and syntax, recast the operational semantic rules in terms of AIL primitives, and then test and evaluate the semantics.

References

- [1] M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8: 277–295, 1989.
- [2] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance Verification of Agent Interaction: A Logic-based Tool. *Applied Artificial Intelligence*, 20(2-4):133–157, February-April 2006.
- [3] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The SCIFF Abductive Proof Procedure. In *Advances in Artificial Intelligence (AI*IA)*, volume 3673 of *Lecture Notes in Artificial Intelligence*, pages 135–147, Heidelberg, Germany, 2005. Springer-Verlag.
- [4] Natasha Alechina, Mehdi Dastani, Brian Logan, and John-Jules Ch. Meyer. Reasoning about Agent Deliberation. *Autonomous Agents and Multi-Agent Systems*, 22(2):356–381, 2011.
- [5] R. Alur and T.A. Henzinger. Reactive Modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
- [6] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 100–109. IEEE Computer Society Press, 1997.
- [7] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
- [8] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.
- [9] Lacramioara Astefanoaei, Mehdi Dastani, John-Jules Meyer, and Frank S. de Boer. On the Semantics and Verification of Normative Multi-agent Systems. *Journal of Universal Computer Science*, 15(13):2629–2652, 2009.

- [10] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [11] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti. Verification of Protocol Conformance and Agent Interoperability. In *Proc. Sixth International Workshop on Computational Logic in Multi-agent Systems (CLIMA)*, volume 3900 of *Lecture Notes in Computer Science*, pages 265–283. Springer-Verlag, 2005.
- [12] T. Ball and S.K. Rajamani. The SLAM Toolkit. In *Proc. 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 260–264. Springer, 2001.
- [13] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logics*. Research Studies Press, 1996.
- [14] Marianne Baudinet. Temporal Logic Programming is Complete and Expressive. In *Proc. 16th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 267–280. ACM Press, 1989.
- [15] Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*. Elsevier, 2006.
- [16] Bookshelf - Testing Computer Software, Second Edition, CORBA 3 Fundamentals and Programming, Second Edition. *IEEE Software*, 18(3), 2001.
- [17] Rafael H. Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model Checking AgentSpeak. In *Proc. 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003)*, 2003.
- [18] Nils Bulling, Jürgen Dix, and Wojciech Jamroga. Model Checking Logics of Strategic Ability: Complexity. In Mehdi Dastani, Koen V. Hindriks, and John-Jules Ch. Meyer, editors, *Specification and Verification of Multi-agent Systems*, pages 125–158. Springer, 2010.
- [19] K.M. Chandy and Jayadev Misra. An Example of Stepwise Refinement of Distributed Programs: Quiescence Detection. *ACM Trans. Programming Languages and Systems*, 8(3):326–343, 1986.
- [20] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.
- [21] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

- [22] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.
- [23] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Proc. Logics of Programs Workshop*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1981.
- [24] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [25] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 System. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in *Lecture Notes in Computer Science*, pages 76–87. Springer-Verlag, June 2003.
- [26] F. S. de Boer, K. V. Hindriks, W. van der Hoek, and J.-J. Ch. Meyer. A Verification Framework for Agent Programming with Declarative Goals. *J. Applied Logic*, 5(2):277–302, 2007.
- [27] L. A. Dennis and M. Fisher. Programming Verifiable Heterogeneous Agent Systems. In *Proc. 6th International Workshop on Programming in Multi-agent Systems (ProMAS)*, volume 5442 of *LNCS*, pages 40–55. Springer Verlag, 2008.
- [28] Louise A. Dennis and Berndt Farwer. Gwendolen: A BDI Language for Verifiable Agents. In Benedikt Löwe, editor, *Proc. AISB'08 Workshop on Logic and the Simulation of Interaction and Reasoning*, Aberdeen, 2008. AISB.
- [29] Louise A. Dennis, Michael Fisher, Matthew Webster, and Rafael H. Bordini. Model Checking Agent Programming Languages. *Automated Software Engineering*, 19(1):5–63, 2012.
- [30] Dictionary definition of *Formal Verification*. <http://en.wikipedia.org/wiki/Verification>.
- [31] Dictionary definition of *Verification*. <http://www.hyperdic.net/dic/verification.htm>.
- [32] Catalin Dima and Ferucio Laurentiu Tiplea. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. *CoRR*, abs/1102.4225, 2011.
- [33] J. Dix, S. Kraus, and V. S. Subrahmanian. Temporal Agent Programs. *Artificial Intelligence*, 127(1):87–135, 2001.

- [34] Jürgen Dix, Sarit Kraus, and V. S. Subrahmanian. Heterogeneous Temporal Probabilistic Agents. *ACM Trans. Comput. Log.*, 7(1):151–198, 2006.
- [35] C. Dixon, M. Fisher, and A. Bolotov. Resolution in a Logic of Rational Agency. *Artificial Intelligence*, 139(1):47–89, 2002.
- [36] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
- [37] E. Allen Emerson and Chin-Laung Lei. Modalities for Model Checking: Branching Time Logic Strikes Back. *Science of Computer Programming*, 8(3):275–306, June 1987.
- [38] E.A. Emerson and J.Y. Halpern. “Sometimes” and “Not Never” Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM*, 33(1):151–178, 1986.
- [39] Jonathan Ezekiel, Alessio Lomuscio, Levente Molnar, and Sandor M. Veres. Verifying Fault Tolerance and Self-Diagnosability of an Autonomous Underwater Vehicle. In Toby Walsh, editor, *IJCAI*, pages 1659–1664. IJCAI/AAAI, 2011.
- [40] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press: Cambridge, MA, 1995.
- [41] Berndt Farwer and Louise A. Dennis. Translating into an Intermediate Agent Layer: A Prototype in Maude. In *Proc. International Workshop on Concurrency, Specification and Programming (CS&P)*, Lagow, Poland, September 2007.
- [42] Marcelo Finger and Dov M. Gabbay. Combining Temporal Logic Systems. *Notre Dame Journal of Formal Logic*, 37(2):204–232, 1996.
- [43] M. Fisher, D. Gabbay, and L. Vila, editors. *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Advances in Artificial Intelligence*. Elsevier Publishers, 2005.
- [44] M. Fisher and C. Ghidini. Executable Specifications of Resource-Bounded Agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 21(3):368–396, 2010.
- [45] M. Fisher and M. Wooldridge. On the Formal Specification and Verification of Multi-agent Systems. *International Journal of Cooperative Information Systems*, 6(1):37–65, January 1997.
- [46] Michael Fisher. Temporal Semantics for Concurrent MetateM. *Journal of Symbolic Computation*, 22(5/6):627–648, 1996.
- [47] Michael Fisher. Temporal Representation and Reasoning. In Frank van Harmelen, Bruce Porter, and Vladimir Lifschitz, editors, *Handbook of Knowledge Representation*, pages 513–550. Elsevier Press, 2007.

- [48] Michael Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011.
- [49] Michael Fisher and Anthony Hepple. Executing Logical Agent Specifications. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-agent Programming: Languages, Tools and Applications*, pages 1–27. Springer, 2009.
- [50] Nissim Francez. *Fairness*. Springer, 1986.
- [51] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proc. 7th ACM Symposium on the Principles of Programming Languages (POPL)*, pages 163–173. ACM Press, 1980.
- [52] D.M. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyashev. *Many-Dimensional Modal Logics: Theory and Applications*, volume 148 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 2003.
- [53] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *Proc. 15th Workshop on Protocol Specification Testing and Verification (PSTV)*, pages 3–18. Chapman & Hall, 1995.
- [54] Laura Giordano and Alberto Martelli. Verifying Agents’ Conformance with Multiparty Protocols. In Michael Fisher, Fariba Sadri, and Michael Thielscher, editors, *CLIMA*, volume 5405 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2008.
- [55] Laura Giordano, Alberto Martelli, and Camilla Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5(2):214–234, 2007.
- [56] James Gleick. A Bug and a Crash — Sometimes a Bug Is More Than a Nuisance, 1996. <http://www.around.com/ariane.html>.
- [57] Dimitar P. Guelev, Catalin Dima, and Constantin Enea. An Alternating-Time Temporal Logic with Knowledge, Perfect Recall and Past: Axiomatisation and Model-Checking. *Journal of Applied Non-Classical Logics*, 21(1):93–131, 2011.
- [58] D. Harel. Dynamic Logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic (II)*, volume 165 of *Synthese Library*, chapter II.10, pages 497–604. Reidel, 1984.
- [59] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.

- [60] Klaus Havelund and Grigore Rosu. Monitoring Programs Using Rewriting. In *Proc. 16th IEEE International Conference on Automated Software Engineering (ASE)*, pages 135–143. IEEE Computer Society Press, 2001.
- [61] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy A. Vilkomir, Martin R. Woodward, and Hussein Zedan. Using Formal Specifications to Support Testing. *ACM Comput. Surv.*, 41(2):1–76, 2009.
- [62] I. Hodkinson and M. Reynolds. Temporal Logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, chapter 11. Elsevier, 2006.
- [63] Gerard J. Holzmann. Logic Verification of ANSI-C Code with SPIN. In *Proc. 7th International SPIN Workshop on Model Checking of Software (SPIN)*, volume 1885 of *LNCS*, pages 131–147. Springer, 2000.
- [64] Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
- [65] Gerard J. Holzmann and Margaret H. Smith. A Practical Method for Verifying Event-Driven Software. In *Proc. International Conference on Software Engineering (ICSE)*, pages 597–607, 1999.
- [66] Gerard J. Holzmann and Margaret H. Smith. Software Model Checking. In *Proc. Formal Description Techniques (FORTE)*, pages 481–497, 1999.
- [67] W. Jamroga and T. Ågotnes. Modular Interpreted Systems: A Preliminary Report. Technical Report IfI-06-15, Clausthal University of Technology, Clausthal, Germany, 2006.
- [68] W. Jamroga and T. Ågotnes. Modular Interpreted Systems. In *Proceedings of AAMAS’07*, pages 892–899, 2007.
- [69] W. Jamroga and J. Dix. Model Checking Abilities of Agents: A Closer Look. *Theory of Computing Systems*, 42(3):366–410, 2008.
- [70] G.L.J. M. Janssen. Hardware Verification using Temporal Logic: A Practical View. In *Formal VLSI Correctness Verification, VLSI Design Methods-II*. Elsevier Science Publishers, 1990.
- [71] Sung-Shik T. Q. Jongmans, Koen V. Hindriks, and M. Birna van Riemsdijk. Model Checking Agent Programs by Using the Program Interpreter. In *Proc. 11th International Workshop on Computational Logic in Multi-agent Systems (CLIMA)*, volume 6245 of *LNCS*, pages 219–237. Springer, 2010.

- [72] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
- [73] Brian W. Kernighan and Dennis Ritchie. *The C Programming Language, Second Edition*. Prentice-Hall, 1988.
- [74] Michael Köster and Peter Lohmann. Abstraction for model checking modular interpreted systems over ATL. In Liz Sonenberg, Peter Stone, Kagan Tumer, and Pinar Yolum, editors, *AAMAS*, pages 1129–1130. IFAAMAS, 2011.
- [75] R. Kowalski. Algorithm=Logic+Control. *Communications of the ACM*, 22(7):424–436, 1979.
- [76] D. Kozen. Results on the Propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.
- [77] O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
- [78] A. Kurucz. Combining Modal Logics. In J. van Benthem, P. Blackburn, and F. Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 869–924. Elsevier, 2007.
- [79] Francois Laroussinie, Nicolas Markey, and Ghassan Oreiby. On the Expressiveness and Complexity of ATL. *LMCS*, 4:7, 2008.
- [80] Chuchang Liu, Mehmet A. Orgun, and Kang Zhang. A Parallel Execution Model for Chronolog. *Computer Systems: Science & Engineering*, 16(4):215–228, 2001.
- [81] Alessio Lomuscio, Wojciech Penczek, and Hongyang Qu. Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. *Fundamenta Informaticae*, 101(1-2):71–90, 2010.
- [82] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A Model Checker for the Verification of Multi-agent Systems. In *Proc. 21st International Conference on Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.
- [83] Alessio Lomuscio and Franco Raimondi. MCMAS: A Model Checker for Multi-agent Systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 450–454. Springer, 2006.
- [84] Zohar Manna and Richard J. Waldinger. Toward Automatic Program Synthesis. *ACM Communications*, 14(3):151–165, 1971.

- [85] Model-Checking Agent Programming Languages project. <http://cgi.csc.liv.ac.uk/MCAPL>.
- [86] MCMAS — a Model Checker for Multi-agent Systems. <http://www-lai.doc.ic.ac.uk/mcas>.
- [87] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
- [88] J.J. Meyer, W. van der Hoek, and B. van Linder. A Logical Approach to the Dynamics of Commitments. *Artificial Intelligence*, 113(1-2):1–40, 1999.
- [89] Ali Mili, Jules Desharnais, and Jean Raymond Gagné. Formal Models of Stepwise Refinements of Programs. *ACM Computer Surveys*, 18(3):231–276, 1986.
- [90] Peter G. Neumann. Cause of AT&T Network Failure. *The Risks Digest*, 9(62), 1990. <http://catless.ncl.ac.uk/Risks/9.62.html#subj2.1>.
- [91] Thomas Nicely. Original Pentium Bug Email, 1994. <http://www.trnicely.net/pentbug/bugmail1.html>.
- [92] M.A. Orgun and W. Wadge. Theory and Practice of Temporal Logic Programming. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*. Oxford University Press, 1992.
- [93] Mehmet A. Orgun and William W. Wadge. Towards a Unified Theory of Intensional Logic Programming. *Journal of Logic Programming*, 13(1–4):413–440, 1992.
- [94] Sam Owre, John Rushby, N. Shankar, and David Stringer-Calvert. PVS: An Experience Report. In Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullman, editors, *Applied Formal Methods*, volume 1641 of *Lecture Notes in Computer Science*, pages 338–345. Springer, 1998.
- [95] R. Parikh. Propositional Dynamic Logics of Programs: A Survey. *Lecture Notes in Computer Science*, 125:102–144, July 1979.
- [96] L.C. Paulson. *A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.
- [97] Wojciech Penczek and Alessio Lomuscio. Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.
- [98] Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. Synthesis of Reactive(1) Designs. In *Proc. 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 3855 of *LNCS*, pages 364–380. Springer, 2006.

- [99] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of FOCS*, pages 46–57, 1977.
- [100] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. 16th ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190. ACM Press, 1989.
- [101] A. Pnueli and R. Rosner. On the Synthesis of an Asynchronous Reactive Module. In *Proc. 16th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.
- [102] M.R. Prasad, A. Biere, and A. Gupta. A Survey of Recent Advances in SAT-based Formal Verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.
- [103] V. Pratt. Dynamic Logic. In J. W. DeBakker and J. van Leeuwen, editors, *Computer Science III, Part 2: Languages, Logic, Semantics*, volume 109 of *Mathematical Centre Tracts*, pages 53–83. (Mathematical Centre Tracts 109), Mathematisch Centrum, Amsterdam, 1979.
- [104] F. Raimondi. *Model Checking Multi-agent Systems*. PhD thesis, University College London, United Kingdom, 2006.
- [105] Franco Raimondi and Alessio Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking via Ordered Binary Decision Diagrams. *Journal of Applied Logic*, 5(2):235–251, 2007.
- [106] A.S. Rao. Decision Procedures for Propositional Linear-Time Belief-Desire-Intention Logics. *Journal of Logic and Computation*, 8(3):293–342, 1998.
- [107] A.S. Rao and M. P. Georgeff. Modeling Agents within a BDI-Architecture. In *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484. Morgan Kaufmann, 1991.
- [108] F. Sadri and F. Toni. A Formal Analysis of KGP Agents. In *Proc. European Conference on Logics in Artificial Intelligence (JELIA)*, volume 4160 of *Lecture Notes in Artificial Intelligence*, pages 413–425, Heidelberg, Germany, 2006. Springer-Verlag.
- [109] Sven Schewe and Bernd Finkbeiner. Bounded Synthesis. In *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 474–488. Springer, 2007.
- [110] K. Schild. On the Relationship Between BDI Logics and Standard Logics of Concurrency. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):259–283, 2000.

- [111] P. Y. Schobbens. Alternating-Time Logic with Imperfect Recall. *Electronic Notes in Theoretical Computer Science*, 85(2), 2004.
- [112] S. Shapiro, Y. Lespérance, and H.J. Levesque. The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In *Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 19–26, New York, NY, USA, 2002. ACM Press.
- [113] A.P. Sistla, M. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.
- [114] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT Press, 1987.
- [115] C. Stirling. Modal and Temporal Logics. In *Handbook of Logic in Computer Science*. Oxford University Press, 1992.
- [116] V.S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Özcan, and Robert Ross. *Heterogenous Active Agents*. MIT Press, 2000.
- [117] B. Thomas, Y. Shoham, A. Schwartz, and S. Kraus. Preliminary Thoughts on an Agent Description Language. *International Journal of Intelligent Systems*, 6:497–508, 1991.
- [118] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the Complexity of Practical ATL Model Checking. In *Proc. AAMAS*, pages 201–208, 2006.
- [119] H. van Ditmarsch, W. van der Hoek, and B. Kooi. Playing Cards with Hintikka — An Introduction to Dynamic Epistemic Logic. *Australasian Journal of Logic*, 3:108–134, 2005. http://www.philosophy.unimelb.edu.au/ajl/2005/2005_8.pdf.
- [120] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library Series*. Springer, 2007.
- [121] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. Public Announcements and Belief Expansion. In *Proc. 5th International Conference on Advances in Modal Logic (AiML)*, pages 335–346. King’s College Publications, 2005.
- [122] Frank van Harmelen, Bruce Porter, and Vladimir Lifschitz, editors. *Handbook of Knowledge Representation*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier Press, 2007.
- [123] B. van Linder, W. van der Hoek, and J.J. Ch. Meyer. Formalising Abilities and Opportunities of Agents. *Fundamentae Informaticae*, 34(1-2):53–101, 1998.

- [124] Birna van Riemsdijk, Frank S. de Boer, Mehdi Dastani, and John-Jules Ch. Meyer. Prototyping 3APL in the Maude Term Rewriting Language. In *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1279–1281. ACM, 2006.
- [125] Moshe Y. Vardi and Pierre Wolper. Reasoning About Infinite Computations. *Information and Computation*, 115(1):1–37, 1994.
- [126] Willem Visser, Klaus Havelund, Guillaume P. Brat, Seungjoon Park, and Flavio Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.
- [127] Website, ESA. Ariane 5 Flight 501 Failure — Report by the Inquiry Board, 1996. <http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>.
- [128] Website, Java PathFinder. <http://javapathfinder.sourceforge.net>.
- [129] Website, The SLAM Project: Debugging System Software via Static Analysis. <http://research.microsoft.com/slam>.
- [130] Website, Wikipedia. Pentium FDIV bug, 2011. http://en.wikipedia.org/wiki/Pentium\FDIV_bug.
- [131] M. Webster, M. Fisher, N. Cameron, and M. Jump. Model Checking and the Certification of Autonomous Unmanned Aircraft Systems. In *Proc. 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, 2011.
- [132] Michael Winikoff. Implementing Commitment-Based Interactions. In *Proc. 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1–8, New York, NY, USA, 2007. ACM.
- [133] M. Wooldridge, M. Fisher, M-P. Huget, and S. Parsons. Model Checking for Multi-agent Systems: The MABLE Language and its Applications. *International Journal of Artificial Intelligence Tools*, 15(2):195–225, 2006.

Chapter 15

Agent-Oriented Software Engineering

Michael Winikoff and Lin Padgham

1 Introduction

Increasingly, software is called upon to operate successfully in complex and dynamic environments, and to be adaptable, flexible, and robust. This can be achieved by software designed as a collection of agents: software entities that operate autonomously within their environment, and are able to proactively achieve goals, while responding to changes in the environment [132]. For example, in a transport logistics application [46], autonomous agents negotiate with each other to schedule deliveries, and renegotiate in the event of delays.

There have been many demonstrated applications of agents including [88, 105]: production scheduling, simulation in a range of domains, energy production and distribution, transport logistics [46], crisis management [113], flexible manufacturing [59], air traffic control [82], and business process management [16]. There is certainly anecdotal evidence that agent technology leads to much faster and more modular development of somewhat complex applications, particularly those operating in dynamic domains. Unfortunately this is difficult to verify scientifically or to quantify objectively. The only substantial study we are aware of was written by an agent technology provider, based on a study done by one of its customers. This study indicated productivity gains of up to 350% [7]. The reason for substantial efficiency gains results at least in part from the fact that the execution engine manages plan selection (based on context evaluation) and

failure recovery (based on program structure), thus relieving the programmer of explicitly coding such details. The gain in efficiency that is provided by having an infrastructure that allows abstraction from coding details can be likened to the orders of magnitude efficiency gains in moving from assembly languages to modern programming languages.

The field of agent-oriented software engineering (AOSE; and sometimes also known as agent-based software engineering) is concerned with the engineering aspects of developing agent-based systems, and how to support their development. Specifically, work in AOSE aims to provide the practitioners with *methodologies* for the design of agent systems, and with supporting tools. Because methodologies are used by humans, the nature of work in the area of AOSE is less about algorithms, theories, formal models (although they are sometimes used), or theorems – and more about (human-oriented) models, processes, and tools. Work in the field varies in its focus: some papers take a higher-level view and describe whole methodologies, whereas others focus on a particular part or aspect of the software development process, for example, extending the modeling notation to better represent organizational aspects, or providing techniques for testing agent systems.

In this chapter (and in our own work) we take a broad view of the notion of a methodology, viewing it as comprising a range of aspects that are required by a software engineer who is developing an agent system. Specifically, we view a methodology as defining an overall process, which uses design artifacts (“models”) to capture key outcomes of the process. These models are expressed using one or more notations (which may be more or less formally defined). We also view it as important that a methodology provide detailed guidelines for how to carry out key steps. For example, if a methodology says that the second step in the overall process is to identify the goals of the system, this is not much use to the designer without some indication of the sorts of techniques that could be used to identify the goals. To summarize, we view a methodology as comprising the following elements: process, models, notation, and techniques. Additionally, a methodology rests upon a foundation of concepts. In the case of object-oriented (OO) design the concepts were previously defined by OO programming languages (e.g., class, object, inheritance). In the case of agent-based software engineering, it is useful for a methodology to also include a definition of the relevant underlying concepts that it uses (see Section 2). Finally, tool support is extremely valuable, in order to help ensure a consistent design. However, that is not a focus of this chapter; for a recent survey of tools, see Pokahr and Braubach [102].

One natural question is whether designing agent systems requires a specific agent-based methodology in the first place. Or should we simply use existing methodologies? The answer is that while there are clearly some similarities be-

tween agents and objects, there are also key differences (e.g., autonomy, proactiveness), and these differences are sufficiently significant to justify the development and use of agent-specific design methodologies. For example, when designing an agent system, which, by definition, exhibits proactiveness, and which tends to be conceptualized and implemented using goals, it is important to identify and model the goals in the system. This activity, and resulting model, are not covered by existing OO design methodologies.

However, as mentioned, there are similarities between agents and objects, and as we will see, AOSE methodologies in general do adopt and adapt various elements (techniques, notations, models, processes) from OO design where it is applicable.

Furthermore, the phases of software development do not change because we are using agents: we still need to identify the purpose and scope of the system-to-be (“requirements”), plan the system’s overall structure (“design”), flesh out the details of parts of the system (“detailed design”), implement the system, and test and debug it. Furthermore, as with non-agent development, these phases are typically performed in an iterative fashion, not in a strict waterfall-like sequence. Thus, the high-level process followed by an agent-oriented methodology is similar to any methodology in that it includes activities that are concerned with defining the purpose of the system, designing the system (with varying degrees of detail), and implementing, testing, and refining the system. We also note that in many cases a system being developed will not be purely agent based, but may include parts that are best conceptualized, designed, and implemented in terms of objects, or in terms of procedural code. However, in this chapter we focus on those aspects of a system which are agent based.

The aims of this chapter are, first, to give a feeling for what an AOSE methodology looks like, without going into full details (which would require a whole book!), and, second, to give a sense of the current state of work in the field: what has been done and what the outstanding challenges are. Most of the chapter (Sections 4–9) describes the activities that one might find in a typical AOSE methodology – that is, requirements, design, detailed design, implementation, assurance (e.g., testing), as well as software maintenance. Each section begins by discussing the common “core,” i.e., activities and models that are common to a number of methodologies. Each section then goes on to discuss some of the interesting variations, i.e., particular activities or models that are unique to a small number of methodologies. In this way we hope to give a sense of where there is general agreement in the field on the use of particular models and activities, and where there are differences between the various proposed methodologies. Sections 4–9 are preceded by a discussion of the foundational concepts (Section 2) and by an introduction to the running example (Section 3). The presentation is followed by a

Year	Methodologies
1995	DESIRE
1996	AAII, MAS-CommonKADS
1999	MaSE
2000	Gaia (v1), Tropos
2001	MESSAGE, Prometheus
2002	PASSI, INGENIAS
2003	Gaia (v2)
2005	ADEM
2007	O-MaSE

Figure 15.1: A brief history of AOSE.

discussion of work on comparing methodologies (Section 10), and we close with a look at the state of the field, future directions, and challenges (Section 11).

Finally, an apology: we have tried to strike a balance between covering the various approaches that are currently well-established, without unduly cluttering the chapter by trying to mention all of the more than 50 AOSE methodologies that can be found in the literature. To the extent that we may not have succeeded completely in making the right judgments, we extend our apology to any authors whose work may have been unjustly omitted.

1.1 History of AOSE

Figure 15.1 gives a rough timeline in terms of the development of prominent AOSE methodologies. We intentionally do not present a detailed figure with a complete list of methodology and indications of influences. Constructing such a figure is difficult, and every single such figure that we have seen in other papers contains significant errors. More importantly, a cluttered figure with dozens of methodologies is not particularly useful in gaining insight.

The list of methodologies in Figure 15.1 includes those methodologies that we feel can be argued to be significant. We consider a methodology to be significant if it either influenced subsequent methodologies in a significant way, or if it was significant in its own right (e.g., widely adopted, mature, with tool support). Note that we excluded methodologies that were only described in a single paper, which includes some of the early pioneering work. A description of some of the early work in the field can be found in the 1999 survey by Iglesias et al. [69], which covers methodologies such as MASB, CoMoMAS, and Cassiopeia. One notable omission from the list is AUML (Agent UML) [67]. AUML has been quite in-

fluent, and hence is clearly significant, but, like UML, it is a notation, not a methodology.

Roughly speaking, we can see the history of AOSE methodologies in terms of three generations. The first generation of methodologies emerged in the mid to late 1990s. They can be characterized as being generally briefly described (e.g., a single brief paper), lacking tool support, and sometimes not covering all of the core activities of analysis, design, and detailed design. This first generation includes¹ DESIRE [14], AAI [78], MAS-CommonKADS [70], and Gaia [131].

The second generation of methodologies emerged in the late 1990s and early 2000s. They can be characterized as having detailed descriptions (multiple or longer papers, and in the case of Prometheus a text book), having tool support, and covering all the core activities from analysis through to implementation. The second generation of methodologies includes MaSE [37, 40], Tropos [15, 89], MESSAGE [31, 54], and Prometheus [94, 95, 130]. Tropos is interesting in that for a long time it didn't have tool support. The extended version of Gaia [137] can also be viewed as a second-generation methodology, although for a long time it too lacked tool support.

A third generation of methodologies emerged in the mid to late 2000s. Compared with the second generation, these third-generation methodologies (PASSI, INGENIAS, and ADEM) can be characterized as having an increased focus on compatibility with UML as a notation and/or a focus on model-driven development. They also tend to be more complex than second-generation methodologies. Note that although initial papers on PASSI [17] and INGENIAS [58] appeared in 2002, the INGENIAS methodology didn't really crystalize until the mid 2000s [100], and, similarly, the definitive PASSI paper appeared in 2005 [29]. It is also notable that some of the INGENIAS developers were involved in developing MESSAGE. The ADEM methodology was first described in 2005 [24], and the associated notation (AML) was first described in 2004 [25]. However, the definitive description of AML is a 2007 book [23]. ADEM and AML are influenced by many earlier methodologies, including most of the methodologies in Figure 15.1, and also by UML, OCL, and RUP.

There are two key observations that can be made. First, there is limited recent work on developing new AOSE methodologies: the prominent and significant second- and third-generation methodologies are well-developed and supported, and it is hard to justify developing yet another methodology.

The second key observation concerns diversity and convergence. In the early years of work on AOSE methodologies, there was a lot of diversity and dozens of methodologies. Over the years, most of these methodologies have faded away,

¹Note that citations given are sometimes the definitive description, rather than the earliest paper available.

Property	Supporting Concepts
Situated	Action, Percept
Proactive & Autonomous	Goal
Reactive	Event
Social	Message, Protocol ...

Figure 15.2: Relationship between properties and supporting concepts.

and a smaller number of methodologies, which have seen significant work – and typically the development of tool support – have remained active and prominent. More recently, there is increasing awareness of the drawback of diversity, and the need to standardize (e.g., [60]) or, at least, to try and reduce unnecessary differences between methodologies (e.g., [96]). Looking forward, we might expect the future of AOSE research to focus on consolidation and standardization, rather than on the development of more methodologies. See Section 11 for further discussion of future directions.

2 Agent Concepts

As discussed earlier in this book (see Chapter 1), agents are defined as having certain properties, such as being proactive, and being situated in an environment. In order to design systems of agents that have these properties, we need to use certain design concepts. For example, one way of designing agents that display proactive behavior is to model, design, and implement them in terms of the concept of *goals*. We now consider in turn each of the defining properties of agents, and which concepts can be used to support the design and implementation of agents that possess a given property. The properties and concepts used to support them are summarized in Figure 15.2.

The first, and most basic, property of agents and agent systems is that they are *situated* in an environment. In order to design agents and agent systems that inhabit environments, we need to model the environment in some way. At a minimum we need to capture the interface between the agent system and its environment. This can be done in terms of the ways in which agents affect their environment (“actions”), and the ways in which the agent system is affected by its environment, typically by receiving information from the environment (“percepts”). For example, a manufacturing robot may have certain actions that it can perform (“load a part,” “join two parts”), and may be able to perceive certain information from outside the system (“manufacturing request,” “table malfunction”).

In describing actions and percepts one also considers properties of the environment such as [112]: do actions have predictable outcomes?, is the environment fully visible?, and can actions fail? For example, a robot's actions may be subject to failure, but may nonetheless have predictable outcomes: an action either succeeds or reports an error, in which case it has no effect. Finally, although modeling the environment in terms of its interface with the agent system is sufficient for many systems, in some situations a richer model of the environment can be valuable (e.g., [109], and see Chapter 13).

A key property of agents that distinguishes them from objects is that they are expected to behave in a way that balances the pursuit of goals ("proactive") with responding to significant changes in their situation ("reactive"). In order to design and implement agents that exhibit proactive behavior we use the concept of *goals*. A goal is a certain condition that the agent persistently strives to bring about [63]. Although the literature discusses a range of goal types (e.g., [13, 36, 47, 62, 123]), in practice it is often sufficient to consider so-called "achievement" goals. These are goals that are described in terms of a condition that is required to hold at a single point in time (for example, having completed manufacturing a part). Since goals are persistently pursued by the agent, they result in autonomous behavior: the agent continues to actively pursue its goals without requiring external guidance or stimulus.

In order to design and implement agents that are able to be reactive (i.e., respond in a timely manner to changes in the situation), we design them using the concept of *events*. An event is some change of status that is significant to the agent. For example, a machine breaking down, or the arrival of a new order request. Events can arise from the receipt of messages from other agents, or from internal changes. They can also arise from percepts, if the information from the environment is significant.

Finally, agent systems are comprised of a number of agents that interact (i.e., are social). There are *many* concepts and approaches that could be used to design interacting agents, including norms, social commitments, institutions, and agent society models (see Section 11 for further discussion, and also see Chapters 2 and 13). However, the minimal concept that is almost invariably used to support the design of social agents is *messages*. When designing message-based interaction, it is often useful to consider a collection of related messages together, which is usually done by grouping related messages in an *interaction protocol* ("protocol" for short).² It is worth noting that it is also possible to design agents that interact without messages, by making changes to the environment which other agents ob-

²These protocols are specifications of the message types exchanged in a particular interaction, or conversation, and their ordering, not the details of the low-level exchange mechanisms and their transfer protocols.

serve [97]. This approach, which is called “stigmergy,” is often used with systems that comprise a large number of very simple agents, and in which goals are not represented in the agents, but are hardwired into their behavior. In the remainder of this chapter we will focus on the so-called cognitive approach, in which agents are more coarse-grained, use some form of (limited) deliberation to select their actions, and communicate directly using messages.

3 Running Example

This chapter uses a running example to illustrate the design of a multiagent system. The example used is that of a *holonic manufacturing system*. We now briefly introduce this example. Since this chapter is about agent-oriented software engineering and not holonic manufacturing *per se*, our coverage of holonic manufacturing is very brief, and we refer the interested reader to the literature for further information (e.g., [74]).

Traditional approaches to manufacturing tend to use a fixed layout and process, which works well when manufacturing jobs are consistent and identical, but are not well suited to flexible manufacturing where jobs may vary in details (e.g., packing different items into a box), and the sizes of orders may be relatively small. Holonic manufacturing has been proposed as a means of realizing flexible manufacturing by conceptualizing a manufacturing process in terms of a collection of autonomous entities that interact to realize system goals.

A *holon* (from the Greek word “holos,” whole, and the suffix “-on,” part-of) is an independent entity that does not exist in isolation, but is part of something larger. Specifically, holons are viewed as being part of a *holarchy*, which is a hierarchy of holons. Although proposed independently, it is clear (and recognized) that there is much similarity between holons and autonomous agents [19]: both agents and holons are autonomous entities that interact with each other to realize design goals. One difference is that holons are viewed as existing in a hierarchical structure, whereas agents may or may not exist in a hierarchical structure.

The simple manufacturing scenario that we use is based on the assembly cell described by Jarvis et al. [74]. There are three parts – labeled A, B, and C – and the assembly unit needs to assemble these into combined parts, which may be “ABC” parts or “AB” parts, using the following process (see Figure 15.3):

1. robot1 loads an A part into one of the jigs on the rotating table
2. robot1 loads a B part on top of it
3. the table rotates so the A and B parts are at robot2
4. robot2 joins the parts together, yielding an “AB” part

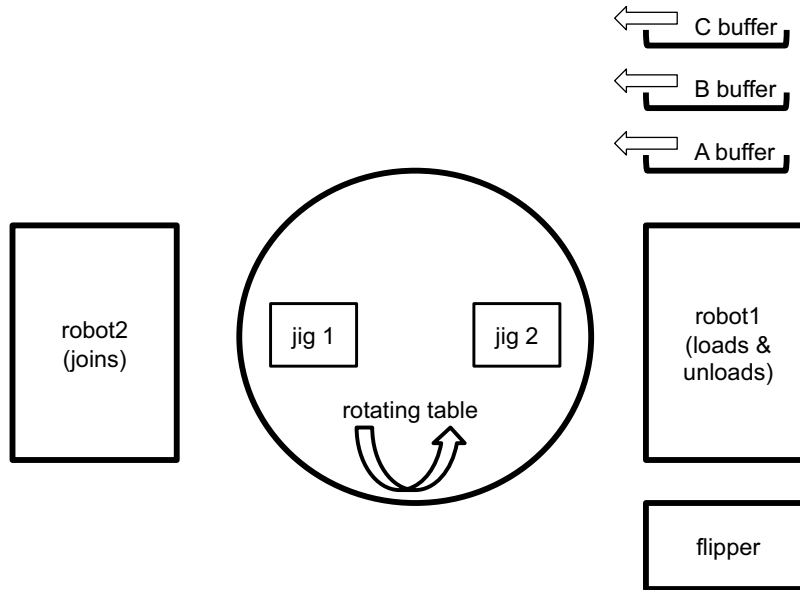


Figure 15.3: A simple example of a manufacturing assembly cell (redrawn from [74, Figure 1]).

5. the table rotates back to robot1
6. if an AB part is required, robot1 unloads the part, else continue with step 7
7. robot1 moves the AB part to the flipper
8. the flipper flips the part over (“BA”) at the same time as robot1 loads a C part into the jig
9. robot1 loads the BA part on top of the C part
10. the table rotates
11. robot2 joins the C and BA parts, yielding a complete ABC part
12. the table is rotated, and
13. robot1 then unloads the finished part.

Although this process may sound straightforward, it is made more complex by the need to manage a number of concurrent assembly jobs. In other words, we want to be able to exploit parallelism, for instance, having robot2 be assembling one part while robot1 is unloading a different order. On the other hand, we need to respect synchronization requirements such as not moving the table while robot1 or robot2 are operating. Additionally, we would like our system design to include manufacturing of BAC or BC parts, or even adding additional D parts into the pro-

cess. Furthermore, in a realistic manufacturing situation we also need to deal with failures of individual actions, and with persistent malfunctioning of equipment.

In a full system we would also need mechanisms for keeping track of the number of items in an order, possibly prioritization of orders, selection of which cells (table, robots, flipper groupings) to use for manufacturing the items in a given order, and so on. However, for simplicity we focus here only on a single cell and do not consider these broader aspects.

We will assume that robot1 receives a percept of the form *manufacture(composite)* to manufacture a composite part whose components and their order are given by the form of “composite” (ABC or AB in Jarvis’s example, but also our potential BAC, BC, BACD, etc.). We will also assume the following primitive actions:

- robot1:
 - *load(part)*, which loads a particular part type onto the jig at R1’s position (E(ast) in this example).
 - *unload()*, which unloads the part at R1’s position.
 - *moveToFlipper()*, which moves the part on the jig at R1’s position to the flipper.
 - *moveFromFlipper()*, which moves the part on the flipper, to the jig at R1’s position.
- robot2:
 - *join(jig)*, which joins the bottom part at specified jig to the top part (which may be a composite part).
- flipper:
 - *flip()*, which turns upside down the item at the flipper.
- table:
 - *rotateTo(jig,pos)* where jig is the jig number and position is E(ast) or W(est).

4 Requirements

The requirements activity is concerned with defining the required functionality of the agent system-to-be. Our description will focus on techniques, notations, and general processes that are used in a number of AOSE methodologies. We will

finish this section with a brief discussion of some techniques that are specific to certain AOSE methodologies. The sections on design and detailed design will also follow this pattern of discussing the “commonalities” first, and then highlighting interesting differences.

There are three commonly used activities in agent-oriented requirements:

- specifying instances of desired system behavior using *scenarios*;
- capturing system *goals* and their relationships; and
- defining the *interface* between the system-to-be and its environment.

Although we have described these sequentially, in practice these are done in parallel in an iterative manner. In addition, some methodologies (e.g., O-MaSE, Prometheus, Gaia) define *roles*. A role can be seen as a coherent grouping of actions, percepts, and goals that relate to a given functionality. As will be seen, agent types are typically formed by considering combinations of roles.

Using scenarios for requirements is not unique to agent-oriented methodologies. It is in essence similar to use cases common in OO methodologies [73], where the basic idea is that eliciting and describing the functionality of a system can be done in terms of specific instances of the system’s behavior for given situations, usually initiated externally. However, the details and exact structure differ. For example, when describing the holonic manufacturing system (Section 3), we described its operation in terms of an example trace – i.e., a specific instance of system behavior. While many AOSE methodologies use scenarios, they vary in the degree to which the format is prescribed. Henderson-Sellers [60, p. 14] argues that it is important to use a “textual description of each use case” to facilitate understanding. Furthermore, using structured objects can facilitate some automated processing. For example, in the Prometheus methodology, scenarios are described in a structured format as a sequence of steps, where each step is an action, percept, goal, or subscenario, performed by a given role, and accessing certain data. As an example, a scenario for the holonic manufacturing system is depicted in Figure 15.4. As can be seen, each step is described in a structured way, showing the type of each step (G for Goal, A for Action), the name of the action or goal (e.g., “build2”), and which roles are involved. We use the following five roles in our design:

manager: this role is responsible for overall management of the manufacturing process. It does not perform any actions.

pickAndPlacer: this role is responsible for moving parts in and out of the jig when it is located on the East side of the table. Associated actions are: load, moveToFlipper, moveFromFlipper, unload.

Scenario: manufacturePart(ABC)

Type	Name	Roles
G	build2	manager, pickAndPlacer, fastener
G	decideParts	manager
G	loadPart	pickAndPlacer
A	load(A)	pickAndPlacer
G	loadPart	pickAndPlacer
A	load(B)	pickAndPlacer
G	fastenParts	fastener, transporter
A	rotateTo(1,W)	transporter
A	join(1)	fastener
G	addPart	manager, pickAndPlacer, fastener
G	decideNext	manager
G	flipOver	manager
A	rotateTo(1,E)	transporter
A	moveToFlipper()	pickAndPlacer
A	flip()	flipper
G	loadPart	pickAndPlacer
A	load(C)	pickAndPlacer [in parallel with flip]
A	moveFromFlipper()	pickAndPlacer
G	fastenParts	fastener, transporter
A	rotateTo(1,W)	transporter
A	join(1)	fastener
G	complete	manager
G	assess	manager
A	rotateTo(1,E)	transporter
A	unload()	pickAndPlacer

Figure 15.4: Scenario steps.

fastener: this role is responsible for joining parts together. Associated action: join.

transporter: this role is responsible for transporting items by rotating the table. Associated action: rotateTo.

flipper: this role is responsible for flipping parts using the “flip” action.

Note that actions are always associated with the role that performs the action.

Goals need to be associated with roles. Typically, low-level goals are associated with a single role, whereas high-level goals are associated with multiple roles.

For example, the goal “fastenParts” has two roles that are jointly responsible for the achievement of the goal. However, in some cases we may choose to assign a high-level goal to a single role, which initiates, or has overall responsibility for the goal. For example, the goal “complete” is assigned to the “manager” role only, even though its achievement involves other roles as well.

The notation used in Figure 15.4 also indicates the “nesting” by using indentation of the step type. The first step (“build2”) has its type, G, unindented, representing a top-level goal. The second step (“decideParts”) is indented to indicate that decideParts is a subgoal of build2. Similarly, the load(A) action is indented to indicate that it is a part of the achievement of the loadPart goal. The structure of goals implied by the indentation is based on the goal model, which is derived in parallel with the scenario. However, as explained below, it may be the case that subgoals of different higher-level goals are interleaved, so that a subgoal may not belong to its nearest outer goal, but to something earlier. Finally, note that a scenario only captures one possible execution trace, rather than capturing all possibilities.

The second commonly used activity in requirements is to capture the goals of the system using a goal model. The goal model is complementary to the use case scenarios [110] in that it captures the different system functionalities and their relationships, and is not specific to a given execution trace. It is worth noting that the use of goals for requirements is motivated both by the fact that agents are defined in terms of goals, but also by evidence (from non-agent work) that goals are a good way to model requirements [122].

Creating a goal model can be done by identifying certain goals from the scenarios, and then refining them. One technique for refinement is asking “why” a particular goal is achieved, which identifies its parent goal, and “how” a particular goal is achieved, which identifies its subgoals [122]. Another technique is to consider how goals influence other goals. In the case of the holonic manufacturing system, the goal model is actually quite simple. One possible goal model might be defined by beginning with a top-level goal (“manufacturePart”) and asking *how* the goal is achieved, leading to the identification of subgoals. Asking *why* regarding an identified goal can lead to identification of motivating goals, which in turn leads to identification of additional subgoals. For example, asking *why* for manufacturePart could lead to identifying a goal of filling orders, which in turn leads to identification of subgoals to obtain orders and prioritize orders.

Some methodologies capture the goal model using a tree structure where each goal has as children its subgoals. Other methodologies use more sophisticated models, which can capture the influences between goals, e.g., that one goal inhibits or supports the achievement of another goal [15]; or that particular goals are triggered by certain events [53]. Figure 15.5 shows a simple goal model for

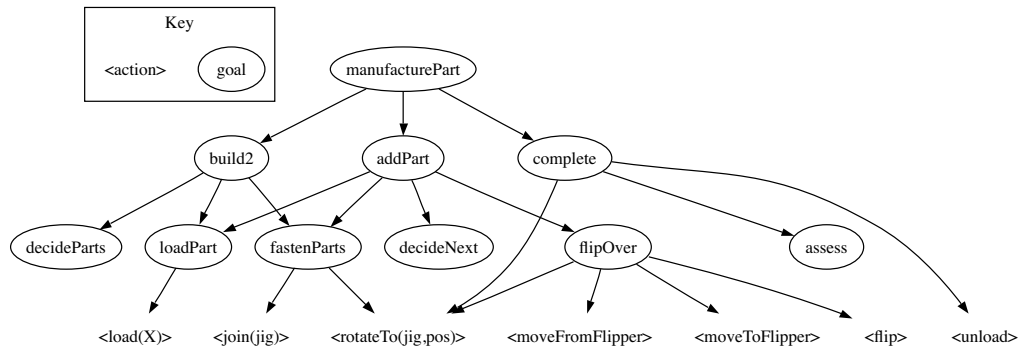


Figure 15.5: Simple goal model.

the holonic manufacturing system. This goal model does not show dependencies between goals, other than the parent-child relationship (depicted by an arrow from the parent goal to the subgoal). Note that Figure 15.5 also depicts actions, which can be helpful in understanding the design, but is actually not normally done by methodologies.

Finally, as discussed earlier, since the agent system is situated in an environment, the requirements need to include information on the environment, which can be done by specifying the interface with the environment in terms of actions and percepts. In some systems the interface is prescribed by existing hardware or systems. For instance, in the holonic manufacturing example, we have assumed that each robot and piece of equipment has specific invocable actions (e.g., `load(part)`, `rotateTo(jig,pos)`, etc.) In some systems the boundary between the agent system and its environment is more flexible, and can be fully specified by the designer. In others it may be that there is a defined low-level interface, but the designer chooses to specify the agent interface at a higher level and implement separately the low-level controls. For example, it may be that the interface to a robot is specified in terms of low-level primitives such as “open-pincer,” and “move-to-position,” but that the designer chooses a more abstract interface (e.g., `load()`) for developing the agent specification. Particularly with robotic or vision systems, it is common that a separate layer is needed to abstract from the lowest level of sensor input (e.g., pixel maps) and effector actions (which often need feedback control mechanisms at the low level, and do not benefit from an agent level abstraction).

There is overlap between these three models – scenarios, goals, and environment interface – for instance, scenarios may include goals, actions, and percepts. This overlap means that each of the three models influences the others. For example: the goals in the scenario may be a starting point for the goal model; the refined goal model may suggest additional goals for the use case scenarios; and

the actions identified in the use case scenario need to be defined as part of the environmental model.

However, it is worth noting that although there is a close correspondence between the scenario and the goal overview diagram, they are not identical. The goal overview shows structural relationships – which subgoals are part of parent goals – whereas the scenario shows a process, with relevant sequencing information. In particular, we note that the subgoals of “flipOver” are not sequenced directly after each other, but rather are interleaved with the subgoals of “loadPart” (in particular, “moveFromFlipper” is after “load(C)”).

Having discussed the common activities, we now consider some variations and differences. One significant variation is the use of an *early requirements phase*. As the name suggests, this phase, which exists in Tropos [15], based on earlier work on *i** [136], is performed before the requirements phase. The aim of the early requirements phase is to capture the organizational context in which the system-to-be will exist. This is done by identifying the stakeholders (e.g., users, owners, and other relevant organizations), their goals, and the ways in which they depend on each other and on the system-to-be. The early requirements phase allows the requirements of the system-to-be to be motivated in relation to its organizational context. One key benefit of doing early requirements is that the resulting models allow for systematic consideration of how changes in the organizational context affect the requirements of the system. Another benefit is that it is possible for alternatives to be investigated and assessed.

Capturing the domain concepts (or ontology) is important. However, perhaps because it is not agent-specific, explicitly covering this activity, including having a model for capturing the domain concepts (using a suitable notation), is something that a number of methodologies omit. Some methodologies (such as MAS-CommonKADS [71], O-MaSE [53], and PASSI [29]) do specify an activity to capture the domain concepts. PASSI and O-MaSE both use UML class diagrams as a notation to capture the domain. Another possible notation for capturing domain concepts is Protégé. Certainly it is common to do some analysis of data available and/or needed during the requirements phase. For example, Prometheus suggests designers identify data required in order to achieve goals, execute actions, or that process percepts within scenarios. This information can then be used to help determine the agent architecture.

Finally, the environment model described in this section, although generally adequate, is fairly minimal. There has been work on richer environmental models (see Chapters 2 and 13). Additionally, as noted earlier in this section, a number of methodologies also use some sort of role model.

5 Design

The design phase in AOSE methodologies aims to define the overall structure of the system. It addresses the following key questions:

- What agent types exist, and what (roles and) goals do they incorporate?
- What are the communication pathways between agents?
- How do agents interact to achieve the system's goals?

The answers to these questions are captured in two key models: a static view of the structure of the system, and a model that captures the dynamic behavior of the system. Further details of data, in particular, shared data, are also determined at this stage, although this is not specifically agent-oriented.

As in the previous section, we begin by presenting certain models and processes that are common to a number of methodologies, and thus can be regarded as “core.” We then briefly describe some interesting differences between methodologies.

The first question that needs to be answered in defining the system's structure is, which agent types should exist? A common technique for identifying agent types is to consider smaller “chunks” of functionality (e.g., capabilities in Tropos, roles in Prometheus), and to make a trade-off design decision based on the various factors that support certain chunks being grouped together. For instance, when two chunks have a related purpose or make use of common data, then there is a force pulling them together; in other words, there is a reason for grouping them together in a single agent type. Some of the factors that might be considered are:

- The degree of coupling between agents. Having a system design where each agent type needs to interact with every other agent type is undesirable because it implies a high level of coupling: any change that is made to an agent type may require consequent changes in some or even all other agent types.
- The cohesiveness of agent types. A system design is easier to understand if each agent type has a clearly defined purpose. On the other hand, if there are agent types that do a number of unrelated things, then they become complex and harder to understand and work with.
- Whether there are reasons to keep certain chunks in separate agents. There are a number of reasons why it might be a good idea *not* to have two particular chunks in the same agent type. One reason is that we may require the two chunks to exist on different hardware, for instance, in the holonic manufacturing example, the pickAndPlacer role and the transporter role interact

closely to manufacture composite parts, but correspond to different robots. Another reason is that there may be certain data that only certain chunks or roles should have access to (i.e., security and privacy).

Note that in some cases the decision as to which agent types to use is already given, or is quite clear. For instance, in a simulation where agents represent existing entities in a human organization, we would simply map each human or organizational entity into an agent type. Similarly, in the holonic manufacturing scenario it is natural to have each robot represented as a separate agent type. Indeed, this is basically the approach that we adopt. The manager role we also include in Robot1 as the manager role is making the decisions about what to load and when to unload, which are closely associated with the pickAndPlacer role. If we were including an agent that coordinates the whole system of cells, we may have decided to put the manager role in that agent. However, in our small example, we assign the manager role to the Robot1 agent. Assigning goals to agent types can be done based on the assignment of goals to roles, or may be done during the process of determining agent types. In this case, the assignment is based on the assignment of goals to roles (see Figure 15.4). Where goals that have subparts are assigned to an agent that does not itself do all the subparts, then the responsible agent will need to request other agents to do the necessary subparts. An example is the goal flipOver, which is assigned to Robot1, who will need to request assistance from Table and FlipperRobot to do the actions required for this subgoal. Similarly, the goal fastenParts, which is assigned to both the fastener and transporter role, is assigned to a single agent (Robot2, which is assigned the fastener role).

Role		Agent Type	Goals and Actions ³
pickAndPlacer	→	Robot1	loadPart, <i>load</i> , <i>unload</i> , <i>moveToFlipper</i> , <i>moveFromFlipper</i>
manager	→	Robot1	decideParts, decideNext, flipOver, assess
transporter	→	Table	<i>rotateTo</i>
fastener	→	Robot2	fastenParts, <i>join</i>
flipper	→	FlipperRobot	<i>flip</i>

We also need to define how the agents interact with the environment; that is, which actions and percepts each agent deals with. If these have been assigned to roles, then they are simply assigned to the appropriate agent along with the role. In our example the actions were predefined based on the hardware, as specified on page 704. We must also ensure that each percept is handled by some agent. In

³Actions are shown in italics.

our example we have the `manufacture(composite)` percept. We could assign this to `Robot1` (which performs the first action in the sequence needed to assemble a part), or alternatively we could introduce a new agent type, `coordinator`, which manages which parts will be manufactured in which cells, etc. However, to keep our example simple, we assign this percept to `Robot1`.

Once the agents have been determined, the next step is to define the structure of the system. Specifically, we need to define which agents communicate, what messages they use, and what the sequences are of messages (i.e., interaction protocols). However, this is not done from scratch. Instead, the previously defined scenarios help guide this process. For example, considering the `manufacturePart(ABC)` scenario (Figure 15.4), we see that the first few steps are performed by the same agent, so no message is needed. However, after `Robot1` has loaded the `B` part, in order for the table to rotate, there needs to be a message that links the completion of the `load(B)` action with the need to rotate the table. This could be a message directly from `Robot1` to the `Table` agent, or could be a sequence of messages via another agent such as `Robot2` in our example. Each time control of a process needs to be transferred to a different agent, some mechanism must be in place to facilitate this happening.

Once a protocol has been specified that allows a particular scenario to occur, this must also be generalized. One way to do this is at each point to ask the question – what else could happen here? Alternatives may involve different messages, finishing the interaction early (e.g., if an error occurs), or simply different orderings of messages. For example in the `Lock` protocol (Figure 15.6), we see alternatives where in one variation the interaction is as desired, but in the other the locking fails, in which case the protocol specifies that the robot will continue trying until it succeeds.

There are several options here as to exactly what the communications should be and how they should be specified. In most contemporary methodologies, interactions are specified by defining the patterns of messages that occur in the system, i.e., interaction protocols. These interaction protocols are often defined using the sequence diagrams within the AUMN notation [67]. For the reader who is not familiar with this notation, a brief description is provided in the Appendix.

In our example we choose to define a top-level protocol (“`ManufacturePart`”, Figure 15.7), along with a number of subprotocols. The `ManufacturePart` protocol indicates that the overall process of manufacturing a part begins with an initial loading process (locking the jig in the `East` position, then loading the first two parts) followed by fastening the two parts together (the “`Fasten`” subprotocol). This is followed by setting up with an additional part (the “`AddPart`” subprotocol) and joining this to the composite (“`Fasten`” subprotocol again), both of which are done zero or more times (“`loop`”), and then the jig is locked in the `East` position

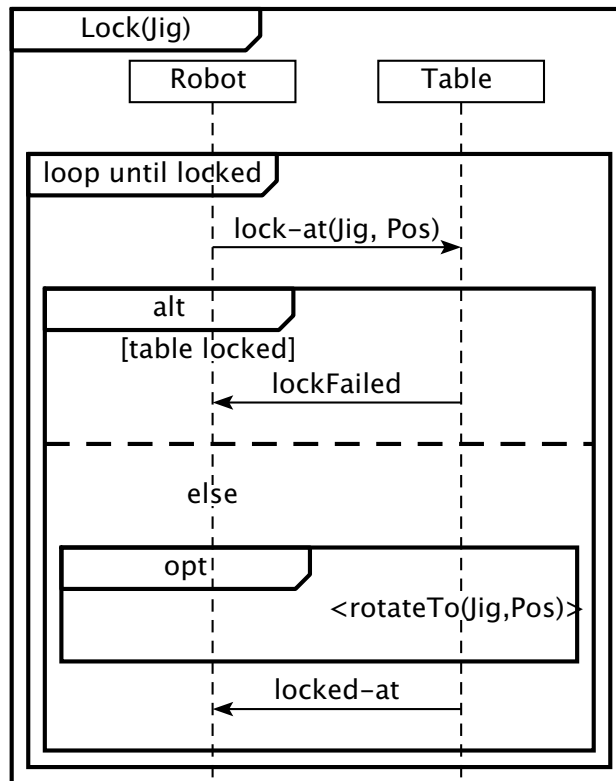


Figure 15.6: Lock protocol.

and the final part unloaded.

The Fasten protocol simply involves a request from Robot1 to Robot2 to fasten a part, the jig is positioned (using the Lock subprotocol), then the part is joined, and Robot1 notified.

In the AddPart protocol (Figure 15.8), we see that the first message exchange involves the Lock protocol, which results in the relevant jig being positioned at the East position. Following this, there is an action by Robot1 to move the composite part to the flipper. Then there are two things that (potentially) happen in parallel: the new part is loaded by Robot1, and if needed (shown by an optional box) the composite part is flipped by the flipper, with the appropriate message request from Robot1 and acknowledgment from the FlipperRobot agent. The Robot1 agent then moves the (possibly flipped) composite part back into the jig, ready for the parts to be fastened (using the Fasten protocol again), and sends the message to unlock the table. We assume that the FlipperRobot agent is only able to receive a simple flip request and respond that it is done, so Robot1 requests the positioning of the table and the lock. There are two interesting things to notice about this protocol. First, showing actions within the protocol greatly assists in understanding

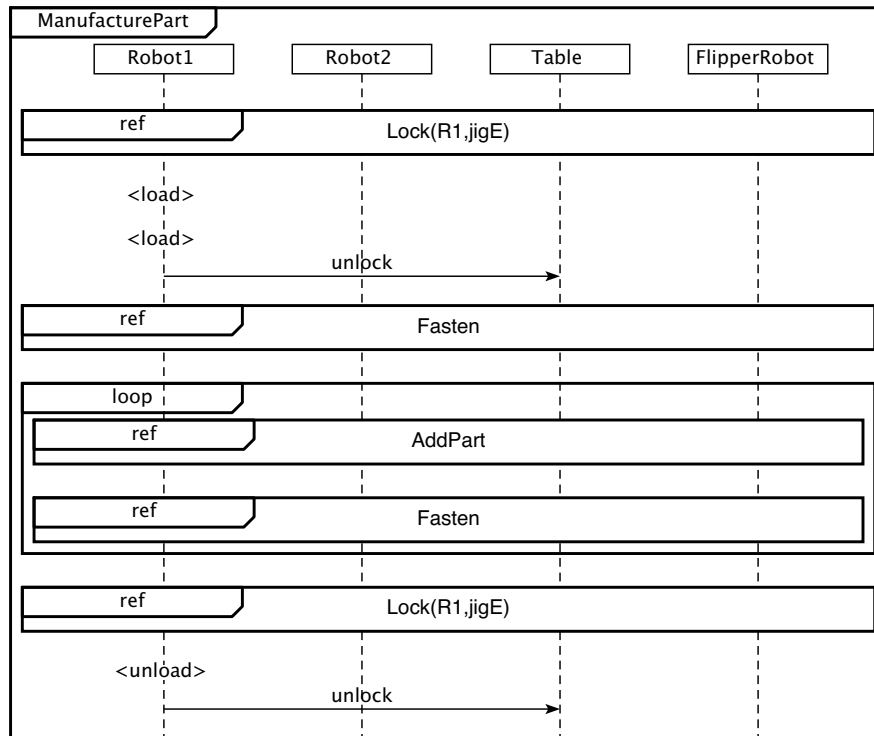


Figure 15.7: Top-level protocol to manufacture a part.

the scenario(s) the protocol supports. In particular it is necessary to show actions within the protocol if we wish to capture the requirement to load the new part in parallel with flipping the composite part if that is required. Second, capturing the optional flipping, within the parallel box, is slightly tricky to get correct. We note that designing correct protocols is where designers/developers, even those with substantial experience, often have the most difficulty.

Finally, the Lock protocol captures the message exchanges and actions to ensure that the table is locked in the required position. The locking is needed because we want to be able to interleave manufacturing of multiple parts (e.g., be loading a part while another part is being joined). If we were only manufacturing one part at a time, then we could simply rotate the table whenever we wanted to, and the Lock protocol would reduce to a simple request-action-response. However, since we do want to be able to interleave the manufacturing of multiple parts, we need to introduce locking to ensure that the table does not move while an action (e.g., loading) is being performed. In essence, the Table agent is responsible for not rotating when it is locked (i.e., between a “locked-at” and an “unlock” message), so if a lock-at message is received while the table is locked, the Table agent responds with a message indicating that the request failed (“lockFailed”) and does

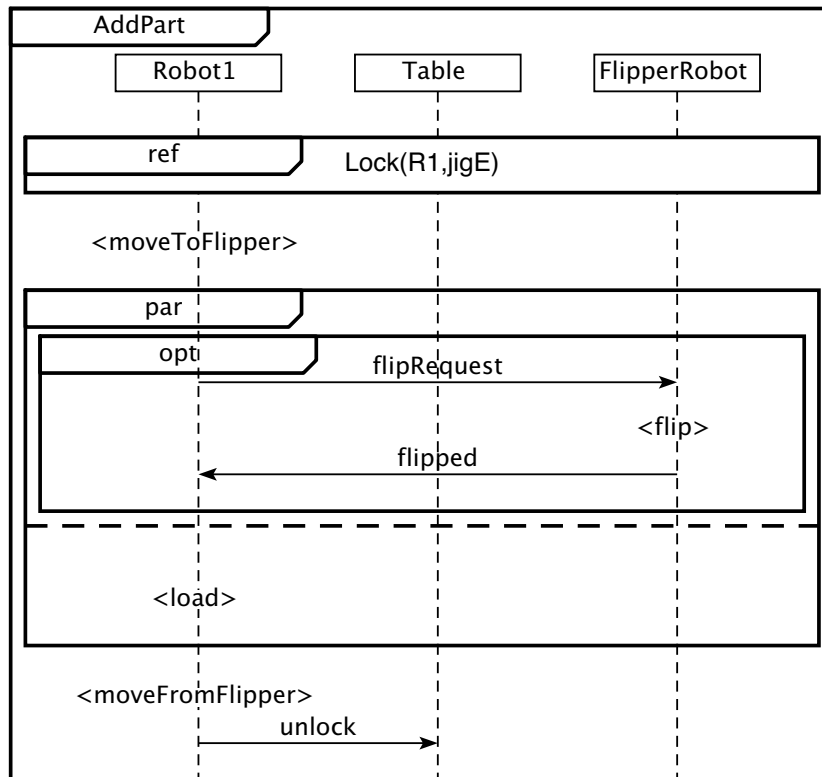


Figure 15.8: AddPart protocol.

not rotate. An alternative to this design could be to have a central coordinator that oversees and coordinates all aspects. However, it is simpler and more robust to allow each agent to manage its own part of the process. We note that the rotateTo action in the Lock protocol is actually optional: if the jig is already in the desired position, then no action needs to be taken.

One can verify that with these protocols in place it is possible to obtain the scenario described earlier. It is left as an exercise for the reader to verify that with this design it is not possible for a rotation to be performed at the same time as another action.

Finally, in addition to communication between agents and other agents, and between agents and the environment, we also define data. Often a good design will not have data that is shared between agents (which is the case in the holonic manufacturing example), so that data definition is done in the detailed design phase, when the internals of each agent type are designed in detail. Again, to the extent that the requirements phase defined the data used, this can be used as a basis for defining data in the design phase.

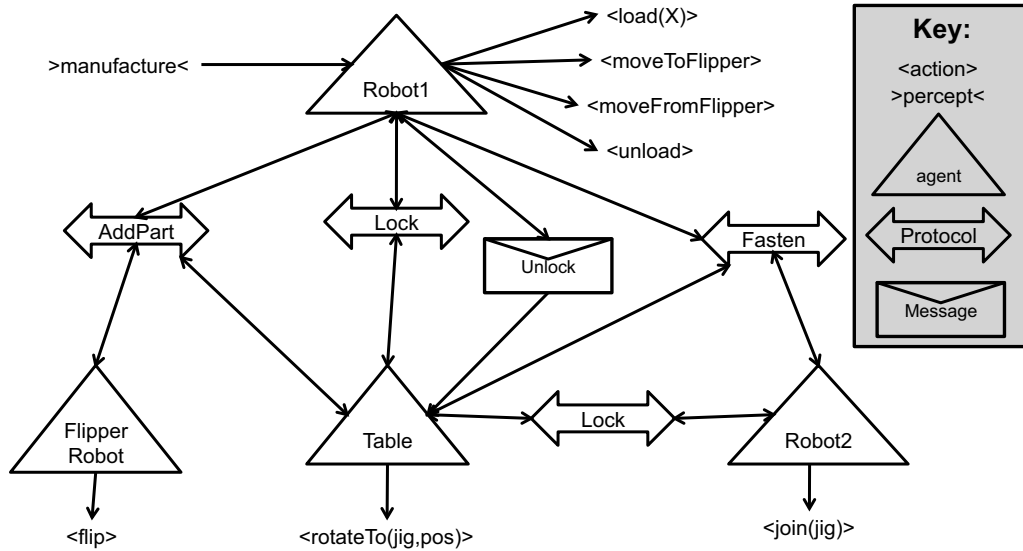


Figure 15.9: System overview diagram.

The basic structure of the system can now be captured with a model that specifies the agent types, any shared data that is used, and the communication links between agents. For example, in Prometheus a system overview diagram is used to capture the system's (static) structure. Figure 15.9 shows the system overview model for the holonic manufacturing example.⁴ It shows the agent types, which agents perform which actions, which agent handles the manufacture percept, and which agents communicate with which other agents. In many methodologies communication pathways are shown in terms of individual messages between agents, or just as links between agents that communicate. In the system overview diagram in Figure 15.9 we use a number of interaction protocol nodes (e.g., "AddPart"), which hide a number of messages.

To summarize, the two key models that result from the design phase are:

1. Some type of overview model, which shows the (static) structure of the system, including the agent types, communication paths between them, and shared data (if any). In some methodologies (e.g., Prometheus and O-MaSE), the overview model also includes the interface between the system and its environment.
2. Some sort of model of the dynamic behavior of the system. Most often,

⁴Although there have been limited attempts to standardize notation, there is no widely accepted standard. Consequently, we have used a set of simple graphical symbols that are not specific to any given methodology.

this is defined in terms of patterns of messages, using AUML sequence diagrams.

The “core” aspects of design are in fact common to most of the more prominent contemporary AOSE methodologies, where the key aspects involve defining agent types by grouping smaller chunks, specifying interaction protocols to capture system dynamics, and making use of some sort of overview diagram to capture the static structure.

However, there are a number of extensions to the core design process presented. One area of difference is the use of richer models of the relationships between agent types. The approach discussed above merely captured whether any two given agent types communicated or not, and whether they share any data. What is not captured in any depth is the relationship between the agents. Is one agent a controller or supervisor of another? Are they part of a team? Furthermore, the approach discussed above assumes that the system’s structure is static. However, if we assume that agents may take on and drop roles, or may join and leave groups or teams, then these aspects need to be considered, designed, and modeled. There has been a range of work that deals with richer organizational modeling (e.g., [42, 43, 44, 64, 65], and see also Chapters 2 and 13). Additionally, in the context of open systems (where agents designed by different people can join and leave), there has been work on *institutions* [4], and there has been some work on how to design such systems [116].

6 Detailed Design

We now need to describe the internals of agents, using as a basis the static and dynamic behavior of the system as a whole, as captured in the system overview diagram and the protocols. That is, we need to specify how agents operate to achieve their goals, and how they respond to messages in order to achieve the desired interactions. The aim of the detailed design phase is to define the internal structure of agents in sufficient detail to allow implementation to be done.

In order to define the details of agent internals in a way that supports implementation of the agents, it is necessary to know the kind of implementation platform that will be used. Will the agents be implemented in terms of finite-state machines, Petri nets, collections of event-triggered plans, or JADE behaviors? The more that is known about the implementation platform, the more detailed and specific the design can be. For example, both Tropos and Prometheus assume that agents will be implemented in terms of event-triggered plans, making them well suited to BDI agent platforms [108] such as JACK [18, 126], Jason [12], Jadex [103], and others. On the other hand, O-MaSE [38] assumes that agent internals are defined in terms of finite-state machines, and uses finite-state automata

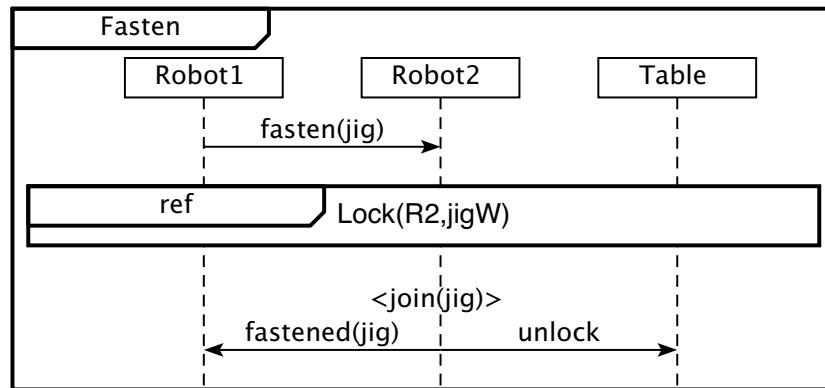


Figure 15.10: Fasten protocol.

to model the internal behavior of agents. Similarly, PASSI [29] uses activity diagrams or state charts to describe the behavior of individual agents.

In this section we will describe (briefly) the basic idea of how detailed design is done, and then illustrate it using the holonic manufacturing scenario. We will consider two cases: first, where the agents are implemented using a BDI programming language (see Chapter 13); and second, where agents' behaviors are defined in terms of a finite-state machine.

Regardless of the agent architecture that is used, the process of detailed design begins with the interface of each agent. In the design phase, each agent was defined as being able to receive and send certain messages, to perform certain actions, to deal with certain percepts, and to realize certain goals. These are the starting points for detailed design. For example, consider Robot1 in the overview diagram (Figure 15.9). We see that it participates in the AddPart protocol (Figure 15.8), the Lock protocol (Figure 15.6), and the Fasten protocol (Figure 15.10); and that it sends and receives messages to and from Robot2, the FlipperRobot, and the Table. We also know that it has actions *load*, *unload*, *moveToFlipper*, and *moveFromFlipper*, and that it receives a percept *manufacture(composite)* (or if we have a coordinator managing whole orders, this may be a message). We also know (see page 711) that it has goals *loadPart*, *decideParts*, *decideNext*, and *flipOver*. This implies (from the goal hierarchy, Figure 15.5) that it participates in:

- the goal *build2* by achieving subgoals *decideParts* and *loadPart* (the latter of which requires the action “load”)
- the goal *addPart* by achieving subgoals *decideNext*, *flipOver*, and *loadPart* (as well as doing actions *moveToFlipper* and *moveFromFlipper*)

- the goal “complete” by achieving the subgoal “assess” (as well as doing the action “unload”)

We focus here on the part Robot1 plays in the manufacturePart goal, although in general there may be multiple high-level goals an agent participates in, some of which may share subgoals. Each subgoal represents an identified chunk of activity, which will need to be triggered either by an internally posted goal, by a percept from the environment, or by a message from another agent. Both the scenarios and the protocols provide guidance about the ordering of activities, which also inform the detailed design.

Just as for earlier stages, we need some notation for capturing and communicating our detailed design. As discussed earlier, which notation is appropriate depends on the underlying agent architecture that will be used to implement the agents. If the design notation and the implementation approach are well aligned, then implementation maps each design entity to an implementation entity. For example, when implementing a UML class diagram using an object-oriented language, each UML class is mapped to a class in the programming language. Similarly, if the design of an agent’s internals is done in terms of event-triggered plans, and the implementation is done using a BDI agent programming language, then each design entity (e.g., event or plan) is mapped to a corresponding implementation entity.

6.1 Example Design: BDI Platform

We now consider our example and how the activities required to achieve the goals might be specified in terms of event-triggered plans. Each (sub)goal will need to be realized by one or more plans, each of which will need a trigger.⁵

6.1.1 Initial Structure

We know that the goal manufacturePart is triggered by the percept manufacture(composite), which is handled by the Robot1 agent. So we can put in place a plan, triggered by this percept,⁶ which can then post subgoal events to build2, addPart, and complete. Each of these events will then need to be associated with at least one plan (we will discuss later the use of multiple plans). Let us call these build2Plan, addPartPlan, and completePlan. This initial step is shown in Figure 15.11.

⁵An event that is derived from a (sub)goal (within the agent), a message (from another agent), or a percept (from the environment).

⁶We assume the parameter(s) carried by a percept, message, or subgoal are hidden within the “contents” of the entity, and not shown in the name.

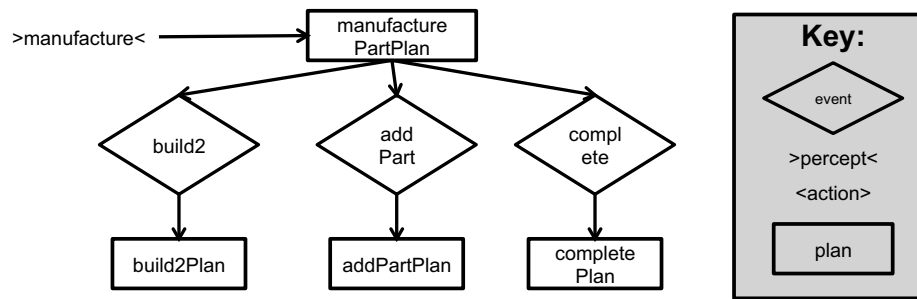


Figure 15.11: Initial step in design development for Robot1 internals.

6.1.2 Subgoal Structure for build2

Looking at our goal hierarchy we see that build2plan has subgoals decideParts, loadPart, and fastenParts. The fastenParts subgoal is also done by Robot1 so we generate a subgoal and associated plan for it. As the loadPart plan is very simple, being essentially two load actions, we can consider collapsing away this subgoal and associated plan, and simply have the actions done by the build2Plan. A good question to ask when deciding whether to collapse in this way is whether there could potentially be alternative ways to do the subgoal that is going to be eliminated. If the answer is yes, then it is better to leave the structure in place, to facilitate later adding of flexibility. However, in this case we see no alternative approach to loading the two parts, and so we do the action directly in the build2 plan, adding these two actions in our Figure 15.12. The fastenPart subgoal is done by Robot2, so we generate a message fasten(jig) to be sent to Robot2, as specified in our Fasten protocol. The details of how this subgoal is accomplished are then specified within this agent. Looking further at the top-level protocol (and the Lock protocol nested within it), we see that Robot1 also needs to lock and unlock the table, so we add these messages to Figure 15.12.

6.1.3 Subgoal Structure for addPart and complete

We now need to complete the addPartPlan and the completePlan. Looking again at the goal hierarchy we see that addPart (and therefore its plan addPartPlan) has subgoals loadPart, fastenParts, decideNext, and flipOver. As with build2Plan, loadPart becomes the action load, and fastenPart requires the message request fasten(jig). We then add subgoals and associated plans for decideNext and flipOver. Looking at the AddPart protocol we see that the plan needs to send a flipRequest message, and that the agent will receive a flipped message. Looking at the AddPart protocol we also see that Robot1 needs to lock and unlock the table as well

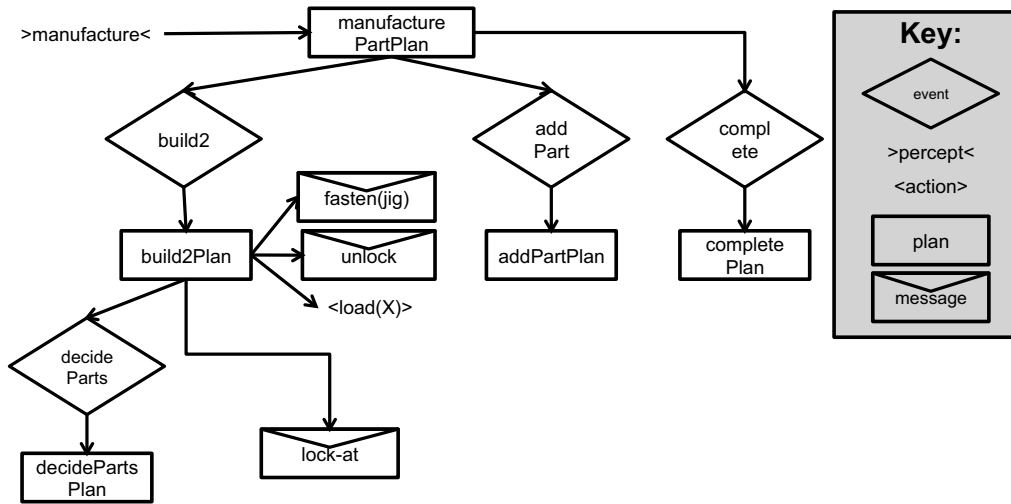


Figure 15.12: Developing the build2Plan for Robot1 internals.

as do actions `moveToFlipper` and `moveFromFlipper`, so we add all these to our Figure 15.13.⁷

We now come to the `completePlan`. Looking at our goal hierarchy again, we see that this has a subgoal assess, and actions to rotate the table (which is an action belonging to the Table agent) and unload the piece. Looking at the `ManufacturePart` protocol, we understand that the intent is to continue adding parts and fastening them (`AddPart` and `Fasten` protocols) in a loop, until finally (when the part is completed) the jig should be locked in position (the Table agent will rotate the table itself if needed), the part unloaded, and the jig unlocked again. So we have the `completePlan` do this final step by sending the message `lock-at`, doing the action `unload`, and sending the message `unlock`, as in Figure 15.13. Note that the subgoal assess is not shown. Assessment needs to be done in order to determine whether we are in a position to complete the manufacturing of the composite part (i.e., do the complete plan). We therefore decide to represent the assessment as part of the context condition of the `completePlan`: if we are in fact done, then the plan will be applicable; if not, the plan will not be applicable, and will therefore not be used.

⁷We note that if looking at the goal hierarchy alone, we would not add `moveToFlipper` and `moveFromFlipper` here, but rather in the `flipOverPlan`. However, the protocol is a more precise description, and there we see that in fact these actions need to happen outside the `flipOverPlan` in order to obtain the desired parallelism of flipping and load actions.

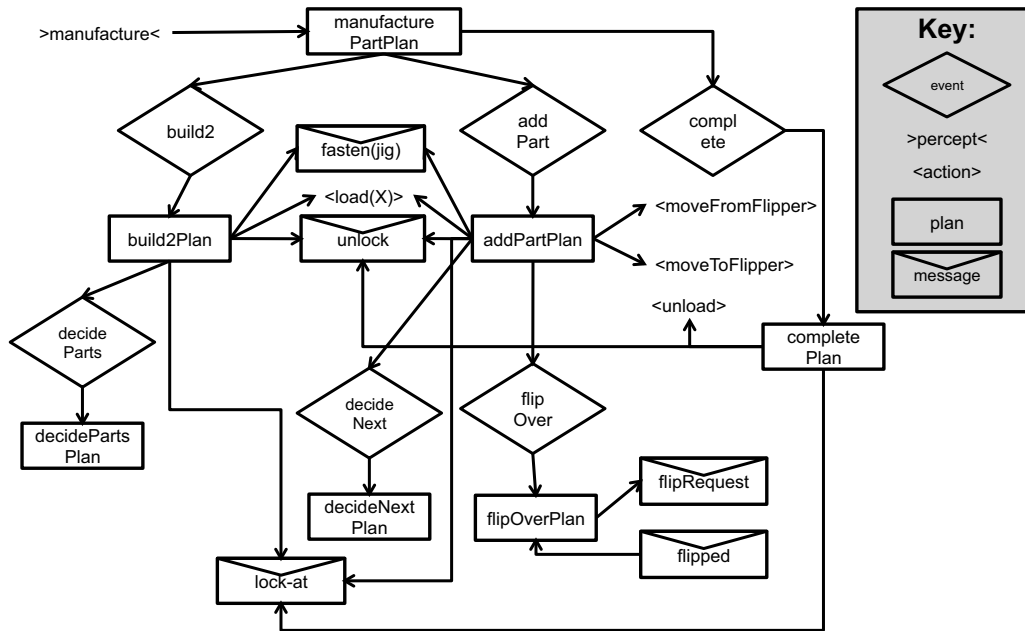


Figure 15.13: Developing the addPartPlan and completePlan for Robot1 internals.

6.1.4 Finalizing the Design

We now proceed to finalize the design. We need to check to ensure that all messages that Robot1 can expect to receive are handled properly, and that it generates all messages required of it by the protocols. The messages sent by Robot1 are lock-at, fasten, flipRequest, and unlock, whereas those received are lockFailed, locked-at, fastened, and flipped. We see that all sent messages are already present in Figure 15.13, but that the received messages lockFailed, locked-at, and fastened are not present.

The locked-at and lockFailed messages are not yet shown in Figure 15.13. We decide to use a modularization construct, often referred to as a *capability*, to abstract out the process of potentially receiving lockFailed messages and retrying lock-at until receiving a successful locked-at message. Because designs are typically much larger than our simple example, some ability to package related plans and events into abstract entities is important for understandability. These modules/packages/capabilities may often map back to the roles of the requirements phase. In Figure 15.14, we see that the lock-at message has now become an internal subgoal to be handled by this capability, which will send and receive the relevant messages (not shown in Figure 15.14). Capabilities must then have their internal structure defined in a similar way to an agent. Capabilities can be nested within each other as required.

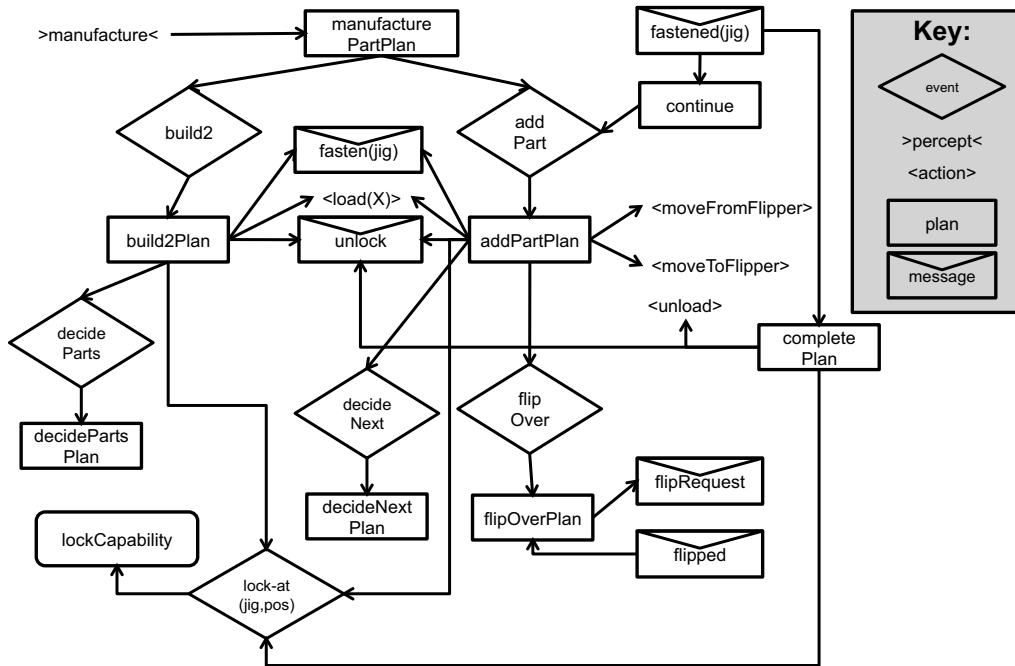


Figure 15.14: Agent overview design diagram for Robot1 internals.

We also do not yet have the fastened message shown, so we add this to Figure 15.14 as an incoming message, and consider what should happen when this is received. Looking at our ManufacturePart protocol, we see that this can be received either when we have joined our first two parts, or after we have added a part. In both cases what we want to do is ascertain whether the part is now complete (in which case we would do completePlan), or whether we need to add a further part (i.e., generate the subgoal addPart). We add a plan called “continue,” to allow us to generate the addPart subgoal in the latter case. We have already provided a context condition that will make completePlan applicable only if the part is completed. We should then add a context condition to the “continue” plan, to ensure that it is applicable only when the part still requires additional components. In this way, the context conditions of the two plans relevant for responding to the fastened message will be chosen appropriately, depending on the situation. If our current composite matches the part to be made, we will choose the completePlan. If not, we choose the continue plan and repeat the process of adding a suitable part. This illustrates a situation in which an event (in this case a message event) can have multiple plans potentially responding to it, with selection governed by the context condition. We can now see the completed design for Robot1 in Figure 15.14.

6.1.5 Multiple Plans

The ability of a given event to be handled by more than one plan is one of the strengths of the BDI architecture. This can be used to readily add additional flexibility into the system. For example, in our system, we may sometimes need to flip a composite part before adding to it, whereas in other cases we may not. Let us assume now that we are manufacturing a CABD part, and that we already have built AB. The Robot1 agent now has a choice for the next part of the process: to simply move away AB, load D beneath it, and then arrange joining (yielding ABD), or, alternatively, to move away and flip AB (giving BA), load C, and again arrange joining (yielding BAC, which can be flipped to yield CAB). This decision may be driven by the immediate availability of C and D parts to load. We can envisage two alternative plans to add a part: SimpleAdd and FlipAndAdd. SimpleAdd would have the context condition that the bottom part is needed and available, and would simply move the composite aside, load the bottom part, then move the composite back and request a fasten. FlipAndAdd would have the context condition that the top part is needed and available and would include requesting a flip. If both plans are applicable, a default could be used – perhaps having the SimpleAdd used first on the basis that the less that has to be done, the less there is to go wrong. Alternatively, further reasoning could be done to choose between the alternatives.

In addition to facilitating modular encoding of variations for different situations, the use of multiple plans to respond to an event also facilitates robustness when coupled with a failure recovery mechanism that is common in most BDI platforms. In the above example, suppose that we have selected SimpleAdd, but while obtaining a lock, some other process has taken the required C part, and when we go to do load(C) the action fails as there is no C part available. At this point, rather than having to fail the whole process, we could potentially select the FlipAndAdd plan, in order to achieve the goal of AddPart in an alternative way.⁸

One can also envision a situation in which at a higher level it is not possible to add the next part in building a particular composite (perhaps no next part is available). Rather than stalling the system and waiting for availability, we could add a plan to deal with this situation. Such a plan could temporarily remove the partially built composite and watch for the availability of the required part in order to continue. These kinds of plans to provide additional robustness are often added after the core functionality is built.

⁸Implementation would need to address the fact that if the composite part had already been moved to the flipper, then the goal to move the composite part succeeds without any action required. This is straightforward to do in a clean goal-oriented way. The goal is for the composite to be at the flipper. If that is already achieved, then the goal succeeds trivially.

6.1.6 Control Information

We note that control information such as loops or sequencing is not captured in the agent overview diagram in Figure 15.14, but is often important for full understanding. One approach is to have the designer include pseudo-code or a textual description with each plan. Another is to include graphical notations such as a loop icon and sequencing arrows. Yet another option, which is used by Tropos [15], is to specify the behavior of a single plan using a precise notation (Tropos uses UML activity diagrams for this purpose).

6.2 Example Design: Finite-State Automaton

We have considered an example of a detailed design that targets a BDI-like implementation platform. We now consider how the detailed design might instead be specified in terms of a finite-state automaton. Specifically, we use the notation used by O-MaSE [38], in which states represent activities, and transitions are annotated with guards, and received and sent messages, in the format:

[guard] receive(message,sender) / send(message,receiver).

Typically, each agent will have a single finite-state automaton, which describes its behavior. For example, the behavior of the Robot1 agent can be described by the automaton of Figure 15.15.

Figure 15.15 was derived from the interaction protocols by identifying the possible states of the interaction (corresponding to the gaps between messages in the protocol) and considering them as states of the system. Messages correspond to transitions between interaction states. For the finite-state machine (FSM) of a given agent, we compress interactions that do not involve that agent. For example, Figure 15.15 describes the behavior of Robot1, and so the interaction between Robot2 and the Table (to Lock the Table, as part of the Fasten subprotocol) is not shown.

One issue that arose in this example is that whereas the AUML notation supports subprotocols (such as “Lock”), the FSM notation used by O-MaSE does not support subprotocols. This means that subprotocols need to be expanded in the FSM. However, in this case, this would result in multiple copies of the Lock protocol. In order to avoid a needlessly complex FSM, the protocol was modified by lifting out the Lock protocol, and positioning it as a common prefix, which is then followed by either loading two parts, adding a part, or unloading the final product – where loading two parts as well as adding a part are both followed by fastening the parts together. This gives slightly different possible behaviors to the original interaction protocol. For instance, it does not require that the initial loading (“load(A)” and “load(B)”) only occur at the start of the interaction. However,

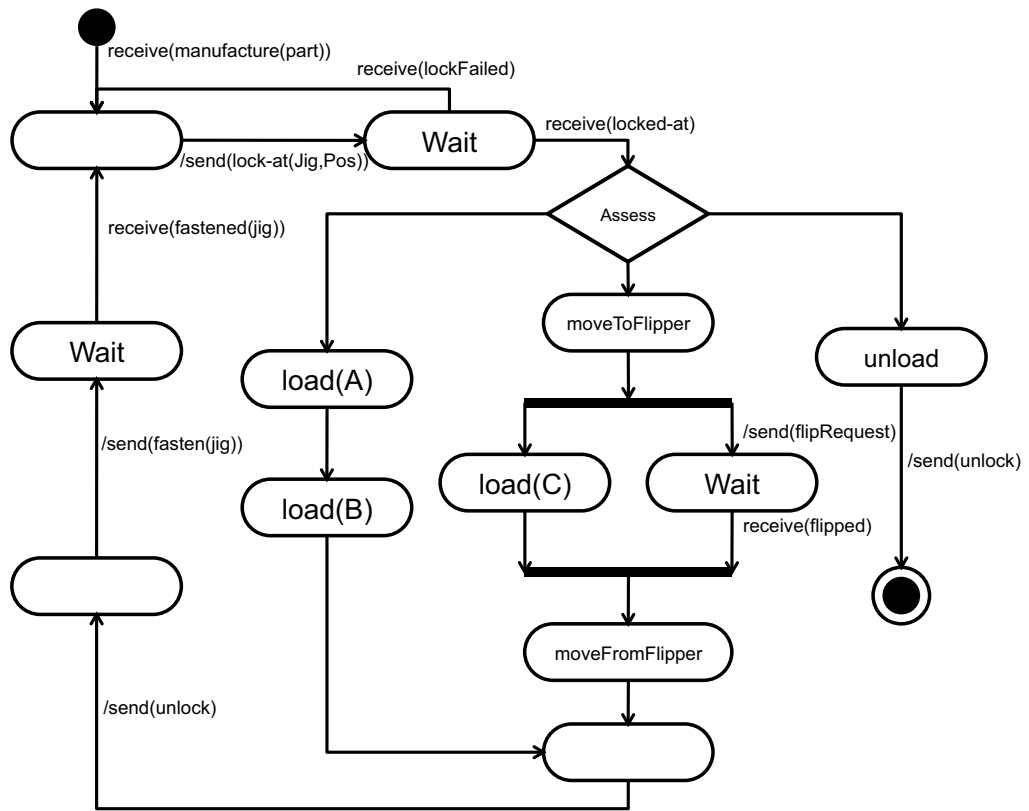


Figure 15.15: Behavior of Robot1 (finite-state automaton a la O-MaSE).

although the FSM permits the agent to load(A) and load(B) in the middle of a manufacturing process, in practice the Assess condition would be defined in such a way as to avoid this from occurring.

6.3 Final Features

Regardless of whether the behavior of an agent is specified as a single finite-state machine, or as a collection of plans, one issue that needs to be considered is whether the set of behaviors specified by an interaction protocol is equivalent to the behaviors specified by the collection of individual agents' behaviors. One aspect which must be considered is that in general agents can be pursuing multiple goals, and the protocols that govern the interactions between agents must ensure the desired behavior, even when this happens. In our example, there may be multiple (actually only two unless the system is extended in some way) ManufacturePart protocols, running in parallel and interleaved. Our design and im-

plementation must ensure proper behavior, even when planned sequences of (inter)actions are interleaved with the parallel pursuit of other tasks (or additional instances of the same task). It is left as an exercise for the reader to ensure that two instances of the *ManufacturePart* protocol can in fact be interleaved, without causing problems.

As in previous sections, having described the common core of the various methodologies, we now briefly touch on a few interesting differences. One feature that is unique to MaSE (but that does not appear to have been retained in O-MaSE) is the use of a deployment diagram to capture the run-time location of agents. Another difference that has been mentioned earlier is that methodologies differ in the notations used to capture behaviors (e.g., informal pseudocode vs. UML activity diagram).

7 Implementation

While work in AOSE tends to focus on software engineering aspects (e.g., requirements, design), it is clearly necessary for a design to be implemented, and, therefore, there is a close relationship between AOSE and the related field of agent-oriented programming languages (and other support tools for implementing agent systems).

The result of detailed design is intended to be easily mapped to an implementation. Clearly, there needs to be an alignment between the type of implementation platform used and the implementation platform type assumed by the methodology. For example, if a methodology's detailed design phase assumes a BDI-style implementation, then the results of the detailed design will be expressed in terms of agents that have event-triggered plans, and these will map naturally to a BDI-style implementation platform (see Chapter 13). On the other hand, if the detailed design phase assumes agents that are message-exchanging black boxes, and specifies the behavior of each agent using, say, a finite-state machine, then the results of the detailed design will map more naturally to an agent platform such as JADE [6].

Mapping design to implementation is generally done manually, with some assistance from tools in specifying skeleton code, which is then fleshed out. This is similar to the way that a UML design is mapped to an object-oriented language, resulting in class definitions, where the internal details of methods need to be completed. Several of the tools associated with AOSE methodologies provide support for implementation in the form of production of skeleton code. Specifically, the PDT tool (supporting Prometheus) can generate JACK code, the TAOM4E tool (supporting Tropos) can generate Jadex code, the IDK tool (supporting INGENIAS) can generate JADE code, agentTool III (supporting O-MaSE) can generate

JADE code, and the PTK tool (supporting PASSI) can generate JADE code using AgentFactory [29]. However, there is still room for improvement in tool support for the transition from design to implementation [9], and one area in particular is to support “round trip” engineering, where changes to generated code can be reflected in the design. This is partially supported by PDT, but, as far as we are aware, not by any other tools.⁹

One approach to improving the link between design and implementation is to use model-driven development [84] of agent systems (e.g., [75, 100]). The aim is to produce a complete executable system directly from the design model. This requires that the design models are specified with enough detail that this is possible. The benefits of this approach are that the implementation phase is eliminated and that it is no longer possible for the implementation and design to diverge over time. However, the cost is that the design must contain a lot of additional detail, which makes it harder to develop: in effect the implementation work is shifted into the detailed design phase. Furthermore, the model is more complex, which tends to reduce its value as a means of understanding the system. Nevertheless for some systems and user targets, this can be a good approach. The work of Jayatilleke et al. [75] showed that this approach could facilitate the work of domain experts who are not programmers in extending and refining a system through the design models.

8 Assurance

Whereas support for the “core” activities of specification, design, and implementation is now well developed in AOSE methodologies, support for other activities, such as testing, debugging, and software maintenance, is less well developed. This section briefly summarizes the state of the art in the assurance of agent systems: how can we be confident that a developed multiagent system meets its specification? This is usually accomplished through testing and debugging (Section 8.1). However, due to the characteristics of agent systems, it has been argued that testing (at least if done manually) is not likely to be adequate, and so there is also work on assurance of agent systems using formal methods, which we briefly survey in Section 8.2. The topic of support for the ongoing maintenance and modification of agent software is covered in the next section (Section 9).

Much of the work on testing and debugging takes advantage of the existence of structured design models. These models can potentially be used in a comprehensive approach to support testing of agent systems (ideally in an automated way).

⁹INGENIAS provides support for repeated generation of code from the design, but not for code changes to be propagated back to the design.

For example, it is possible to monitor the execution of an agent system and automatically detect behaviors that contradict information in the design models, such as two agents communicating when they are not supposed to do so. An additional advantage of using design models in testing is that this helps to ensure that the design models and the code remain in sync, i.e., it assists in avoiding divergence between design and code.

8.1 Testing and Debugging

Testing of agent systems is acknowledged to be inherently complex due to the nature of agent systems, which are essentially distributed, parallel systems that can be non-deterministic due to the autonomous nature of agents, and which are often realizing complex applications. For these reasons, testing is both challenging and important.

Generally speaking testing starts with unit testing, where the basic units of the system are tested in isolation to ensure that individually they perform as expected. Then there is some sort of integration testing, or module testing, where the units within a module are tested together. This is then followed by interaction testing to consider interactions between modules. Finally there is system testing and acceptance testing to investigate the behavior of the system as a whole.

In addition to the level of testing being addressed, there are also different aspects of the testing process that can be supported or automated. First, there is the design of test cases, which is often the main focus of test support tools. To define a test case, key variables must be identified, and values must be assigned to these variables. This specifies different possible configurations under which the unit or subsystem should be executed to observe behavior. Having specified a set of test cases, it is necessary to execute these, either manually or automatically. Typically this requires some sort of test driver, which initializes the system (or subsystem) being tested, sets the test case variables as specified, executes the system, and collects information about outputs. The observed outputs must then be checked against expected outputs. The expected outputs can either be specified explicitly, as part of the test case, as is done in tools such as JUnit,¹⁰ or they can be determined from a system model in the form of some kind of modeling artifacts. Each of these aspects – test case generation, test case execution, and checking of test case output – can be done either manually or automatically. Often support tools allow or require a combination of manual and automated processes.

Testing is probably the area in which there is the greatest difference between the current well-established AOSE methodologies, even though most do address it to some level, with the exception of O-MaSE. Most of the well-developed contem-

¹⁰See www.junit.org.

porary methodologies, including Tropos, Prometheus, PASSI, and INGENIAS, as well as some other less prominent ones (e.g., Seagent [120]), provide at least basic support for automated execution and monitoring of test cases, often by integration with JUnit. Most commonly this requires the developer to manually define test cases – including input, deployment setup, and expected output. The details of how this is done and what is provided differs from system to system. PASSI, for example, interfaces with JUnit with a tool that takes a system implemented in JADE and provides a GUI to assist the user in specifying test cases to test each task within an agent [20]. INGENIAS [57] also interfaces with JUnit, but provides some additional debugging features, such as visualization of which agents interact. As JUnit is quite general, it can potentially be used for tests associated with any unit type represented directly in the implementation – from the low-level plans within agents, to agents themselves.

The Tropos methodology attempts to provide a systematic approach to testing of agent systems across all the phases of unit, agent, integration, system, and acceptance testing, based on goals at various levels of abstraction [90]. However, only the agent and integration levels are developed in any detail and included in their testing tool called eCAT.

At the agent level, Tropos provides tool support for generating skeleton test cases for agent testing, based on the agent's goals. However, as with the agent development tools that integrate JUnit, the details of the test cases must be specified by the developer.

At the integration, or interaction level, eCAT provides support for automated testing of agent interactions using ontology tools to create message content for a broad range of automatically generated test cases. The interaction testing template is manually created, but the tool then obtains data by analyzing the ontology specified in the application, and obtaining values for concepts within messages. Messages are then automatically generated and used by a testing agent to send to an agent under test to ensure that it responds appropriately. By continuously generating messages using this tool, taking care to use diverse inputs where possible, agent interactions can be tested much more extensively than is possible manually. As interaction between agents is a common source of errors in agent systems, this is an important area for testing and testing support.

Prometheus does not have a single integrated testing tool, although automated unit testing has been integrated within a version of the Prometheus Design Tool (PDT), allowing design, implementation, and, then, testing to be done from the one environment. Prometheus takes the approach of automating the entire testing process, from test case generation, to execution, checking of outputs, and reporting. This is currently done comprehensively only at the level of the basic units of plans, events, and beliefs. This approach means that many thousands of test cases

can readily be executed, and has been shown to isolate complex sporadically occurring bugs in existing programs, which had defied manual debugging [138]. As the entire process is automated,¹¹ an oracle is required to determine whether output produced is correct or not. The detailed design models produced by the methodology are used for this purpose [139]. Prometheus also has preliminary work on testing system requirements or acceptance testing, using scenario specifications [119]. For a scenario to pass the acceptance test, all test cases must result in some valid sequence of inputs and outputs according to the scenario definition.

An important question in any testing regime is whether the test cases have adequately tested the system. A common testing concept here is that of “coverage.” In traditional systems it is common to consider “statement coverage” (each statement is executed at least once), “branch coverage” (each branch is executed at least once), and “path coverage” (each combination of branches is executed). In general, path coverage is not possible due to loops and recursion, though in practice these are capped. Low et al. [83] propose a representation of plans in terms of arcs and nodes, and then provide some coverage criteria that involve these arcs and nodes, as well as the plans. They explore the subsumption hierarchy of the coverage criteria, showing that this is more complex than for traditional coverage where path coverage subsumes branch coverage, which subsumes statement coverage.

Zheng and Alagar [140] also explore coverage criteria for agent system testing, although this is not implemented, and appears to be based on a finite-state machine representation, which would involve a computational explosion for many agent systems, making it impractical in practice.

The notion of coverage is further explored by Miller et al. [85], where it is applied to the testing of agent interactions. In this approach the simplest level of coverage is one in which each message in a protocol is sent and received. The most complete is plan-path coverage, which addresses the notion of paths that traverse different plans to produce the messages, and ensures that each such path is traversed. To our knowledge, none of the implemented agent testing tools explicitly use notions of coverage, although the Prometheus unit testing implicitly includes some coverage notions in that it provides warnings where sets of test cases do not show generation of an event, or execution of a plan, which would be expected at some point according to the design.

In summary, all the major current methodologies include some support for testing within their tools, with the exception of O-MaSE. However this is primarily support for automated execution (often using JUnit) with test cases; the correct results need to be manually specified. Only Prometheus and Tropos have tools for fully automated testing process, and in both cases these are limited to specific

¹¹It is also possible to manually add test cases if desired.

aspects: in the case of Prometheus, that of unit testing, and in the case of Tropos, that of receipt of messages. Both do allow execution of a very large number of test cases in the particular testing subspace. Clearly, testing of agent-based systems is an area where there is a need for substantial additional work.

8.2 Formal Methods

As noted earlier, agent systems are distributed, parallel, and potentially non-deterministic. Additionally, the space of possible behaviors for agents can be extremely large (e.g., see [129] for an analysis of the behavior space size for BDI agents). These factors suggest that testing alone may be a less effective means of obtaining assurance for agent systems than for traditional software systems. These observations have motivated work on the development of formal verification techniques for multiagent systems. For a detailed discussion of verification of multiagent systems, see Chapter 14.

Broadly speaking, much of the work uses model checking, often using existing tools, typically by translating to formalisms for which tools exist, such as Promela. Some work aims to verify actual agent programs, written in an agent-oriented programming language, whereas other work aims to verify abstract agent systems.

An example of verifying agent programs is the early work of Wooldridge et al. [133]. It verified agents written in a simple imperative language (MABLE) against specifications written in a notation that combined temporal logic, dynamic logics, and modal operators for belief, desire, and intention. This work was extended by Bordini et al. [10, 11] by using an agent-oriented programming language, AgentSpeak(F), which is a limited subset of the AgentSpeak(L) language. More recently this work has developed a means of supporting a wide range of programming languages, by defining a common notation (AIL) and translating other languages into AIL [41]. In terms of practicality, it is worth noting that the programs being verified are quite limited, for instance a six-line contract-net protocol with three agents.

An example of verifying abstract agent systems is the ongoing work of Lomuscio and colleagues (e.g., [49, 107]), which verifies *interpreted* systems, where, roughly speaking, each agent is modeled as a finite-state machine. These abstract systems are verified against properties expressed in temporal logic extended with a knowledge operator. This work is useful in verifying properties of algorithms and protocols, but does not assist in the verification of implemented agent systems.

Whereas most of the work in this area uses model checking [28], there is also some work that uses theorem proving. One example is the work of Shapiro et al. [114], which proves properties of ConGolog programs by translating to PVS, a typed higher-order logic. The program verified was a simple meeting scheduler, more complex than a six-line contract-net protocol, but still a long way from real

agent systems. Another more recent example is the work of Alechina et al. [1, 2, 3], which takes agent programs written in a simple agent language (SimpleAPL) and translates them into propositional dynamic logic. These translations, along with the agent's starting state, can be used to prove safety and liveness properties using an existing PDL theorem prover. Their key contribution is the ability to model the agent's execution strategy, and prove properties that may rely on a given execution strategy (e.g., interleaved vs. non-interleaved execution of plans). The approach appears to have only been applied to toy programs (single agent, a couple of plans, and propositional beliefs).

To summarize, current work on formal methods for verifying agent systems (see [35]) tends to be applicable to toy programs only. We would expect the efficiency of tools to improve over time. However, the logics that tend to be used for stating desired properties tend to have a high computational complexity for model checking (see Chapter 14). Theorem proving is, in general, less promising than model checking, since it is less amenable to full automation. Additionally, model checking has a significant advantage in that it provides a counterexample to a failed property [27].

Finally, all formal verification work is concerned with showing that all executions of an agent program P will satisfy a specification ϕ . One issue that needs to be considered is – where does the specification ϕ come from? In particular, it is possible for a specification to be incomplete. Indeed, it has been argued that it is quite likely that specifications are incomplete due to assumptions about the interface between software and its environment [76], or due to easily-made implicit assumptions about the execution model that allow proofs to be incorrect. For example, Jackson [72, p. 87] describes a bug that was found in a widely-used implementation of a binary search algorithm (which had been proved correct). Winikoff [128] presents a proof of correctness for a simple waste disposal robot, and then goes on to describe various errors in the simple program that exist despite it having been proved to be correct!

More broadly, it has been argued that “*Problems with requirements and usability dwarf the problems of bugs in code, suggesting that the emphasis on coding practices and tools, both in academia and industry, may be mistaken*” [72, pp. 86–87]. Based on these observations, a number of authors (e.g., [72, 111]) have argued for the use of *safety cases* to provide *direct* end-to-end evidence of correctness, that is, an end-to-end argument that provides evidence that the system exhibits desired properties [72], and which establishes which (explicit) assumptions need to be made in order for certain properties to follow from the system. Jackson also argues that properties should be expressed in real-world terms rather than in software terms. For example, specifying a safety property in terms of the radiation dose received by a patient, rather than in terms of the software comput-

ing a correct dose. These ideas have yet to be explored in the context of agent systems.

Quality assurance is unequivocally a very important issue, especially in complex systems, which is often where an agent paradigm gives the most benefit. Consequently, a broad range of techniques that contribute to this endeavor – from automated testing to formal verification, as well as approaches such as safety cases – are likely to be of value.

9 Software Maintenance

Once software has been designed, implemented, assured, and deployed, it is subject to ongoing maintenance to fix bugs (“corrective maintenance”), adapt to changes in the application’s environment (“adaptive maintenance”), or adapt to changes in user requirements (“perfective maintenance”). The ongoing maintenance of existing software is an important issue, since maintenance activities can account for the majority of the costs of software (as much as two-thirds [124]). Software maintenance (also termed “software evolution”) is an active area of research for non-agent-based software.

The agent-oriented approach to design and implementation is inherently advantageous for maintenance, especially adaptive or perfective maintenance, due to its modular nature. This has been demonstrated to some extent by Bartish and Thevathayan [5] and by Jayatilleke et al. [75]. The former compared the effort and amount of code required to extend and modify a game implemented as both a finite-state machine and as an agent system, finding that the agent system was much more efficient. The latter extended an aircraft weather alerting system based on temperature and wind predictions to include alerts involving volcanic ash. This extension was built and integrated extremely efficiently and with little modification required to existing code.

However, little has been done specifically on software maintenance of agent-based software. Perhaps due to the relative youth of the field, there has been very little work on software maintenance of agent systems. In fact, the only work we are aware of on software maintenance for agent systems is by Dam et al. [32, 33].

Dam et al. deal with change propagation in design models. Change propagation is concerned with the issue that making changes to a software system involves making some initial changes, but these initial changes almost invariably have consequences, and require additional secondary changes to be made. Change propagation tools aim to support the software developer in identifying and making these secondary changes.

The approach used by Dam et al. is to focus on design models, rather than code, and to define consistency constraints (expressed in OCL) which capture

consistency requirements of the design – for example, that when a message type is defined to be received by a certain agent type, then that agent type must have a plan that deals with the message type. Change propagation is then performed by repairing violations of these constraints, caused by primary changes. For example, creating a new message and linking it to an agent violates the constraint above, and a possible secondary change is to define a new plan in the agent, and have this new plan handle the new message type.

The framework and techniques proposed by Dam et al. are generic, and have been applied to both agent-oriented designs (Prometheus) and object-oriented designs (UML). Interestingly, the underlying change propagation engine uses abstract repair plans expressed as BDI event-triggered plans. These plans are derived directly from the OCL constraints, and are provably sound and complete.

Evaluation has shown that the approach is effective in performing a significant proportion (around two-thirds) of secondary changes, including some that would be likely to be missed because they concern parts of the design that would not normally be considered by the designer (e.g., updating analysis-related design artifacts when making changes to the detailed design). In terms of efficiency and scalability, the approach is viable for small to medium design models, but further opportunities for efficiency improvement exist, since constraint checking is a bottleneck, and there exist known techniques for making constraint checking “instant” [48].

10 Comparing Methodologies

As noted in Section 1.1, the early years of work on agent-oriented software engineering saw the development of a large number of methodologies. This raised a question: given many competing methodologies, how could one select a methodology to use? This question saw the appearance of a body of work that focused on *comparing* different methodologies. Roughly speaking, this body of work developed around 2001–2003 [22, 34, 115, 118]. By the middle of the decade the area of AOSE methodologies had begun to consolidate, resulting in fewer methodologies being serious contenders for adoption, and therefore there was reduced interest in comparing methodologies, although there was some continued work (e.g., [121]).

All of the work in this area adopted a feature-based comparison approach. This approach compares methodologies by first identifying a set of features (e.g., [117]), such as which phases the methodology covered, whether it supported the design of proactive agents, whether it provided good traceability support, whether its notation was intuitive, and whether the methodology was well-documented. Each methodology being compared was then evaluated against the (typically fairly

long) list of features.

This exercise resulted in a large table with methodologies on one axis, features on the other, and each table cell containing an assessment such as “High” (i.e., the methodology provides a high level of support for the feature in question). One benefit of the feature-based approach to comparing methodologies is that for a given project, one could assess the needs of the project (e.g., “for this project we need a methodology that has strong support for designing complex interactions, but we don’t need support for proactive agents”). This assessment of important and less important features could be used, together with the large assessment table, to select a suitable methodology [92]. However, in order for this to work well, all potentially important features had to be identified, which tended to result in a long list of features being used.

Feature-based comparison suffers from a number of weaknesses. One is that typically the assessments are given in terms of coarse-grained levels (e.g., low/medium/high). This provides quite limited information on each feature, and how well it is supported. A more significant weakness of feature-based comparison is its subjectivity: the assignment of a rating to a given feature for a methodology is subjective, and not without controversy. It is not uncommon for a methodology’s creator to see an assessment table and object to certain ratings (“*but my methodology does support [feature X]!*”). Furthermore, even the creators of a given methodology were not reliable assessors. Dam & Winikoff [34] used a feature-based approach, and for each methodology asked its creators to assess the methodology against the various features. They found that in a number of cases where a methodology had more than one creator, the two creators assigned significantly different scores to their methodology’s support for features.

Although it may not be productive to try to directly compare methodologies on long lists of possible features, it is clear that some methodologies have specialized in particular aspects of AOSE, or particular kinds of agent systems. The ideas that have been developed in these methodologies may eventually be incorporated into other, more mainstream or general-purpose methodologies. Some specific aspects around which specialized methodologies have been developed include: organizations and norms (OperA [42], Moise+ [66]); open systems (SODA [86, 93]); strategies for negotiating agents (STRATUM [106]); emergent systems (ADELFE [101]); and stakeholder engagement (ROADMAP [77]).

11 Conclusions

The bulk of this chapter was devoted to describing the state of the art in agent-oriented software engineering. It is now time to look ahead and try to answer the question: What’s next? What are the current areas of research? The big

challenges?

We consider two broad areas for future work: those that require research, and those that address hurdles to wider adoption but are not research topics as such.

One area where we believe there is an urgent need for research is in understanding the benefits of the agent paradigm. Although agent technologies have been around for a while now, our understanding of the benefits that they offer, and of the context in which these benefits may be realized, is still, in essence, a collection of well-documented anecdotes in the form of case studies (e.g., [88, 105]). What is largely missing is measurement of the costs and benefits of agent technologies. Although some early results have been reported by Benfield et al. [7], they lack detail, and are only a small number of data points. Another possible area for work is conducting experiments comparing the development of systems using agent technologies, and using traditional approaches, along the lines of Prechelt's work comparing programming languages [104].

Another area for research concerns designing flexible interactions. As was discussed earlier in this chapter, most methodologies use interaction protocols that are message-centric (using a notation such as AUML's sequence diagrams). The issue is that this approach is not a good match for agent systems: message-centric interaction protocols define precisely and explicitly the legal message sequences, and tend to result in overly prescriptive designs that do not always support maximum flexibility, robustness, or autonomy [26]. There is ongoing work on developing alternative methods and models for designing agent interaction (e.g., [51, 52, 79, 80, 81, 134, 135], and see Chapter 3), but at present none of the approaches proposed have been widely accepted, and it could be argued that they are not yet ready for use on real systems.

More broadly, it has been observed [45] that there are different types of agent systems, and that in fact many of the well-known methodologies have focused on supporting the design of particular types of systems, typically those that are relatively closed, and which involve coarse-grained agents with a certain degree of cognitive capabilities (such as BDI agents). However, there are three types of agent systems that are not well-supported by current methodologies.

One such type of system is one in which there is a need to model a complex organizational structure, which may change during execution. There has been considerable work in recent years on organizational modeling (see Chapter 2), and this is gradually finding its way into methodologies (e.g., [42]).

Another type of system which is not well supported by current AOSE work is one involving large numbers of very simple agents that rely on emergence of desired behavior. Although there has been some work in this area (e.g., [98]), one issue is that the work on designing emergent agent systems is proceeding independently of work on the design of cognitive agent systems. However, ultimately,

what are needed are integrated methodologies that can support the design of systems with both emergent and cognitive aspects [99].

A third type of system is one in which agents are developed by different people and organizations, and in which agents join an “open” society of agents. Again, there has been some work on methodologies to support the design of open agent systems (e.g., [4, 116]), but more work is needed.

Thus, a significant longer-term challenge for the AOSE community is to develop methodologies (along with tools) that support the engineering of a wider range of types of agent systems, including emergent systems, open systems, and systems with complex and dynamic organizational aspects.

Finally, as noted in Section 8.2, obtaining assurance that an agent system will never malfunction is a significant challenge, given that agent systems are engineered to be adaptive, and to exhibit a complex range of possible behaviors. Munroe et al. [88, section 3.7.2] note that “... *validation through extensive tests was mandatory*” but go on to observe that “... *the task proved challenging for several reasons. First, agent-based systems explore realms of behavior outside people’s expectations and often yield surprises.*” Similarly, Hall et al. identify the question of how “... *the aggregate behavior of the agent-based system be guaranteed to meet all the system requirements?*” [59] as being a key obstacle to the adoption of agent technology in manufacturing.

As argued earlier, the complex range of behaviors that can be exhibited by agent systems makes manual testing an ineffective means of obtaining assurance, and so there is a need for research in this area. There are a number of approaches to be explored, including (fully) automated testing, formal verification, the use of safety cases, and end-to-end evidence. The issue of assurance is probably one of the most crucial areas for further development if multiagent systems are to be extensively used by industry. While there is some substantial work in some areas, much remains to be done to ensure that agent systems can be extensively and easily tested, giving some reasonable assurance of robustness in the face of the myriad of potential executions possible.

The areas of work described next are not major research challenges, but primarily involve the engineering, development, and integration of software (such as design tools). However, the following challenges *are* significant, and do need to be addressed in order to enable wider adoption of agent technologies.

One area of work concerns standardization. As noted earlier, the number of viable methodologies competing for adoption has shrunk in recent years, but there are still a number of methodologies; and while these methodologies are in some ways quite similar, there are still differences, some of which are significant, and others merely gratuitous. One area of work, therefore, is to work toward the development of a standardized methodology [39, 60, 96]. It is recognized that in

general there won't be a single methodology that will suit all users, but it should be possible to define a core methodology along with a number of variations or customizations for different purposes and settings.

One approach that needs to be mentioned with regard to standardization is *method engineering* [61]. Briefly, the idea is that methodologies are broken up into “fragments,” which describe either a process or a product resulting from carrying out a design process. These fragments are captured within a framework that includes a metamodel (e.g., SPEM¹²) defining the notions of work product, work process, etc. The key idea of method engineering is that a methodology instance is created by assembling it from a collection of fragments, held in a repository. We believe that there may be contexts in which method engineering is valuable and applicable, but we argue that in many cases, it is more appropriate to begin with a complete methodology and customize it as needed, rather than beginning with a repository of fragments. This is because assembling an effective and sensible methodology out of fragments is not only a significant effort, but doing so requires expertise and experience in AOSE development. This explains why method engineering has not seen widespread adoption [50]. For further information on method engineering in an AOSE context, see the recent survey by Cossentino et al. [30].

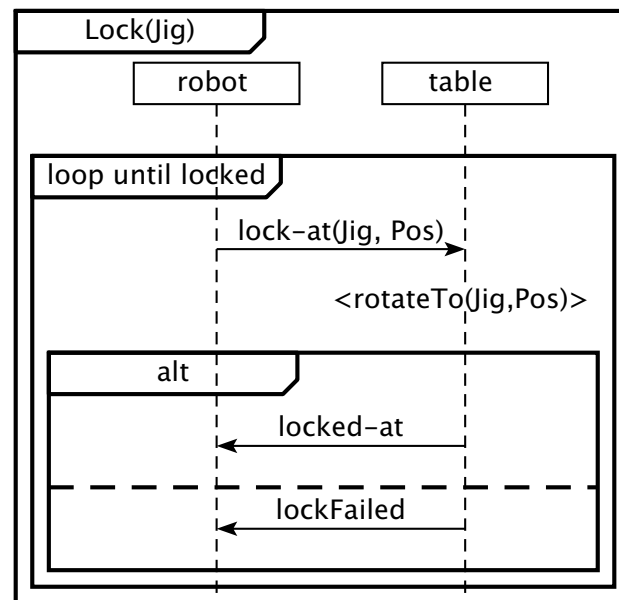
The wider industrial adoption of agent technologies is also hindered by a number of other factors [21, 55, 56, 125, 127]. One factor is that mainstream (including object-oriented) practices, standards, and tools need to be integrated¹³ with agent-oriented practices, standards, and tools. Companies and software engineers have a great deal invested in current (especially object-oriented) approaches to design and development. The more that agent-oriented approaches can leverage this existing expertise and be consistent with it when possible – while being clear about where agent-oriented design is unique and different – the more likely it is that industry will be willing to use and invest in agent technology. Also, real applications typically are not a standalone agent system, but involve a multiagent system that is part of a broader system, which includes object-oriented software, databases, and other components. So there is a real need for tools and methodologies that incorporate both agent and non-agent aspects. This line of work is arguably best driven by companies, rather than by academic researchers. However, if researchers can clearly articulate the value proposition of agent technologies, then companies may be more likely to invest in the integration of these technologies with existing tool sets and approaches.

¹²Software Process Engineering Metamodel.

¹³It is interesting to note that the issue of AOSE having limited industrial impact due to the lack of integration with mainstream approaches had been discussed at least as far back as 2002 [87].

12 Exercises

1. **Level 1** Extend and modify the presented design for the cell system to include a coordinator agent that will make the decision as to which part should be manufactured by the cell. This decision should be based on prioritizing orders that are highest priority. Orders arrive dynamically, and if a higher priority order comes in, the coordinator should shift to manufacturing parts for that order, once parts currently in production on the cell are completed.
2. **Level 1** Design a smart printer system that will monitor paper levels in printers, sending a message to refill as needed; and intelligently re-route jobs to a nearby printer if a given printer is overloaded, out of paper, or not functioning.
3. **Level 2** Extend the above design to incorporate automated calling of maintenance if a printer is not functioning, and maintenance of user statistics (pages printed/day by each user). Implement and document the system, ensuring that design and documentation are consistent.
4. **Level 2** Select three agent-oriented software engineering methodologies and compare them using existing criteria (e.g., [22, 115]).
5. **Level 2** Implement the holonic manufacturing design using a suitable programming language.
6. **Level 2** Extend the holonic manufacturing example with the ability to recover from failures, such as an attempt to join two parts failing, or the table jamming. Assume that when an action fails, the failure is reported to the system.
7. **Level 2** Extend the holonic manufacturing example with variations (e.g., manufacturing BAC as well as ABC).
8. **Level 3** Find a way of convincing a skeptical but rational critic that your holonic manufacturing implementation will never misbehave, i.e., that the table never moves while loading, unloading, or joining are being done. Are the synchronization constraints sufficient?
9. **Level 2** Consider the following version of the Lock protocol. If this protocol is used instead of the one in Figure 15.6 (on page 713), is it possible for a rotation to occur at the same time as another action?



10. **Level 2** Assume that we add the possibility of resting a partially finished composite piece by having the `unload()` action do either an `unload(bin)` or an `unload(shelf)` action. The `load(part)` action can then load a composite part (from the shelf) as well as single parts. How will this change the design? Extend the existing design, making as few changes as possible to what is there, to allow this extension. Include in your design some mechanism for deciding when it makes sense to do this.
11. **Level 3** Expand the simple cell design discussed in this chapter to capture a hierarchical organization of agents, where the bottom level contains the agents described in this chapter that control specific machines; then there would be a layer of cell coordinators that each manage their cell, and, above this, a supervisor agent that does the prioritization of orders. Investigate some of the approaches to team and organizational structures, and suggest how the basic methodology (process and notation) described in this chapter could be extended to incorporate the development of organizational or team design.
12. **Level 3** Develop a smart meeting scheduler that schedules meetings between participants based on their availability and the availability of rooms. The system should be able to automatically reschedule a meeting if a previously booked meeting room or participant is subsequently required for a more important meeting, and if the new, more important meeting cannot be scheduled within the required timeframe without modifying some other

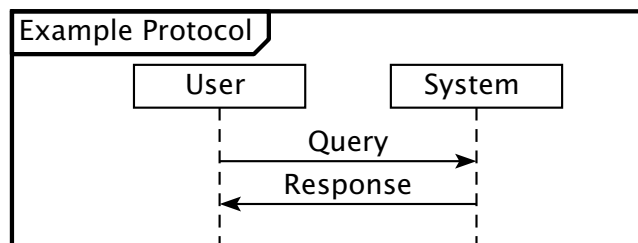
meeting. Rescheduling should be kept to a minimum, especially rescheduling of time.

13. **Level 4** Design a multiagent system for managing the logistics of a courier system that must continuously take in and assign jobs in an efficient manner, where jobs have a pickup and delivery location, and also a priority level of 1, 2, or 3. Compare the resulting approach with state-of-the-art approaches within logistics.
14. **Level 4** Interactions between agents in a complex system, in which multiple things may be happening simultaneously, can lead to race conditions and deadlocks. However such problems can be very difficult to find and test for. Consider how you might approach the development of test cases that would be likely to uncover such problems if they existed.

Appendix: Agent UML Sequence Diagram Notation

This appendix briefly explains the essential elements of the Agent Unified Modeling Language (AUML) sequence diagram notation, which is used in Section 5 to specify interaction protocols. For brevity we use “AUML” as shorthand for “AUML sequence diagram,” since the sequence diagram is arguably the most used of the diagram types proposed by AUML. For more details, see the AUML website (www.auml.org) or relevant publications [68, 91]

The AUML sequence diagram notation is an extension of the interaction diagram. As such, an AUML sequence diagram has lifelines for each agent with messages depicted by arrows between the lifelines and with time increasing as one moves downward. An interaction protocol is placed within a frame, with the name of the protocol in the “tag” at the top left side of the frame. The example below shows a User agent sending a Query message to a System agent followed by the System agent sending a Response message to the User agent.

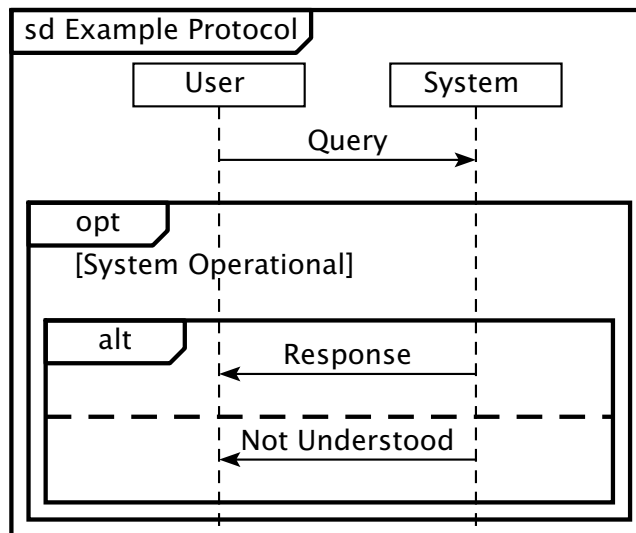


The primary way that AUML allows alternatives, parallelism, etc., to be specified is using *boxes*. A box is a region within the sequence diagram that contains messages and may contain nested boxes. Each box has a tag that describes the

type of box (for example, Alternative, Parallel, Option, etc.). A box can affect the interpretation of its contents in a range of ways, depending on its type. For example, an Option box (abbreviated “opt”) indicates that its contents may be executed as normal, or may not be executed at all (in which case the interpretation of the sequence diagram continues after the Option box).

Whether or not the box is executed can be specified by *guards*. A guard, denoted by text in square brackets, indicates a condition that must be true in order for the Option box to execute.

Most box types can be divided into *regions*, indicated by heavy horizontal dashed lines. For example, an Alternative box (abbreviated “alt”) can have a number of regions (each with its own guard) and exactly one region will be executed. The example below shows an example of nested boxes. The Option box indicates that nothing happens if the system is not operational. If the system is operational, then we have two alternatives (separated by a horizontal heavy dashed line). The first alternative is that the System sends the user a Response message. The second alternative is that the System indicates that the user’s query was not understood.



The following are some of the box types that are defined by AUML (and include all of the box types that we use):

- **Alternative:** Specifies that one of the box’s regions occurs. One of the regions may have “else” as the guard.
- **Option:** Can only have a single region. Specifies that this region may or may not occur.
- **Parallel:** Specifies that each of the regions takes place simultaneously and the sequence of messages is interleaved.

- **Loop:** Can only have a single region. Specifies that the region is repeated some number of times. The tag gives the type (“Loop”) and also an indication of the number of repetitions, which can be a fixed number (or a range) or a Boolean condition.
- **Ref:** This box type is a little different in that it doesn’t contain subboxes or messages. Instead, it contains the name of another protocol. This is basically a form of procedure call – the interpretation of the Ref box is obtained by replacing it with the protocol it refers to.

Finally, in our use of the AUML sequence diagram notation to capture interaction protocols, we find it useful to be able to indicate the places in a protocol where actions are performed. Since the AUML sequence diagram notation does not provide a way to do this, we have extended the notation to do so. We place an action indicator, the name of the action in angled brackets (“<action>”), on an agent’s lifeline to indicate that the agent performs the action.

References

- [1] N. Alechina, M. Dastani, B.S. Logan, and J.-J. Ch. Meyer. A logic of agent programs. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pages 795–800, 2007.
- [2] N. Alechina, M. Dastani, B.S. Logan, and J.-J. Ch. Meyer. Reasoning about agent deliberation. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings, Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 16–26, 2008.
- [3] Natasha Alechina, Mehdi Dastani, Brian Logan, and John-Jules Ch. Meyer. Reasoning about agent execution strategies (short paper). In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1455–1458, 2008.
- [4] Josep Lluís Arcos, Marc Esteva, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Engineering open environments with electronic institutions. *Eng. Appl. of AI*, 18(2):191–204, 2005.
- [5] Arran Bartish and Charles Thevathayan. BDI agents for game development. In *AA-MAS ’02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 668–669, 2002.
- [6] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Wiley, 2007.

- [7] Steve S. Benfield, Jim Hendrickson, and Daniel Galanti. Making a strong business case for multiagent technology. In Peter Stone and Gerhard Weiss, editors, *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 10–15. ACM Press, 2006.
- [8] Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems*. Kluwer Academic Publishing (New York), 2004.
- [9] Rafael Bordini, Mehdi Dastani, and Michael Winikoff. Current issues in multi-agent systems development. In *Post-proceedings of the Seventh Annual International Workshop on Engineering Societies in the Agents World.*, volume 4457 of *LNAI*, pages 38–61, 2007.
- [10] Rafael H. Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model checking AgentSpeak. In *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 409–416, 2003.
- [11] Rafael H. Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. Verifying multi-agent programs by model checking. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 12:239–256, 2006.
- [12] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.
- [13] Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for BDI agent systems. In *Programming Multiagent Systems, Second International Workshop (ProMAS'04)*, volume 3346 of *LNAI*, pages 44–65. Springer, Berlin, 2005.
- [14] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int. Journal of Cooperative Information Systems*, 6(1):67–94, 1997.
- [15] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8:203–236, May 2004.
- [16] Birgit Burmeister, Michael Arnold, Felicia Copaciu, and Giovanni Rimassa. BDI-agents for agile goal-oriented business processes. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Industry Track*, pages 37–44. IFAAMAS, 2008.
- [17] P. Burrafato and M. Cossentino. Designing a multi-agent solution for a bookstore with the PASSI methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto (Ontario, Canada) at CAiSE'02, 27-28 May 2002.

- [18] Paolo Busetta, Ralph Rönquist, Andrew Hodgson, and Andrew Lucas. JACK Intelligent Agents – Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998. Available from <http://www.agent-software.com>.
- [19] Stefan Bussmann. An agent-oriented architecture for holonic manufacturing control. In *Proceedings of the 1st International Workshop on Intelligent Manufacturing Systems*, pages 1–12. EPFL, Lausanne, Switzerland, 1998.
- [20] G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci. Multi-agent systems implementation and testing. In *Fourth International Symposium: From Agent Theory to Agent Implementation*, pages 14–16, 2004.
- [21] Monique Calisti and Giovanni Rimassa. Opportunities to support the widespread adoption of software agent technologies. *Int. J. Agent-Oriented Software Engineering*, 3(4):411–415, 2009.
- [22] L. Cernuzzi and G. Rossi. On the evaluation of agent oriented modeling methods. In *Proceedings of Agent Oriented Methodology Workshop*, Seattle, November 2002.
- [23] Radovan Cervenka and Ivan Trencansky. *The Agent Modeling Language AML: A Comprehensive Approach to Modeling Multi-Agent Systems*. Birkhäuser, 2007. ISBN 978-3-7643-8395-4.
- [24] Radovan Cervenka, Ivan Trencanský, and Monique Calisti. Modeling social aspects of multi-agent systems: The AML approach. In Jörg P. Müller and Franco Zambonelli, editors, *AOSE*, volume 3950 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2005.
- [25] Radovan Cervenka, Ivan Trencanský, Monique Calisti, and Dominic A. P. Greenwood. AML: Agent modeling language toward industry-grade agent-based modeling. In James Odell, Paolo Giorgini, and Jörg P. Müller, editors, *AOSE*, volume 3382 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2004.
- [26] Christopher Cheong and Michael Winikoff. Hermes: Designing flexible and robust agent interactions. In Virginia Dignum, editor, *Multi-Agent Systems – Semantics and Dynamics of Organizational Models*, chapter 5, pages 105–139. IGI, 2009.
- [27] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: Algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.
- [28] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000. ISBN 978-0-262-03270-4.

- [29] Massimo Cossentino. From requirements to code with the PASSI methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, pages 79–106. Idea Group Inc., 2005.
- [30] Massimo Cossentino, Marie-Pierre Gleizes, Ambra Molesini, and Andrea Omicini. Processes engineering and AOSE. In Marie-Pierre Gleizes and Jorge J. Gómez-Sanz, editors, *Post-proceedings of Agent-Oriented Software Engineering (AOSE 2009)*, volume 6038 of *LNCS*, pages 191–212, 2011.
- [31] Wim Coulier, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, and Philip Massonet. MESSAGE: a methodology for the development of agent-based applications. In Bergenti et al. [8], chapter 9.
- [32] Hoa Khanh Dam and Michael Winikoff. An agent-oriented approach to change propagation in software maintenance. *Journal of Autonomous Agents and Multi-Agent Systems*, 23(3):384–452, 2011.
- [33] Khanh Hoa Dam. *Supporting Software Evolution in Agent Systems*. PhD thesis, RMIT University, Australia, 2008.
- [34] Khanh Hoa Dam and Michael Winikoff. Comparing agent-oriented methodologies. In Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, editors, *AOIS*, volume 3030 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2003.
- [35] Mehdi Dastani, Koen V. Hindriks, and John-Jules Ch. Meyer, editors. *Specification and Verification of Multi-agent systems*. Springer, Berlin/Heidelberg, 2010.
- [36] Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch. Meyer. Goal types in agent programming. In *Proceedings of the 17th European Conference on Artificial Intelligence 2006 (ECAI'06)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 220–224. IOS Press, 2006.
- [37] Scott A. DeLoach. Multiagent systems engineering: A methodology and language for designing agent systems. In *Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle, WA, May 1998.
- [38] Scott A. DeLoach. Developing a multiagent conference management system using the O-MaSE process framework. In M. Luck and L. Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science (LNCS)*, pages 168–181. Springer-Verlag, 2008.
- [39] Scott A. DeLoach. Moving multi-agent systems from research to practice. *Int. J. Agent-Oriented Software Engineering*, 3(4):378–382, 2009.
- [40] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

- [41] Louise A. Dennis, Berndt Farwer, Rafael H. Bordini, and Michael Fisher. A flexible framework for verifying agent programs. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1303–1306. IFAAMAS, 2008.
- [42] Virginia Dignum. *A Model for Organizational Interaction: Based On Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.
- [43] Virginia Dignum, editor. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Hershey, New York, 2009. ISBN 978-1-60566-256-5.
- [44] Virginia Dignum and Frank Dignum. The knowledge market: Agent-mediated knowledge sharing. In *Proceedings of the Third International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 03)*, pages 168–179, June 2003.
- [45] Virginia Dignum and Frank Dignum. Designing agent systems: State of the practice. *International Journal of Agent-Oriented Software Engineering*, 4(3):224–243, 2010.
- [46] Klaus Dorer and Monique Calisti. An adaptive solution to dynamic transport optimization. In Michal Pěchouček, Donald Steiner, and Simon G. Thompson, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands – Special Track for Industrial Applications*, pages 45–51. ACM, 2005.
- [47] Simon Duff, James Harland, and John Thangarajah. On proactive and maintenance goals. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1033–1040. ACM, 2006.
- [48] Alexander Egyed. Instant consistency checking for the UML. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 381–390, New York, NY, USA, 2006. ACM.
- [49] Jonathan Ezekiel and Alessio Lomuscio. Combining fault injection and model checking to verify fault tolerance in multi-agent systems. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 113–120, 2009.
- [50] Brian Fitzgerald, Nancy L. Russo, and Tom O’Kane. Software development method tailoring at Motorola. *Commun. ACM*, 46:64–70, April 2003.
- [51] Roberto A. Flores and Robert C. Kremer. A pragmatic approach to build conversation protocols using social commitments. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1242–1243, July 2004.

- [52] Roberto A. Flores and Robert C. Kremer. A pragmatic approach to build conversation protocols using social commitments. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1242–1243. ACM Press, 2004.
- [53] Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenon, and Jorge Valenzuela. O-MaSE: A customizable approach to developing multiagent development processes. In M. Luck and L. Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science (LNCS)*, pages 1–15. Springer-Verlag, 2008.
- [54] Francisco Garijo, Jorge J. Gomez-Sanz, Juan Pavon, and Phillipe Massonet. Multi-agent system organization: An engineering perspective. In *Proceedings of Modelling Autonomous Agents in a Multi-Agent World, 10th European Workshop on Multi-Agent Systems (MAAMAW'2001)*, pages 101–108, May 2001.
- [55] Michael Georgeff. The gap between software engineering and multi-agent systems: Bridging the divide. *Int. J. Agent-Oriented Software Engineering*, 3(4):391–396, 2009.
- [56] Aditya Ghose. Industry traction for MAS technology: Would a rose by any other name smell as sweet? *Int. J. Agent-Oriented Software Engineering*, 3(4):397–401, 2009.
- [57] Jorge J. Gómez-Sanz, Juan Botía, Emilio Serrano, and Juan Pavón. Testing and Debugging of MAS Interactions with INGENIAS. In Michael Luck and Jorge J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, pages 199–212, Berlin, Heidelberg, 2009. Springer-Verlag.
- [58] Jorge J. Gómez-Sanz and Rubén Fuentes-Fernández. Agent-oriented software engineering with INGENIAS. In *Ibero-American Workshop on Multi-Agent Systems*, Seville, Spain, 12/11/2002 2002.
- [59] Kenwood H. Hall, Raymond J. Staron, and Pavel Vrba. Experience with holonic and agent-based control systems and their adoption by industry. In V. Mařík, R.W. Brennan, and M. Pěchouček, editors, *Holonic and Multi-Agent Systems for Manufacturing, Proceedings of the Second International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'05)*, volume 3593 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 1–10, 2005.
- [60] Brian Henderson-Sellers. Consolidating diagram types from several agent-oriented methodologies. In *Proceeding of the 2010 Conference on New Trends in Software Methodologies, Tools and Techniques (SoMeT)*, pages 293–345, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

- [61] Brian Henderson-Sellers and Jolita Ralyté. Situational method engineering: State-of-the-art review. *J. UCS*, 16(3):424–478, 2010.
- [62] Koen V. Hindriks, Wiebe van der Hoek, and M. Birna van Riemsdijk. Agent programming with temporally extended goals. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 137–144. IFAAMAS, 2009.
- [63] K.V. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Intelligent Agents VI – Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL’2000)*. Springer Verlag, 2001.
- [64] Bryan Horling and Victor Lesser. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.
- [65] J.F. Hübner, J.S. Sichman, and O. Boissier. Developing organised multiagent systems using the MOISE⁺ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.
- [66] Jomi Hübner, Olivier Boissier, and Rafael Bordini. From organisation specification to normative programming in multi-agent organisations. In Jürgen Dix, João Leite, Guido Governatori, and Wojtek Jamroga, editors, *Computational Logic in Multi-Agent Systems*, volume 6245 of *Lecture Notes in Computer Science*, pages 117–134. Springer Berlin / Heidelberg, 2010.
- [67] Marc-Philippe Huget and James Odell. Representing agent interaction protocols with agent UML. In *Proceedings of the Fifth International Workshop on Agent-Oriented Software Engineering (AOSE)*, July 2004.
- [68] Marc-Philippe Huget, James Odell, Øystein Haugen, Mariam “Misty” Nodine, Stephen Cranefield, Renato Levy, and Lin Padgham. 7FIPA modeling: Interaction diagrams. On www.auml.org under Working Documents, 2003. FIPA Working Draft (version 2003-07-02).
- [69] Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A survey of agent-oriented methodologies. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag: Heidelberg, Germany, 1999.
- [70] Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. A methodological proposal for multiagent systems development extending CommonKADS. In *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.

- [71] Carlos Argel Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In *Agent Theories, Architectures, and Languages*, pages 313–327, 1997.
- [72] Daniel Jackson. A direct path to dependable software. *Communications of the ACM*, 52(4):78–88, April 2009.
- [73] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press Series. ACM Press, 1992.
- [74] J. Jarvis, R. Rönnquist, D. McFarlane, and L. Jain. A team-based holonic approach to robotic assembly cell control. *Journal of Network and Computer Applications*, 29(2-3):160 – 176, 2006. Innovations in agent collaboration.
- [75] Gaya Jayatilleke, Lin Padgham, and Michael Winikoff. A model driven development toolkit for domain experts to modify agent based systems. In Lin Padgham and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VII: 7th International Workshop, AOSE 2006, Hakodate, Japan, May 2006, Revised and Invited Papers*, 2006.
- [76] Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science (LNCS)*, pages 364–390. Springer, 2007.
- [77] T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, pages 3–10. ACM Press, 2002.
- [78] David Kinny, Michael Georgeff, and Anand Rao. A methodology and modelling technique for systems of BDI agents. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1996.
- [79] Sanjeev Kumar and Philip R. Cohen. STAPLE: An agent programming language based on the joint intention theory. In *Proceedings of the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2004)*, pages 1390–1391. ACM Press, July 2004.
- [80] Sanjeev Kumar, Philip R. Cohen, and Marcus J. Huber. Direct execution of team specifications in STAPLE. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, pages 567–568. ACM Press, July 2002.

- [81] Sanjeev Kumar, Marcus J. Huber, and Philip R. Cohen. Representing and executing protocols as joint actions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 543 – 550, Bologna, Italy, 15 – 19 July 2002. ACM Press.
- [82] Magnus Ljungberg and Andrew Lucas. The OASIS air-traffic management system. In *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence (PRICAI '92)*, Seoul, Korea, 1992.
- [83] Chi Keen Low, Tsong Yueh Chen, and Ralph Rönquist. Automated test case generation for BDI agents. *Autonomous Agents and Multi-Agent Systems*, 2(4):311–332, 1999.
- [84] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18, 2003.
- [85] T. Miller, L. Padgham, and J. Thangarajah. Test coverage criteria for agent interaction testing. In Danny Weyns and Marie-Pierre Gleizes, editors, *Proceedings of the 11th International Workshop on Agent-Oriented Software Engineering*, pages 1–12, 2010.
- [86] Ambra Molesini, Enrico Denti, and Andrea Omicini. Agent-based conference management: A case study in SODA. *IJAOSE*, 4(1):1–31, 2010.
- [87] Jörg P. Müller and Bernhard Bauer. Agent-Oriented Software Technologies: Flaws and Remedies. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002, Revised Papers and Invited Contributions*, volume 2585 of *LNCS*, pages 21–227. Springer-Verlag, 2003.
- [88] S. Munroe, T. Miller, R.A. Belecheanu, M. Pechoucek, P. McBurney, and M. Luck. Crossing the agent technology chasm: Experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, 21(4):345–392, 2006.
- [89] J. Mylopoulos, J. Castro, and M. Kolp. Tropos: Toward agent-oriented information systems engineering. In *Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS2000)*, June 2000.
- [90] Cu D. Nguyen, Anna Perini, and Paolo Tonella. Goal-Oriented Testing for MASs. *International Journal of Agent-Oriented Software Engineering*, 4(1):79–109, 2010.
- [91] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, 2000.

- [92] S. A. O'Malley and S. A. DeLoach. Determining when to use an agent-oriented software engineering. In *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, pages 188–205, Montreal, May 2001.
- [93] Andrea Omicini. SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. In *AOSE*, pages 185–193, 2000.
- [94] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In *Third International Workshop on Agent-Oriented Software Engineering*, July 2002.
- [95] Lin Padgham and Michael Winikoff. *Developing intelligent agent systems: A practical guide*. John Wiley & Sons, Chichester, 2004. ISBN 0-470-86120-7.
- [96] Lin Padgham, Michael Winikoff, Scott DeLoach, and Massimo Cossentino. A unified graphical notation for AOSE. In Michael Luck and Jorge J. Gomez-Sanz, editors, *Proceedings of the Ninth International Workshop on Agent-Oriented Software Engineering*, pages 116–130, Estoril, Portugal, May 2008.
- [97] H. Van Dyke Parunak. “Go to the Ant”: Engineering Principles from Natural Multi-Agent systems. *Annals of Operations Research*, 75:69–101, 1997.
- [98] H. Van Dyke Parunak and Sven A. Brueckner. Engineering swarming systems. In Bergenti et al. [8], chapter 17, pages 341–376.
- [99] H. Van Dyke Parunak, Paul Nielsen, Sven Brueckner, and Rafael Alonso. Hybrid multi-agent systems: Integrating swarming and BDI agents. In Sven Brueckner, Salima Hassas, Márk Jelasity, and Daniel Yamins, editors, *Engineering Self-Organising Systems, 4th International Workshop, ESOA 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers*, volume 4335 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2007.
- [100] Juan Pavón, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. *The INGENIAS Methodology and Tools*, article IX, pages 236–276. Idea Group Publishing, 2005.
- [101] Gauthier Picard and Marie-Pierre Gleizes. The ADELFE methodology: Designing adaptive cooperative multi-agent systems. In Bergenti et al. [8], chapter 8.
- [102] Alexander Pokahr and Lars Braubach. A survey of agent-oriented development tools. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-agent Programming: Languages, Tools, and Applications*, chapter 9, pages 289–329. Springer, 2009.
- [103] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El

- Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. Springer, 2005.
- [104] L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, oct 2000.
- [105] Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 17:397–431, 2008.
- [106] Iyad Rahwan, Liz Sonenberg, Nicholas R. Jennings, and Peter McBurney. Stratum: A methodology for designing heuristic agent negotiation strategies. *Applied Artificial Intelligence*, 21(6):489–527, 2007.
- [107] Franco Raimondi and Alessio Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *J. Applied Logic*, 5(2):235–251, 2007.
- [108] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319. The MIT Press, 1995.
- [109] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Give agents their artifacts: The A&A approach for engineering working environments in MAS. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-agent Systems*, pages 613–615, New York, NY, USA, 2007. ACM.
- [110] Collette Rolland, Georges Grosz, and Régis Kla. Experience with goal-scenario coupling in requirements engineering. In *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering (RE’99)*, 1999.
- [111] John Rushby. A safety-case approach for certifying adaptive systems. In *AIAA Infotech@Aerospace Conference*, April 2009.
- [112] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.
- [113] Nathan Schurr, Janusz Marecki, John P. Lewis, Milind Tambe, and Paul Scerri. The DEFECTO system: Coordinating human-agent teams for the future of disaster response. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 197–215. Springer, 2005.

- [114] S. Shapiro, Y. Lespérance, and H.J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, pages 19–26. ACM New York, NY, USA, 2002.
- [115] Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 624–631. ACM Press, May 2001.
- [116] Carles Sierra, John Thangarajah, Lin Padgham, and Michael Winikoff. Designing institutional multi-agent systems. In Lin Padgham and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VII: 7th International Workshop, AOSE 2006, Hakodate, Japan, May 2006, Revised and Invited Papers*, volume 4405, pages 84–103. Springer LNCS, 2007.
- [117] Arnon Sturm and Onn Shehory. A framework for evaluating agent-oriented methodologies. In Paolo Giorgini and Michael Winikoff, editors, *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems*, pages 60–67, Melbourne, Australia, 2003.
- [118] Jan Sudeikat, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Evaluation of agent-oriented software methodologies: Examination of the gap between modeling and platform. In Paolo Giorgini, Jörg Müller, and James Odell, editors, *Agent-Oriented Software Engineering (AOSE)*, 2004.
- [119] John Thangarajah, Gaya Jayatilleke, and Lin Padgham. Scenarios for system requirements traceability and testing. In Tumer, Yolum, Sonenberg, and Stone, editors, *Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 285–292. IFAAMAS, 2011.
- [120] Ali Murat Tiryaki, Sibel Öztuna, Oguz Dikenelli, and Riza Cenk Erdur. SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems. In *AOSE*, pages 156–173, 2006.
- [121] Quynh-Nhu Numi Tran and Graham C. Low. Comparison of ten agent-oriented methodologies. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, chapter XII, pages 341–367. Idea Group Publishing, 2005.
- [122] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01)*, pages 249–263, Toronto, 2001.
- [123] M. Birna van Riemsdijk, Mehdi Dastani, and Michael Winikoff. Goals in agent systems: A unifying framework. In *Proceedings of the Seventh International Joint*

- Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 713–720, 2008.
- [124] Hans van Vliet. *Software Engineering: Principles and Practice*. John Wiley & Sons, Inc., 2nd edition, 2001. ISBN 0471975087.
- [125] Danny Weyns, Alexander Helleboogh, and Tom Holvoet. How to get multi-agent systems accepted in industry? *Int. J. Agent-Oriented Software Engineering*, 3(4):383–390, 2009.
- [126] Michael Winikoff. JACKTM Intelligent Agents: An industrial strength platform. In Rafael Bordini and et al., editors, *Multi-Agent Programming*, pages 1–29. Kluwer, 2005.
- [127] Michael Winikoff. Future directions for agent-based software engineering. *Int. J. Agent-Oriented Software Engineering*, 3(4):402–410, 2009.
- [128] Michael Winikoff. Assurance of Agent Systems: What Role should Formal Verification play? In Mehdi Dastani, Koen V. Hindriks, and John-Jules Ch. Meyer, editors, *Specification and Verification of Multi-agent systems*, chapter 12, pages 353–383. Springer, 2010.
- [129] Michael Winikoff and Stephen Cranefield. On the testability of BDI agent systems. Information Science Discussion Paper Series 2008/03, University of Otago, Dunedin, New Zealand, 2008.
- [130] Michael Winikoff, Lin Padgham, and James Harland. Simplifying the development of intelligent agents. In Markus Stumptner, Dan Corbett, and Mike Brooks, editors, *AI2001: Advances in Artificial Intelligence. 14th Australian Joint Conference on Artificial Intelligence*, pages 555–568. Springer, LNAI 2256, December 2001.
- [131] M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.
- [132] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons (Chichester, England), 2002. ISBN 0 47149691X.
- [133] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons. Model checking multi-agent systems with MABLE. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 952–959, 2002.
- [134] Pinar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534, 2002.

- [135] Pinar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence (AMAI), Special Issue on Computational Logic in Multi-Agent Systems*, 2004.
- [136] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.
- [137] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.
- [138] Zhiyong Zhang. *Automated Unit Testing of Agent Systems*. PhD thesis, RMIT University, Melbourne, Australia, 2011.
- [139] Zhiyong Zhang, John Thangarajah, and Lin Padgham. Automated testing for intelligent agent systems. In *AOSE*, pages 66–79, 2009.
- [140] M. Zheng and V.S. Alagar. Conformance testing of BDI properties in agent-based software systems. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC)*, pages 457–464. IEEE Computer Society Press, 2005.

Part VI

Technical Background

Chapter 16

Logics for Multiagent Systems

Wiebe van der Hoek and Michael Wooldridge

1 Introduction

If one wants to reason about an agent or about a multiagent system, then logic can provide a convenient and powerful tool. First of all, logics provide a *language* with which to *specify* properties: properties of the agent, of other agents, and of the environment. Ideally, such a language also provides a means to then *implement* an agent or a multiagent system, either by somehow executing the specification, or by transforming the specification into some computational form. Second, given that such properties are expressed as logical formulas that are part of some *inference* system, they can be used to *deduce* other properties. Such reasoning can be part of an agents' own capabilities, but it can also be done by the system designer, the analyst, or the potential user of (one of) the agents. Third, logics provide a formal *semantics* in which the sentences from the language get a precise meaning: if one manages to come up with a semantics that closely models (part of) the system under consideration, one then can *verify* properties of a particular system (model checking). This, in a nutshell, sums up the three main characteristics of any logic (language, deduction, semantics), as well as the three main roles that logics play in system development (specification, execution, and verification) (see also Chapter 14 for a discussion on the role of logic in specification and verification of agent systems).

If the role and value of logic for multiagent systems (MAS) is clear, then why are there so many logics for MAS, with new variants proposed at almost every multiagent conference? What most of these logics compete for is a proper balance

between *expressiveness*, on the one hand, and *complexity* on the other. What kinds of properties are interesting for the scenarios of interest, and how can they be “naturally” and concisely expressed? Then, how complex are the formalisms, in terms of how easily the key relevant properties can be expressed and grasped by human users, and how costly is it to use the formalism when doing verification or reasoning with it? Of course, the complexity of the logic under consideration will be related to the complexity of the domain it tries to formalize.

In multiagent research, this complexity often depends on a number of issues. Let us illustrate this with a simple example, say the context of traffic. If there is only one agent involved, the kinds of things we would like to represent to model the agent’s sensing, planning, and acting could probably be done in a simple propositional logic, using atoms like g_n (light n is green), o_k (gate k is open), and $e_{i,m}$ (agent i enters through gate m). However, taking the agent perspective seriously, one quickly realizes that we need more: there might be a difference between what *actually* is the case and what the agent *believes* is the case, and also between what the agent believes to hold and what the agent would *like* to be true – otherwise there would be no reason to act! So, we would like to be able to say things like $\neg o_k \wedge B_i o_k \wedge D_i(o_k \rightarrow e_{i,k})$ (although the gate is closed, the agent believes it is open and desires to establish that in that case it enters through it).

Things get more interesting when several agents enter the scene. Not only does our agent i need a model of its operational environment, but also a model of j ’s mental state, the latter involving a model of i ’s mental state. We can then express properties like $B_i B_j g_j \rightarrow stop_i$ (if i believes that j believes that the latter’s light is green, then i will stop). Higher-order information enters the picture, and there is no *a priori* level where this would stop (this is for instance important in reasoning about games: see the discussion on common knowledge for establishing a Nash equilibrium [12]). In our simple traffic scenario,¹ assume that i is a pedestrian who approaches a crossing without traffic lights while motorist j advances as well. It might well be that both believe that j is advancing ($B_i ad_j \wedge B_j ad_j$) so, being the more vulnerable party, one would expect that i will wait. However, if i has a strong desire to not lose time with waiting, it may try to make j stop for i by “making j believe that i is not aware of j ” ($B_i ad_j \wedge B_j ad_j \wedge D_i B_i \neg B_j B_i ad_j$), i.e., it is i ’s desire to be convinced that j does not believe that i is aware of j advancing (i can avoid eye contact, for instance, and approach the crossing in a determined way, the aim of this being that j prefers no accident over being involved in one and hence will stop). In other words, i plans to act contrary to its own beliefs.

Another dimension that makes multiagent scenarios (and hence their logics) complex is their *dynamics*: the world changes (this is arguably the goal of the whole enterprise) and the information, the desires, and the goals of agents

¹This example is adapted from one given in a talk by Rohit Parikh.

change as well. So, we need tools to reason either about time, or else about actions explicitly. A designer of a crossing is typically interested in properties like $A \Box \neg(g_i \wedge g_j)$ (this is a *safety* property requiring that in all computations, it is always the case that no two lights are green) and $cl_i \rightarrow A \Diamond g_i$ (this is a *liveness* property expressing that if there is a car in lane i , then no matter how the system evolves, eventually light i will be green). Combining the aspects of multiagents and dynamics is where things really become intriguing: there is not just “a future,” or a “possible future depending on an agent’s choice”: what the future looks like will depend on the choices of several agents at the same time. We will come across languages in which one can express $\neg \langle\langle i \rangle\rangle \Diamond s \wedge \neg \langle\langle j \rangle\rangle \Diamond s \wedge \langle\langle i, j \rangle\rangle \Diamond s$ (although i cannot bring about that eventually everybody has safely crossed the road, and neither can j , by cooperating together, i and j can guarantee that they both cross in a safe manner).

Logics for MAS are often some variant of *modal logic*; to be more precise, they are all *intensional*, contrary to propositional and first-order logic, which are *extensional*. A logic is extensional if the truth-value of a formula is completely determined given the truth-value of all its constituents, the sub-formulas. If we know the truth-value of p and q , we also know that of $(p \wedge q)$, and of $\neg p \rightarrow (q \rightarrow p)$. For logics of agency, extensionality is often not realistic. It might well be that “rain in Utrecht” and “rain in Liverpool” are both true, while our agent knows one without the other. Even if one is given the truth value of p and of q , one is not guaranteed to be able to tell whether $B_i(p \vee q)$ (agent i believes that $p \vee q$), whether $\Diamond(p \wedge q)$ (eventually, both p and q), or whether $B_i \Box(p \rightarrow B_h q)$ (i believes that it is always the case that as soon as p holds, agent h believes that q).

These examples make clear why extensional logics are so popular for multi-agent systems. However, the most compelling argument for using modal logics for modeling the scenarios we have in mind lies probably in the semantics of modal logics. They are built around the notion of a “state,” which can be the state of a system, of a processor, or a situation in a scenario. Considering several states at the same time is then rather natural, and usually, many of them are “related”: some because they “look the same” for a given agent (they define its beliefs), some because they are very attractive (they comprise its desires), or some of them may represent some state of affairs in the future (they model possible evolutions of the system). Finally, some states are reachable only when certain agents make certain decisions (those states determine what coalitions can achieve).

In the remainder of this section, we demonstrate some basic languages, inference systems, and semantics that are foundational for logics of agency. The rest of the chapter is then organized along two main streams, reflecting two trends in multiagent systems research when it comes to representing and reasoning about environments:

Knowledge Axioms	
<i>Kn1</i>	φ where φ is a propositional tautology
<i>Kn2</i>	$K_i(\varphi \rightarrow \psi) \rightarrow (K_i\varphi \rightarrow K_i\psi)$
<i>Kn3</i>	$K_i\varphi \rightarrow \varphi$
<i>Kn4</i>	$K_i\varphi \rightarrow K_iK_i\varphi$
<i>Kn5</i>	$\neg K_i\varphi \rightarrow K_i\neg K_i\varphi$
Rules of Inference	
<i>MP</i>	$\vdash \varphi, \vdash (\varphi \rightarrow \psi) \Rightarrow \vdash \psi$
<i>Nec</i>	$\vdash \varphi \Rightarrow \vdash K_i\varphi$

Figure 16.1: An inference system for knowledge.

Cognitive models of rational action: The first main strand of research in representing multiagent systems focuses on the issue of representing the *attitudes* of agents within the system: their beliefs, aspirations, intentions, and the like. The aim of such formalisms is to derive a model that predicts how a *rational agent* would go from its beliefs and desires to actions. Work in this area builds largely on research in the philosophy of mind. The logical approaches presented in Section 2 focus on this trend.

Models of the strategic structure of the system: The second main strand of research focuses on the strategic structure of the environment: what agents can accomplish in the environment, either together or alone. Work in this area builds on models of effectivity from the game theory community, and the models underpinning such logics are closely related to formal games. In Section 3 we present logics that deal with this trend.

1.1 A Logical Toolkit

In this section, we very briefly touch upon the basic logics to reason about knowledge, about time, and about action. Let i be a variable over a set of agents $Ag = \{1, \dots, m\}$. For reasoning about knowledge or beliefs of agents (we will not dwell here on the distinction between the two), one usually adds, for every agent i , an operator K_i to the language, where $K_i\varphi$ then denotes that agent i knows φ . In the best-known epistemic logics [31, 72], we see the axioms as given in Figure 16.1.

This inference system is often referred to as $S5_m$ if there are m agents. Axiom *Kn1* is obvious, *Kn2* denotes that an agent can perform deductions upon what it knows, and *Kn3* is often referred to as veridicality: what one knows is true. If *Kn3* is replaced by the weaker constraint *Kn3'*, saying $\neg K_i\perp$, the result is a

logic for belief (where “ i believes ϕ ” is usually written as $B_i\phi$). Finally, $Kn4$ and $Kn5$ denote positive and negative introspection, respectively: they indicate that an agent knows what it knows, and knows what it is ignorant of. Modus Ponens (MP) is a standard logical rule, and Necessitation (Nec) guarantees that it is derivable that agents know all tautologies.

Moving on to the semantics of such a logic, models for epistemic logic are tuples $M = \langle S, R_{i \in Ag}, V \rangle$ (also known as *Kripke models*), where S is a set of states, $R_i \subseteq S \times S$ is a binary relation for each agent i , and $V : At \rightarrow 2^S$ gives for each atom p the states $V(p)$ where p is true. Truth of ϕ in a model M with state s , written as $M, s \models \phi$, is standard for the classical connectives (cf. Figure 16.2, left), and the clause $M, s \models K_i\phi$ means that for all t with $R_i s t$, $M, t \models \phi$ holds. In other words, in state s agent i knows ϕ iff ϕ is true in all states t that look similar to s for i . K_i is called the *necessity operator* for R_i . $M \models \phi$ means that for all states $s \in S$, $M, s \models \phi$. So the states describe some atomic facts p about some situation, and $R_i s t$ means that for agent i , the states s and t look the same, or, given its information, are indistinguishable. Let $S5_m$ be all models in which each R_i is an equivalence relation. Let $S5_m \models \phi$ mean that in all models $M \in S5_m$, we have $M \models \phi$. The system $S5_m$ is complete for the validities in $S5_m$, i.e., for all ϕ we have $S5_m \vdash \phi$ iff $S5_m \models \phi$.

Whereas in epistemic logics the binary relation on the set of states represents the agents’ ignorance, in temporal logics it represents the *flow of time*. In its most simple appearance, time has a beginning, and advances linearly and discretely into an infinite future: this is linear-time temporal logic (LTL, [84]). So a simple model for time is obtained by taking as the set of states the natural numbers \mathbb{N} , and for the accessibility relation “the successor” of, i.e., $R = \{(n, n+1)\}$, and V , the valuation, can be used to specify specific properties in states. In the language, we then would typically see operators for the “next state,” (\bigcirc), for “all states in the future” (\Box), and for “some time in the future” (\Diamond). The truth conditions for those operators, together with an axiom system for them, are given in Figure 16.2. Note that \Box is the reflexive transitive closure of R , and \Diamond is its dual: $\Diamond\phi = \neg\Box\neg\phi$.

To give a simple example, suppose that atom p is true in exactly the prime numbers, e is true in all even numbers, and o in all odd numbers. In the natural numbers, in state 0 we then have $\Box\Diamond p$ (there are infinitely many primes), $\Diamond(p \wedge e \wedge \bigcirc \Box \neg(p \wedge e))$ (there is a number that is even and prime, and for which all greater numbers are not even and prime), and $\Box(\Diamond(e \wedge \bigcirc \Diamond o))$ (for every number, one can find a number at least as big that is even and for which one can find a bigger number that is odd).

Often, one wants a more expressive language, adding, for instance, an operator for until:

$$M, n \models \phi \mathcal{U} \psi \text{ iff } \exists m \geq n (M, m \models \psi \ \& \ \forall n \geq k \geq m \ M, k \models \phi)$$

Truth Conditions		LTL Axioms	
$M, n \models p$	iff $n \in V(p)$	<i>LTL1</i>	φ (φ a prop. taut.)
$M, n \models \neg\varphi$	iff not $M, n \models \varphi$	<i>LTL2</i>	$\bigcirc(\varphi \rightarrow \psi) \rightarrow (\bigcirc\varphi \rightarrow \bigcirc\psi)$
$M, n \models \varphi \wedge \psi$	iff $M, n \models \varphi$ and $M, n \models \psi$	<i>LTL3</i>	$\neg\bigcirc\varphi \leftrightarrow \bigcirc\neg\varphi$
$M, n \models \bigcirc\varphi$	iff $M, n+1 \models \varphi$	<i>LTL4</i>	$\Box\varphi \rightarrow (\varphi \wedge \bigcirc\Box\varphi)$
$M, n \models \Box\varphi$	iff $\forall m \geq n, M, m \models \varphi$	Rules of Inference	
$M, n \models \Diamond\varphi$	iff $\exists m \geq n, M, m \models \varphi$	<i>MP</i>	$\vdash \varphi, \vdash (\varphi \rightarrow \psi) \Rightarrow \vdash \psi$
		<i>Nec</i>	$\vdash \varphi \Rightarrow \vdash \bigcirc\varphi$
		<i>Ind</i>	$\vdash \varphi \rightarrow \psi, \vdash \varphi \rightarrow \bigcirc\varphi \Rightarrow \vdash \varphi \rightarrow \Box\psi$

Figure 16.2: Semantics and axioms of linear temporal logic.

A rational agent deliberates about its choices, and to represent those, branching-time seems a more appropriate framework than linear-time. To understand branching-time operators, though, an understanding of linear-time operators is still of benefit. Computational tree logic (CTL) (see [29]) is a branching-time logic that uses pairs of operators; the first quantifies over paths, and the second is an LTL operator over those paths. Let us demonstrate this by mentioning some properties that are true in the root ρ of the branching-time model M of Figure 16.3. Note that on the highlighted path, in ρ , the formula $\Box\neg q$ is true. Hence, on the branching-model M, ρ , we have $E\Box\neg q$, saying that in ρ , there exists a path through it, on which q is always false. $A\Diamond\varphi$ means that on every path starting in ρ , there is some future point where φ is true. So, in ρ , $A\Diamond\neg p$ holds. Likewise, $E p \mathcal{U} q$ is true in ρ because there is a path (the path “up,” for example) in which $p \mathcal{U} q$ is true. We leave it to the reader to check that in ρ , we have $E\Diamond(p \wedge A\Box\neg p)$. In CTL*, the requirement that path and tense operators need to occur together is dropped: CTL* formulas true in ρ are, for instance, $A(\Box\Diamond p \wedge \Box\Diamond\neg p)$ (all paths have infinitely many p states and infinitely many $\neg p$ states), and AEp (for all paths, there is a path such that p).

Rather than giving an axiom system for CTL here, we now describe frameworks where change is not imposed by nature (i.e., by passing of time), but where we can be more explicit about *how* change is brought about. By definition, an agent is supposed to act, so rather than thinking of the flow of time as the main driver for change, transitions between states can be labeled with actions, or, more generally, a pair (i, α) , where i is an agent and α an action. The formalism discussed below is based on dynamic logic [41, 42], which is again a modal logic. On page 772 we will see how temporal operators can be defined in terms of dynamic ones.

Actions in the set Ac are either atomic actions (a, b, \dots) or composed (α, β, \dots) by means of testing of formulas $(\varphi?)$, sequencing $(\alpha; \beta)$, conditioning $(\text{if } \varphi \text{ then}$

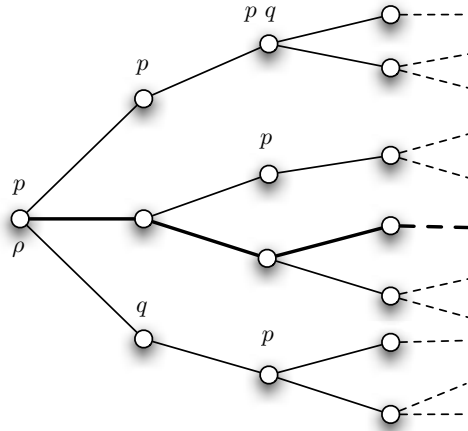


Figure 16.3: A branching-time model. One branch is highlighted.

α else β), and repetition (while ϕ do α). The informal meaning of such constructs is as follows:

$\phi?$	denotes a “test action” ϕ
$\alpha; \beta$	denotes α followed by β
if ϕ then α else β	if ϕ holds, action α is executed, else β
while ϕ do α	as long as ϕ is true, α is executed

Here, the test must be interpreted as a test by the system; it is not a so-called “knowledge-producing action” (like observations or communication) that can be used by the agent to acquire knowledge.

These actions α can then be used to build new formulas to express the possible *result* of the execution of α by agent i (the formula $[do_i(\alpha)]\phi$ denotes that ϕ is a result of i ’s execution of α), the *opportunity* for i to perform α ($\langle do_i(\alpha) \rangle \top$), and i ’s *capability* of performing the action α ($\mathbf{A}_i \alpha$). The formula $\langle do_i(\alpha) \rangle \phi$ is shorthand for $\neg[do_i(\alpha)]\neg\phi$, thus expressing that one possible result of performance of α by i implies ϕ .

In the Kripke semantics, we then assume relations R_a for individual actions, where the relations for compositions are then recursively defined: for instance, $R_{\alpha;\beta}st$ iff for some state u , $R_\alpha su$ and $R_\beta ut$. Indeed, $[do_i(\alpha)]$ is then the necessity operator for R_α . Having epistemic and dynamic operators, one has already a rich framework to reason about an agent’s knowledge about doing actions. For instance, a property like *perfect recall*

$$\mathbf{K}_i[do_i(\alpha)]\phi \rightarrow [do_i(\alpha)]\mathbf{K}_i\phi$$

which semantically implies some grid structure on the set of states: If $R_{\alpha st}$ and R_{itu} , then for some v , we also have R_{isv} and $R_{\alpha vu}$. For temporal epistemic logic, perfect recall is captured in the axiom $K_i \bigcirc \phi \rightarrow \bigcirc K_i \phi$, while its converse, no learning, is $\bigcirc K_i \phi \rightarrow K_i \bigcirc \phi$. It is exactly this kind of interaction properties that can make a multiagent logic complex, both conceptually and computationally.

In studying the way that actions and knowledge interact, Robert Moore argued that one needs to identify two main issues. The first is that some actions *produce knowledge*, and therefore their effects must be formulated in terms of the epistemic states of participants. The second is that of *knowledge preconditions*: what an agent needs to know in order to be able to perform an action. A simple example is that in order to unlock a safe, one must know the combination for the lock. Using these ideas, Moore formalized a notion of *ability*. He suggested that in order for an agent to be able to achieve some state of affairs ϕ , the agent must either:

- know the identity of an action α (i.e., have an “executable description” of an action α) such that after α is performed, ϕ holds; or else
- know the identity of an action α such that after α is performed, the agent will know the identity of an action α' such that after α' is performed, ϕ holds.

The point about “knowing the identity” of an action is that in order for me to be able to become rich, it is not sufficient for me simply to know that *there exists some action* I could perform that would make me rich. I must either know what that action is (the first clause above), or else be able to perform some action that would furnish me with the information about which action to perform in order to make myself rich. This subtle notion of knowing an action is rather important, and it is related to the distinction between knowledge *de re* (which involves knowing the identity of a thing) and *de dicto* (which involves knowing that something exists) [31, p. 101]. In the example of the safe, most people would have knowledge *de dicto* to open the safe, but only a few would have knowledge *de re*. We will see later, when we review more recent work on temporal logics of ability, that this distinction also plays an important role there.

It is often the case that actions are *ontic*: they bring about a change in the world, like assigning a value to a variable, moving a block, or opening a door. However, dynamic epistemic logic (DEL) [113] studies actions that bring about *mental change*: change of knowledge in particular. So in DEL, the actions themselves are epistemic. A typical example is announcing ϕ in a group of agents: $[\phi]\psi$ would then mean that after announcement of ϕ , it holds that ψ . Surprisingly enough, the formula $[\phi]K_i\phi$ (after the announcement that ϕ , agent i knows that ϕ , is not a validity, a counterexample being the infamous *Moore sentences* [74] $\phi = (\neg K_i p \wedge p)$: “although i does not know it, p holds”).

Let us make one final remark in this section. We already indicated that things become interesting and challenging when one combines several notions into one framework (like knowledge and action, or knowledge and time). In fact it already becomes interesting if we stick to one notion, and take the aspect of having a multiagent system seriously. For instance, interesting group notions of knowledge in $S5_m$ gives rise to are

- $E\varphi$ (“everybody knows φ ”, i.e., $K_1\varphi \wedge \dots \wedge K_m\varphi$),
- $D\varphi$ (“it is distributed knowledge that φ ”, i.e., if you would pool all the knowledge of the agents together, φ would follow from it, like in $(K_i(\varphi_1 \rightarrow \varphi_2) \wedge K_j\varphi_1) \rightarrow D\varphi_2$), and
- $C\varphi$ (“it is common knowledge that φ ”: this is axiomatized such that it resembles the infinite conjunction $E\varphi \wedge EE\varphi \wedge EEE\varphi \wedge \dots$).

In DEL for instance, this gives rise to the question of which formulas are *successful*, i.e., formulas for which $[\varphi]C\varphi$ (after φ is announced, it is public knowledge) is valid [114]. Different questions arise when taking the multiagent aspect seriously in the context of actions. What happens with the current state if all agents take some action? How to “compute” the result of those concurrent choices? This latter question will in particular be addressed in Section 3. First, we focus on logics that amplify aspects of the mental state of agents. The next two sections heavily borrow from [111].

2 Representing Cognitive States

In attempting to understand the behavior of agents in the everyday world, we frequently make use of *folk psychology*:

Many philosophers and cognitive scientists claim that our everyday or “folk” understanding of mental states constitutes a theory of mind. That theory is widely called “folk psychology” (sometimes “common sense” psychology). The terms in which folk psychology is couched are the familiar ones of “belief,” “desire,” “hunger,” “pain,” and so forth. According to many theorists, folk psychology plays a central role in our capacity to predict and explain the behavior of ourselves and others. However, the nature and status of folk psychology remains controversial. [126]

The philosopher Dennett coined the phrase *intentional system* to refer to an entity that is best understood in terms of folk psychology notions such as beliefs, desires, and the like [25]. The intentional stance is essentially nothing more than an

abstraction tool. If we accept the usefulness of the intentional stance for characterizing the properties of rational agents, then the next step in developing a formal theory of such agents is to identify the components of an agent's state. There are many possible mental states that we might choose to characterize an agent: beliefs, goals, desires, intentions, commitments, fears, and hopes are just a few. We can identify several important categories of such attitudes, for example:

Information attitudes: those attitudes an agent has toward information about its environment. The most obvious members of this category are knowledge and belief.

Pro-attitudes: those attitudes an agent has that tend to lead it to perform actions. The most obvious members of this category are goals, desires, and intentions.

Moreover, there is also a *social state* when modeling agents which includes:

Normative attitudes: including obligations, permissions, and authorization.

We will not say much about normative attitudes in this chapter other than giving some pointers for further reading in Section 4.

Much of the literature on developing formal theories of agency has been taken up with the relative merits of choosing one attitude over another, and investigating the possible relationships between these attitudes.

2.1 Intention Logic

One of the best-known, and most sophisticated attempts to show how the various components of an agent's cognitive makeup could be combined to form a logic of rational agency is that of Cohen and Levesque [22]. The logic has proved to be so useful for specifying and reasoning about the properties of agents that it has been used in an analysis of conflict and cooperation in multiagent dialogue [36], [35], as well as in several studies in the theoretical foundations of cooperative problem solving [61, 62, 65]. This section will focus on the use of the logic in developing a theory of intention. The first step is to lay out the criteria that a theory of intention must satisfy.

When building intelligent agents – particularly agents that must interact with humans – it is important that a *rational balance* is achieved between the beliefs, goals, and intentions of the agents.

For example, the following are desirable properties of intention: An autonomous agent should act on its intentions, not in spite of them;

adopt intentions it believes are feasible and forego those believed to be infeasible; keep (or commit to) intentions, but not forever; discharge those intentions believed to have been satisfied; alter intentions when relevant beliefs change; and adopt subsidiary intentions during plan formation. [22, p. 214]

Following [15, 16], Cohen and Levesque identify seven specific properties that must be satisfied by a reasonable theory of intention:

1. Intentions pose problems for agents, who need to determine ways of achieving them.
2. Intentions provide a “filter” for adopting other intentions, which must not conflict.
3. Agents track the success of their intentions, and are inclined to try again if their attempts fail.
4. Agents believe their intentions are possible.
5. Agents do not believe they will not bring about their intentions.
6. Under certain circumstances, agents believe they will bring about their intentions.
7. Agents need not intend all the expected side effects of their intentions.

Given these criteria, Cohen and Levesque adopt a two-tiered approach to the problem of formalizing a theory of intention. First, they construct the logic of rational agency, “being careful to sort out the relationships among the basic modal operators” [22, p. 221]. On top of this framework, they introduce a number of derived constructs, which constitute a “partial theory of rational action” [22, p. 221]; intention is one of these constructs.

Syntactically, the logic of rational agency is a many-sorted, first-order, multi-modal logic with equality, containing four primary modalities (see Table 16.1). The semantics of Bel and Goal are given via possible worlds, in the usual way: each agent is assigned a belief accessibility relation and a goal accessibility relation. The belief accessibility relation is Euclidean, transitive, and serial, giving a belief logic of KD45. The goal relation is serial, giving a conative logic KD. It is assumed that each agent’s goal relation is a subset of its belief relation, implying that an agent will not have a goal of something it believes will not happen. A world in this formalism is a discrete sequence of events, stretching infinitely into the past and future. The system is only defined semantically, and Cohen and

Operator	Meaning
(Bel $i \ \varphi$)	agent i believes φ
(Goal $i \ \varphi$)	agent i has goal of φ
(Happens α)	action α will happen next
(Done α)	action α has just happened

Table 16.1: Atomic modalities in Cohen and Levesque's logic.

Levesque derive a number of properties from that. In the semantics, a number of assumptions are implicit, and one might vary on them. For instance, there is a fixed domain assumption, giving us properties such as

$$\forall x(\text{Bel } i \ \varphi(x)) \rightarrow (\text{Bel } i \forall x \varphi(x)) \quad (16.1)$$

The philosophically-oriented reader will recognize a *Barcan formula* in (16.1), which in this case expresses that the agent is aware of all the elements in the domain. Also, agents “know what time it is”; we immediately obtain from the semantics the validity of formulas like $2 : 30\text{PM}/3/6/85 \rightarrow \text{Bel } i \ 2 : 30\text{PM}/3/6/85$.

Intention logic has two basic operators to refer to actions, Happens and Done. The standard future time operators of temporal logic, “ \square ” (always), and “ \diamond ” (sometime), can be defined as abbreviations, along with a “strict” sometime operator, Later:

$$\begin{aligned} \diamond \alpha &\triangleq \exists x \cdot (\text{Happens } x; \alpha?) \\ \square \alpha &\triangleq \neg \diamond \neg \alpha \\ (\text{Later } p) &\triangleq \neg p \wedge \diamond p \end{aligned}$$

A temporal precedence operator, (Before $p \ q$), can also be derived, and holds if p holds before q . An important assumption is that *all* goals are eventually dropped:

$$\diamond \neg (\text{Goal } x \ (\text{Later } p))$$

The first major derived construct is a *persistent* goal.

$$\begin{aligned} (\text{P-Goal } i \ p) &\triangleq \begin{aligned} &(\text{Goal } i \ (\text{Later } p)) && \wedge \\ &(\text{Bel } i \ \neg p) && \wedge \\ &\left[\begin{array}{l} \text{Before} \\ ((\text{Bel } i \ p) \vee (\text{Bel } i \ \square \neg p)) \\ \neg(\text{Goal } i \ (\text{Later } p)) \end{array} \right] \end{aligned} \end{aligned}$$

So, an agent has a persistent goal of p if:

1. It has a goal that p eventually becomes true, and believes that p is not currently true.
2. Before it drops the goal, one of the following conditions must hold:
 - (a) the agent believes the goal has been satisfied;
 - (b) the agent believes the goal will never be satisfied.

It is a small step from persistent goals to a first definition of intention, as in “intending to act.” Note that “intending that something becomes true” is similar, but requires a slightly different definition (see [22]). An agent i intends to perform action α if it has a persistent goal to have brought about a state where it had just believed it was about to perform α , and then did α .

$$(\text{Intend } i \alpha) \triangleq (\text{P-Goal } i \text{ [Done } i (\text{Bel } i (\text{Happens } \alpha))?; \alpha]})$$

Cohen and Levesque go on to show how such a definition meets many of Bratman’s criteria [15] for a theory of intention (outlined above). In particular, by basing the definition of intention on the notion of a persistent goal, Cohen and Levesque are able to avoid overcommitment or undercommitment. An agent will only drop an intention if it believes that the intention has either been achieved, or is unachievable.

A critique of Cohen and Levesque’s theory of intention is presented in [103]; space restrictions prevent a discussion here.

2.2 BDI Logic

One of the best-known (and most widely misunderstood) approaches to reasoning about rational agents is the *belief-desire-intention* (BDI) model [17]. The BDI model gets its name from the fact that it recognizes the primacy of beliefs, desires, and intentions in rational action. The BDI model is particularly interesting because it combines three distinct components:

- *A philosophical foundation*

The BDI model is based on a widely respected theory of rational action in humans, developed by the philosopher Michael Bratman [15].

- *A software architecture*

The BDI model of agency does not prescribe a specific implementation. The model may be realized in many different ways, and indeed a number of

different implementations of it have been developed. However, the fact that the BDI model *has* been implemented successfully is a significant point in its favor. Moreover, the BDI model has been used to build a number of significant real-world applications, including such demanding problems as fault diagnosis on the space shuttle.

- *A logical formalization*

The third component of the BDI model is a family of logics. These logics capture the key aspects of the BDI model as a set of logical axioms. There are many candidates for a formal theory of rational agency, but BDI logics in various forms have proved to be among the most useful, longest-lived, and most widely accepted.

Intuitively, an agent's *beliefs* correspond to information the agent has about the world. These beliefs may be incomplete or incorrect. An agent's *desires* represent states of affairs that the agent would, in an ideal world, wish to be brought about. (Implemented BDI agents require that desires be *consistent* with one another, although *human* desires often fail in this respect.) Finally, an agent's *intentions* represent desires that it has *committed* to achieving. The intuition is that an agent will not, in general, be able to achieve *all* its desires, even if these desires *are* consistent. Ultimately, an agent must therefore fix upon some subset of its desires and commit resources to achieving them. These chosen desires, to which the agent has some commitment, are intentions [22]. The BDI theory of human rational action was originally developed by Michael Bratman [15]. It is a theory of *practical reasoning* – the process of reasoning that we all go through in our everyday lives, deciding moment by moment which action to perform next.

The BDI model has been implemented several times. Originally, it was realized in IRMA, the intelligent resource-bounded machine architecture [17]. IRMA was intended as a more or less direct realization of Bratman's theory of practical reasoning. However, the best-known implementation is the procedural reasoning system (PRS) [37] and its many descendants [26, 33, 55, 87]. In the PRS, an agent has data structures that explicitly correspond to beliefs, desires, and intentions. A PRS agent's beliefs are directly represented in the form of PROLOG-like facts [21, p. 3]. Desires and intentions in PRS are realized through the use of a *plan library*.² A plan library, as its name suggests, is a collection of plans. Each plan is a recipe that can be used by the agent to achieve some particular state of affairs. A plan in the PRS is characterized by a *body* and an *invocation condition*. The body of a plan is a course of action that can be used by the agent to achieve some particular state of affairs. The invocation condition of a plan defines the circumstances under

²In this description of the PRS, we have modified the original terminology somewhat to be more in line with contemporary usage; we have also simplified the control cycle of the PRS slightly.

which the agent should “consider” the plan. Control in the PRS proceeds by the agent continually updating its internal beliefs, and then looking to see which plans have invocation conditions that correspond to these beliefs. The set of plans made active in this way corresponds to the *desires* of the agent. Each desire defines a possible course of action that the agent may follow. On each control cycle, the PRS picks one of these desires, and pushes it onto an execution stack for subsequent execution. The execution stack contains desires that have been chosen by the agent, and thus corresponds to the agent’s *intentions*.

The third and final aspect of the BDI model is the logical component, which gives us a family of tools that allow us to reason about BDI agents. There have been several versions of BDI logic, starting in 1991 and culminating in Rao and Georgeff’s 1998 paper on systems of BDI logics [88, 90, 91, 92, 93, 94, 95]; a book-length survey was published as [121]. We focus on [121].

Syntactically, BDI logics are essentially branching-time logics (CTL or CTL*, depending on which version one is reading about), enhanced with additional modal operators *Bel*, *Des*, and *Intend*, for capturing the beliefs, desires, and intentions of agents, respectively. The BDI modalities are indexed with agents, so for example the following is a legitimate formula of BDI logic:

$$(\text{Bel } i (\text{Intend } j A \Diamond p)) \rightarrow (\text{Bel } i (\text{Des } j A \Diamond p))$$

This formula says that if *i* believes that *j* intends that *p* is inevitably true eventually, then *i* believes that *j* desires *p* is inevitable. Although they share much in common with Cohen-Levesque’s intention logics, the first and most obvious distinction between BDI logics and the Cohen-Levesque approach is the explicit starting point of CTL-like branching-time logics. However, the differences are actually much more fundamental than this. The semantics that Rao and Georgeff give to BDI modalities in their logics are based on the conventional apparatus of Kripke structures and possible worlds. However, rather than assuming that worlds are instantaneous states of the world, or even that they are linear sequences of states, it is assumed instead that worlds are themselves branching temporal structures: thus each world can be viewed as a Kripke structure for a CTL-like logic. While this tends to rather complicate the semantic machinery of the logic, it makes it possible to define an interesting array of semantic properties, as we shall see below.

Before proceeding, we summarize the key semantic structures in the logic. Instantaneous states of the world are modeled by *time points*, given by a set *T*; the set of all possible evolutions of the system being modeled is given by a binary relation $R \subseteq T \times T$. A *world* (over *T* and *R*) is then a pair $\langle T', R' \rangle$, where $T' \subseteq T$ is a non-empty set of time points, and $R' \subseteq R$ is a branching-time structure on T' . Let *W* be the set of all worlds over *T*. A pair $\langle w, t \rangle$, where $w = \langle T_w, R_w \rangle \in W$

and $t \in T_w$, is known as a *situation*. If $w \in W$, then the set of all situations in w is denoted by S_w . We have belief accessibility relations B , D , and I , modeled as functions that assign to every agent a relation over situations. Thus, for example:

$$B : Ag \rightarrow 2^{W \times T \times W}$$

We write $B_t^w(i)$ to denote the set of worlds accessible to agent i from situation $\langle w, t \rangle$: $B_t^w(i) = \{w' \mid \langle w, t, w' \rangle \in B(i)\}$. We define D_t^w and I_t^w in the obvious way. The semantics of belief, desire, and intention modalities are then given in the conventional manner:

- $\langle w, t \rangle \models (\text{Bel } i \ \phi)$ iff $\langle w', t \rangle \models \phi$ for all $w' \in B_t^w(i)$
- $\langle w, t \rangle \models (\text{Des } i \ \phi)$ iff $\langle w', t \rangle \models \phi$ for all $w' \in D_t^w(i)$
- $\langle w, t \rangle \models (\text{Intend } i \ \phi)$ iff $\langle w', t \rangle \models \phi$ for all $w' \in I_t^w(i)$

The primary focus of Rao and Georgeff's early work was to explore the possible interrelationships between beliefs, desires, and intentions from the perspective of semantic characterization. In order to do this, they defined a number of possible interrelationships between an agent's belief, desire, and intention accessibility relations. The most obvious relationships that can exist are whether one relation is a subset of another: for example, if $D_t^w(i) \subseteq I_t^w(i)$ for all i, w, t , then we would have as an interaction axiom $(\text{Intend } i \ \phi) \rightarrow (\text{Des } i \ \phi)$. However, the fact that worlds themselves have structure in BDI logic also allows us to combine such properties with relations on the *structure* of worlds themselves. The most obvious structural relationship that can exist between two worlds – and the most important for our purposes — is that of one world being a *subworld* of another. Intuitively, a world w is said to be a subworld of world w' if w has the same structure as w' but has fewer paths *and is otherwise identical*. Formally, if w, w' are worlds, then w is a subworld of w' (written $w \sqsubseteq w'$) iff $\text{paths}(w) \subseteq \text{paths}(w')$ but w, w' agree on the interpretation of predicates and constants in common time points.

The first property we consider is the *structural subset* relationship between accessibility relations. We say that accessibility relation R is a structural subset of accessibility relation \bar{R} if for every R -accessible world w , there is an \bar{R} -accessible world w' such that w is a subworld of w' . Formally, if R and \bar{R} are two accessibility relations, then we write $R \subseteq_{\text{sub}} \bar{R}$ to indicate that if $w' \in R_t^w(i)$, then there exists some $w'' \in \bar{R}_t^w(i)$ such that $w' \sqsubseteq w''$. If $R \subseteq_{\text{sub}} \bar{R}$, then we say R is a *structural subset* of \bar{R} .

We write $\bar{R} \subseteq_{\text{sup}} R$ to indicate that if $w' \in \bar{R}_t^w(i)$, then there exists some $w'' \in R_t^w(i)$ such that $w'' \sqsubseteq w'$. If $\bar{R} \subseteq_{\text{sup}} R$, then we say R is a *structural superset* of \bar{R} . In other words, if R is a structural superset of \bar{R} , then for every R -accessible world w , there is an \bar{R} -accessible world w' such that w' is a subworld of w .

Name	Semantic Condition	Corresponding Formula Schema
BDI-S1	$B \subseteq_{sup} D \subseteq_{sup} I$	$(\text{Intend } i \ E(\varphi)) \rightarrow (\text{Des } i \ E(\varphi)) \rightarrow (\text{Bel } i \ E(\varphi))$
BDI-S2	$B \subseteq_{sub} D \subseteq_{sub} I$	$(\text{Intend } i \ A(\varphi)) \rightarrow (\text{Des } i \ A(\varphi)) \rightarrow (\text{Bel } i \ A(\varphi))$
BDI-S3	$B \subseteq D \subseteq I$	$(\text{Intend } i \ \varphi) \rightarrow (\text{Des } i \ \varphi) \rightarrow (\text{Bel } i \ \varphi)$
BDI-R1	$I \subseteq_{sup} D \subseteq_{sup} B$	$(\text{Bel } i \ E(\varphi)) \rightarrow (\text{Des } i \ E(\varphi)) \rightarrow (\text{Intend } i \ E(\varphi))$
BDI-R2	$I \subseteq_{sub} D \subseteq_{sub} B$	$(\text{Bel } i \ A(\varphi)) \rightarrow (\text{Des } i \ A(\varphi)) \rightarrow (\text{Intend } i \ A(\varphi))$
BDI-R3	$I \subseteq D \subseteq B$	$(\text{Bel } i \ \varphi) \rightarrow (\text{Des } i \ \varphi) \rightarrow (\text{Intend } i \ \varphi)$
BDI-W1	$B \cap_{sup} D \neq \emptyset$	$(\text{Bel } i \ A(\varphi)) \rightarrow \neg(\text{Des } i \ \neg A(\varphi))$
	$D \cap_{sup} I \neq \emptyset$	$(\text{Des } i \ A(\varphi)) \rightarrow \neg(\text{Intend } i \ \neg A(\varphi))$
	$B \cap_{sup} I \neq \emptyset$	$(\text{Bel } i \ A(\varphi)) \rightarrow \neg(\text{Intend } i \ \neg A(\varphi))$
BDI-W2	$B \cap_{sub} D \neq \emptyset$	$(\text{Bel } i \ E(\varphi)) \rightarrow \neg(\text{Des } i \ \neg E(\varphi))$
	$D \cap_{sub} I \neq \emptyset$	$(\text{Des } i \ E(\varphi)) \rightarrow \neg(\text{Intend } i \ \neg E(\varphi))$
	$B \cap_{sub} I \neq \emptyset$	$(\text{Bel } i \ E(\varphi)) \rightarrow \neg(\text{Intend } i \ \neg E(\varphi))$
BDI-W3	$B \cap D \neq \emptyset$	$(\text{Bel } i \ \varphi) \rightarrow \neg(\text{Des } i \ \neg \varphi)$
	$D \cap I \neq \emptyset$	$(\text{Des } i \ \varphi) \rightarrow \neg(\text{Intend } i \ \neg \varphi)$
	$B \cap I \neq \emptyset$	$(\text{Bel } i \ \varphi) \rightarrow \neg(\text{Intend } i \ \neg \varphi)$

Table 16.2: Systems of BDI logic. (Source: [90, p. 321].) In the first six rows, the corresponding formulas of type $A \rightarrow B \rightarrow C$ are shorthand for $(A \rightarrow B) \wedge (B \rightarrow C)$.

Finally, we can also consider whether the *intersection* of accessibility relations is empty or not. For example, if $B_t^w(i) \cap I_t^w(i) \neq \emptyset$, for all i, w, t , then we get the following interaction axiom:

$$(\text{Intend } i \ \varphi) \rightarrow \neg(\text{Bel } i \ \neg \varphi)$$

This axiom expresses an *intermodal consistency* property. Just as we can undertake a more fine-grained analysis of the basic interactions among beliefs, desires, and intentions by considering the structure of worlds, so we are also able to undertake a more fine-grained characterization of intermodal consistency properties by taking into account the structure of worlds. We write $R_t^w(i) \cap_{sup} \bar{R}_t^w(i)$ to denote the set of worlds $w' \in \bar{R}_t^w(i)$ for which there exists some world $w'' \in R_t^w(i)$ such that $w' \sqsubseteq w''$. We can then define \cap_{sub} in the obvious way.

Putting all these relations together, we can define a range of BDI logical systems. The most obvious possible systems, and the semantic properties that they correspond to, are summarized in Table 16.2.

2.3 Discussion

Undoubtedly, formalizing the informational and motivational attitudes in a context with evolving time, or where agents can do actions, has greatly helped to improve our understanding of complex systems. At the same time, admittedly, there are many weaknesses and open problems with such approaches.

To give one example of how a formalization can help us to become more clear about the interrelationship between the notions defined here, recall that Rao and Georgeff assume the notion of *belief-goal compatibility*, saying

$$\mathbf{Goal}_i\varphi \rightarrow \mathbf{B}_i\varphi$$

for formulas φ that refer to the future.

Cohen and Levesque, however, put a lot of emphasis on their notion of *realizability*, stating exactly the opposite:

$$\mathbf{B}_i\varphi \rightarrow \mathbf{Goal}_i\varphi$$

By analyzing the framework of Cohen and Levesque more closely, it appears that they have a much weaker property in mind, which is

$$\mathbf{Goal}_i\varphi \rightarrow \neg\mathbf{B}_i\neg\varphi$$

To mention just one aspect in which the approach mentioned here is still far from completed, we recall that the three frameworks allow one to reason about many agents, but are in essence still one-agent systems. Although notions such as distributed and common knowledge are well-understood epistemic notions in multi-agent systems, their motivational analogues seem to be much harder and are yet only partially understood (see Cohen and Levesque's [23], and Tambe's [105] or Dunin-Kępicz and Verbrugge's [28] on teamwork).

2.4 Cognitive Agents in Practice

Broadly speaking, logic plays a role in three aspects of software development: as a *specification language*; as a *programming language*; and as a *verification language*. In the sections that follow, we will discuss the possible use of logics of rational agency in these three processes.

2.4.1 Specification Language

The software development process begins by establishing the client's requirements. When this process is complete, a *specification* is developed, which sets out the functionality of the new system. Temporal and dynamic logics have found

wide applicability in the specification of systems. An obvious question is therefore whether logics of rational agency might be used as specification languages.

A specification expressed in such a logic would be a formula ϕ . The idea is that such a specification would express the desirable behavior of a system. To see how this might work, consider the following formula of BDI logic (in fact from [121]), intended to form part of a specification of a process control system.

$$(\text{Bel } i \text{ Open}(\text{valve32})) \rightarrow (\text{Intend } i (\text{Bel } j \text{ Open}(\text{valve32})))$$

This formula says that if i believes valve 32 is open, then i should intend that j believes valve 32 is open. A rational agent i with such an intention can select a speech act to perform in order to inform j of this state of affairs. It should be intuitively clear how a system specification might be constructed using such formulae, to define the intended behavior of a system.

One of the main desirable features of a software specification language is that it should not dictate *how* a specification should be satisfied by an implementation. It should be clear that the specification above has exactly these properties. It does not dictate how agent i should go about making j aware that valve 32 is open. We simply expect i to behave as a rational agent given such an intention.

There are a number of problems with the use of such logics for specification. The most worrying of these is with respect to their semantics. As we set out in Section 1, the semantics for the modal operators (for beliefs, desires, and intentions) are given in the normal modal logic tradition of possible worlds [19]. There are several advantages to the possible worlds model: it is well studied and well understood, and the associated mathematics of correspondence theory is extremely elegant. These attractive features make possible worlds the semantics of choice for almost every researcher in formal agent theory. However, there are also a number of serious drawbacks to possible worlds semantics. First, possible worlds semantics imply that agents

- are logically perfect reasoners (in that their deductive capabilities are sound and complete, this follows for instance from the axiom $Kn2$ and the rule Nec that we gave in Figure 16.1 for knowledge, and, moreover, this axiom and inference rule are part of *any* modal axiomatization of agent's attitudes) and
- have infinite resources available for reasoning (see axioms $Kn1$ and $Kn2$ and rule Nec again).

No real agent, artificial or otherwise, has these properties.

Second, possible worlds semantics are generally *ungrounded*. That is, there is usually no precise relationship between the abstract accessibility relations that are

used to characterize an agent's state, and any concrete computational model. As we shall see in later sections, this makes it difficult to go from a formal specification of a system in terms of beliefs, desires, and so on, to a concrete computational system. Similarly, given a concrete computational system, there is generally no way to determine what the beliefs, desires, and intentions of that system are. If temporal modal logics of rational agency are to be taken seriously as *specification* languages, then this is a significant problem.

2.4.2 Implementation

Once given a specification, we must implement a system that is correct with respect to this specification. The next issue we consider is this move from abstract specification to concrete computational system. There are at least two possibilities for achieving this transformation that we consider here:

1. somehow directly execute or animate the abstract specification; or
2. somehow translate or compile the specification into a concrete computational form, using an automatic translation technique.

Directly Executing Agent Specifications. Suppose we are given a system specification, ϕ , which is expressed in some logical language L . One way of obtaining a concrete system from ϕ is to treat it as an *executable specification*, and *interpret* the specification directly in order to generate the agent's behavior. Interpreting an agent specification can be viewed as a kind of constructive proof of satisfiability, whereby we show that the specification ϕ is satisfiable by *building a model* (in the logical sense) for it. If models for the specification language L can be given a computational interpretation, then model building can be viewed as executing the specification. To make this discussion concrete, consider the Concurrent METATEM programming language [34]. In this language, agents are programmed by giving them a temporal logic specification of the behavior it is intended they should exhibit; this specification is directly executed to generate each agent's behavior. Models for the temporal logic in which Concurrent METATEM agents are specified are linear discrete sequences of states: executing a Concurrent METATEM agent specification is thus a process of constructing such a sequence of states. Since such state sequences can be viewed as the histories traced out by programs as they execute, the temporal logic upon which Concurrent METATEM is based has a computational interpretation; the actual execution algorithm is described in [13]. A somewhat related language is the IMPACT framework of Subrahmanian et al. [104]. IMPACT is a rich framework for programming agents, which draws upon and considerably extends some ideas from logic programming. Agents in

IMPACT are programmed by using rules that incorporate deontic modalities (permitted, forbidden, obliged [73]). These rules can be interpreted to determine the actions that an agent should perform at any given moment [104, p. 171].

Note that executing Concurrent METATEM agent specifications is possible primarily because the models upon which the Concurrent METATEM temporal logic is based are comparatively simple, with an obvious and intuitive computational interpretation. However, agent specification languages in general (e.g., the BDI formalisms of Rao and Georgeff [89]) are based on considerably more complex logics. In general, possible worlds semantics do not have a computational interpretation in the way that Concurrent METATEM semantics do. Hence it is not clear what “executing” a logic based on such semantics might mean.

In response to this issue, a number of researchers have attempted to develop executable agent specification languages with a simplified logical basis that has a computational interpretation. An example is Rao’s AGENTSPEAK(L) language, which although essentially a BDI system, has a simple computational semantics [87]. The 3APL project [44] is also an attempt to have an agent programming language with a well-defined semantics, based on transition systems. One advantage of having a thorough semantics is that it enables one to compare different agent programming languages, such as AGENTSPEAK(L) with 3APL [43] or AGENT0 with 3APL [45]. One complication in bridging the gap between the agent programming paradigm and the formal systems of Sections 2.1–2.2 is that the former usually takes goals to be procedural (a plan), whereas goals in the latter are declarative (a desired state). A programming language that tries to bridge the gap in this respect is the language GOAL [107].

GOLOG [64, 96] and its multiagent sibling CONGOLOG [63] represent another rich seam of work on logic-oriented approaches to programming rational agents. Essentially, GOLOG is a framework for executing a fragment of the situation calculus; the situation calculus is a well-known logical framework for reasoning about action [70]. Put crudely, writing a GOLOG program involves expressing a logical theory of what action an agent should perform, using the situation calculus; this theory, together with some background axioms, represents a logical expression of what it means for the agent to do the right action. Executing such a program reduces to constructively solving a deductive proof problem, broadly along the lines of showing that there is a sequence of actions representing an acceptable computation according to the theory [96, p. 121]; the witness to this proof will be a sequence of actions, which can then be executed.

Compiling Agent Specifications. An alternative to direct execution is *compilation*. In this scheme, we take our abstract specification, and transform it into a concrete computational model via some automatic synthesis process. The main

perceived advantages of compilation over direct execution are in run-time efficiency. Direct execution of an agent specification, as in Concurrent METATEM, above, typically involves manipulating a symbolic representation of the specification at run-time. This manipulation generally corresponds to reasoning of some form, which is computationally costly. Compilation approaches aim to reduce abstract symbolic specifications to a much simpler computational model, which requires no symbolic representation. The “reasoning” work is thus done off-line, at compile-time; execution of the compiled system can then be done with little or no run-time symbolic reasoning.

Compilation approaches usually depend upon the close relationship between models for temporal/modal logic (which are typically labeled graphs of some kind), and automata-like finite-state machines. For example, Pnueli and Rosner [85] synthesize reactive systems from branching temporal logic specifications. Similar techniques have also been used to develop concurrent system skeletons from temporal logic specifications. Perhaps the best-known example of this approach to agent development is the *situated automata* paradigm of Rosenschein and Kaelbling [98]. They use an epistemic logic to specify the perception component of intelligent agent systems. They then used a technique based on constructive proof to directly synthesize automata from these specifications [97].

The general approach of automatic synthesis, although theoretically appealing, is limited in a number of important respects. First, as the agent specification language becomes more expressive, then even off-line reasoning becomes too expensive to carry out. Second, the systems generated in this way are not capable of *learning* (i.e., they are not capable of adapting their “program” at run-time). Finally, as with direct execution approaches, agent specification frameworks tend to have no concrete computational interpretation, making such a synthesis impossible.

2.4.3 Verification

Once we have developed a concrete system, we need to show that this system is correct with respect to our original specification. This process is known as *verification*, and it is particularly important if we have introduced any informality into the development process. We can divide approaches to the verification of systems into two broad classes: (1) *axiomatic*; and (2) *semantic* (model checking).

Axiomatic approaches to program verification were the first to enter the mainstream of computer science, with the work of Hoare in the late 1960s [47]. Axiomatic verification requires that we can take our concrete program, and from this program systematically derive a logical theory that represents the behavior of the program. Call this the program theory. If the program theory is expressed in the same logical language as the original specification, then verification reduces to a

proof problem: show that the specification is a theorem of (equivalently, is a logical consequence of) the program theory. The development of a program theory is made feasible by *axiomatizing* the programming language in which the system is implemented. For example, Hoare logic gives us more or less an axiom for every statement type in a simple PASCAL-like language. Once given the axiomatization, the program theory can be derived from the program text in a systematic way.

Perhaps the most relevant work from mainstream computer science is the specification and verification of reactive systems, using temporal logic in the way pioneered by Pnueli, Manna, and colleagues [69]. The idea is that the computations of reactive systems are infinite sequences, which correspond to models for linear temporal logic. Temporal logic can be used both to develop a system specification, and to axiomatize a programming language. This axiomatization can then be used to systematically derive the theory of a program from the program text. Both the specification and the program theory will then be encoded in temporal logic, and verification hence becomes a proof problem in temporal logic.

Comparatively little work has been carried out within the agent-based systems community on axiomatizing multiagent environments. We shall review just one approach. In [120], an axiomatic approach to the verification of multiagent systems was proposed. Essentially, the idea was to use a temporal belief logic to axiomatize the properties of two multiagent programming languages. Given such an axiomatization, a program theory representing the properties of the system could be systematically derived in the way indicated above. A temporal belief logic was used for two reasons. First, a temporal component was required because, as we observed above, we need to capture the ongoing behavior of a multiagent system. A belief component was used because the agents we wish to verify are each symbolic AI systems in their own right. That is, each agent is a symbolic reasoning system, which includes a representation of its environment and desired behavior. A belief component in the logic allows us to capture the symbolic representations present within each agent. The two multiagent programming languages that were axiomatized in the temporal belief logic were Shoham's AGENT0 [100], and Fisher's Concurrent METATEM (see above). Note that this approach relies on the operation of agents being sufficiently simple that their properties can be axiomatized in the logic. It works for Shoham's AGENT0 and Fisher's Concurrent METATEM largely because these languages have a simple semantics, closely related to rule-based systems, which in turn have a simple logical semantics. For more complex agents, an axiomatization is not so straightforward. Also, capturing the semantics of concurrent execution of agents is not easy (it is, of course, an area of ongoing research in computer science generally).

Ultimately, axiomatic verification reduces to a proof problem. Axiomatic approaches to verification are thus inherently limited by the difficulty of this proof

problem. Proofs are hard enough, even in classical logic; the addition of temporal and modal connectives to a logic makes the problem considerably harder. For this reason, more efficient approaches to verification have been sought. One particularly successful approach is that of *model checking* [20]. As the name suggests, whereas axiomatic approaches generally rely on syntactic proof, model checking approaches are based on the semantics of the specification language. The model checking problem, in abstract, is quite simple: given a formula ϕ of language L , and a model M for L , determine whether or not ϕ is valid in M , i.e., whether or not $M \models_L \phi$. Model checking-based verification has been studied in connection with temporal logic. The technique once again relies upon the close relationship between models for temporal logic and finite-state machines. Suppose that ϕ is the specification for some system, and π is a program that claims to implement ϕ . Then, to determine whether or not π truly implements ϕ , we take π , and from it generate a model M_π that corresponds to π , in the sense that M_π encodes all the possible computations of π . We then determine whether or not $M_\pi \models \phi$, i.e., whether the specification formula ϕ is valid in M_π ; the program π satisfies the specification ϕ just in case the answer is “yes.” The main advantage of model checking over axiomatic verification is in complexity: model checking using the branching-time temporal logic CTL [20] can be done in polynomial time, whereas the proof problem for most modal logics is quite complex.

In [94], Rao and Georgeff present an algorithm for model checking BDI logic. More precisely, they give an algorithm for taking a logical model for their (propositional) BDI agent specification language, and a formula of the language, and determining whether the formula is valid in the model. The technique is closely based on model checking algorithms for normal modal logics [40]. They show that despite the inclusion of three extra modalities (for beliefs, desires, and intentions), into the CTL branching-time framework, the algorithm is still quite efficient, running in polynomial time. So the second step of the two-stage model checking process described above can still be done efficiently. However, it is not clear how the first step might be realized for BDI logics. Where does the logical model characterizing an agent actually come from – can it be derived from an arbitrary program π , as in mainstream computer science? To do this, we would need to take a program implemented in, say, JAVA, and from it derive the belief, desire, and intention accessibility relations that are used to give a semantics to the BDI component of the logic. Because, as we noted earlier, there is no clear relationship between the BDI logic and the concrete computational models used to implement agents, it is not clear how such a model could be derived.

One approach to this problem was presented in [122], where an imperative programming language called MABLE was presented, with an explicit BDI semantics. Model checking for the language was implemented by mapping the language

to the input language for the SPIN model checking system [54], and by reducing formulae in a restricted BDI language to the linear temporal logic format required by SPIN. Here, for example, is a sample claim that may be made about a MABLE system, which may be automatically verified by model checking:

```
claim
[]
((believe agent2
    (intend agent1
        (believe agent2 (a == 10))))
->
<>(believe agent2 (a == 10))
);
```

This claim says that it is always (`[]`) the case that if agent 2 believes that agent 1 intends that agent 2 believes that variable *a* has the value 10, then subsequently (`<>`), agent 2 will itself believe that *a* has the value 10. MABLE was developed primarily as a testbed for exploring possible semantics for agent communication, and was not intended for large-scale system verification.

Several model checkers for logics combining knowledge, time, and other modalities have been developed in recent years. For example, using techniques similar to those used for CTL model checkers [20], Raimondi and Lomuscio implemented MCMAS, a model checker that supports a variety of epistemic, temporal, and deontic logics [67, 86]. Another recent approach to model checking multi-agent systems is [48], which involves model checking temporal epistemic logics by reducing the model checking problem to a conventional LTL model checking problem.

3 Representing the Strategic Structure of a System

The second main strand of research that we describe focuses not on the cognitive states of agents, but on the *strategic structure* of the environment: what agents can achieve, either individually or in groups. The starting point for such formalisms is a model of *strategic ability*.

Over the past three decades, researchers from many disciplines have attempted to develop a general purpose logic of strategic ability. Within the artificial intelligence (AI) community, it was understood that such a logic could be used in order to gain a better understanding of planning systems [6, 32, 66]. The most notable early effort in this direction was Moore's dynamic epistemic logic, referred to above [75, 76]. Moore's work was subsequently enhanced by many other researchers, perhaps most notably Morgenstern [77, 78]. The distinctions made by

Moore and Morgenstern also informed later attempts to integrate a logic of ability into more general logics of rational action in autonomous agents [121, 124] (see [123] for a survey of such logics).

In a somewhat parallel thread of research, researchers in the philosophy of action developed a range of logics underpinned by rather similar ideas and motivations. A typical example is that of Brown, who developed a logic of individual ability in the mid-1980s [18]. Brown's main claim was that modal logic was a useful tool for the analysis of ability, and that previous – unsuccessful – attempts to characterize ability in modal logic were based on an oversimple semantics. Brown's account of the semantics of ability was as follows [18, p. 5]:

[An agent can achieve A] at a given world iff *there exists* a relevant cluster of worlds, at *every* world of which A is true.

Notice the $\exists\forall$ pattern of quantifiers in this account. Brown immediately noted that this gave the resulting logic a rather unusual flavor, neither properly existential nor properly universal [18, p. 5]:

Cast in this form, the truth condition [for ability] involves *two* metalinguistic quantifiers (one existential and one universal). In fact, [the character of the ability operator] should be a little like each.

More recently, there has been a surge of interest in logics of strategic ability, which has been sparked by two largely independent developments: Pauly's development of coalition logic [80, 81, 82, 83], and the development of the alternating-time temporal logic (ATL) by Alur, Henzinger, and Kupferman [9, 27, 38]. Although these logics are very closely related, the motivation and background of the two systems is strikingly different.

3.1 Coalition Logic

Pauly's coalition logic was developed in an attempt to shed some light on the links between logic – and in particular, modal logic – and the mathematical theory of games [79]. Pauly showed how the semantic structures underpinning a family of logics of cooperative ability could be formally understood as games of various types. He gave correspondence results between properties of the games and axioms of the logic; gave complete axiomatizations of the various resulting logics; determined the computational complexity of the satisfiability and model checking problems for his logics; and, in addition, demonstrated how these logics could be applied to the formal specification and verification of social choice procedures. The basic modal operator in Pauly's logic is of the form $[C]\phi$, where C is a set of

agents (i.e., a subset of the grand coalition Ag), and ϕ is a sentence; the intended reading is that “ C can cooperate to ensure that ϕ ”.

The semantics of cooperation modalities are given in terms of an *effectivity function*, which defines for every coalition C the states that C can cooperate to bring about. The effectivity function $E : S \rightarrow (2^{Ag} \rightarrow 2^{2^S})$, gives for any state t and coalition C a set of sets of end-states $E_C(t)$, with the intended meaning of $S \in E_C(t)$ that C can enforce the outcome to be in S (although C may not be able to pinpoint the exact outcome that emerges with this choice; this generally depends on the choices of agents outside C , or “choices” made by the environment). This effectivity function comes on a par with a modal operator $[C]$ with truth definition

$$t \models [C]\phi \text{ iff for some } S \in E_C(t) : \text{for all } s (s \models \phi \text{ iff } s \in S)$$

In words: coalition is effective for or can enforce ϕ if there is a set of states S that it is effective for, i.e., which it can choose and which is exactly the denotation of ϕ : $S = \llbracket \phi \rrbracket$. It seems reasonable to say that C is also effective for ϕ if it can choose a set of states S that “just” guarantees ϕ , i.e., for which we have $S \subseteq \llbracket \phi \rrbracket$. This will be taken care of by imposing monotonicity on effectivity functions: we will discuss constraints on effectivity at the end of this section.

In games and other structures for cooperative and competitive reasoning, effectivity functions are convenient when one is interested in the *outcomes* of the game or the encounter, and not so much about *intermediate states*, or *how* a certain state is reached. Effectivity is also a level in which one can decide whether two interaction scenarios are the same. The two games $G1$ and $G2$ in Figure 16.4 are “abstract” in the sense that they do not lead to payoffs for the players but rather to states that satisfy certain properties, encoded with propositional atoms p , q , and u . Such atoms could refer to which player is winning, but also denote other properties of an end-state, such as some distribution of resources, or “payments.” Both games are two-player games: in $G1$, player A makes the first move, which it chooses from L (Left) and R (Right). In that game, player E is allowed to choose between l and r , respectively, but only if A plays R ; otherwise the game ends after one move in the state satisfying p . In game $G2$, both players have the same repertoire of choices, but the order in which the players choose is different. It looks as if in $G1$, player A can hand over control to E , whereas the converse seems to be true for $G2$. Moreover, in $G2$, the player that is not the initiator (i.e., player A) will be allowed to make a choice, regardless of the choice of its opponent.

Despite all these differences between the two games, when we evaluate them with respect to what each coalition can *achieve*, they are the same! To be a little more precise, let us define the powers of a coalition in terms of effectivity functions E . In game $G1$, player A ’s effectivity gives $E_A(\rho_1) = \{\{a\}, \{c, d\}\}$. Similarly, player E ’s effectivity yields $\{\{a, c\}, \{a, c\}\}$: E can enforce the game to end in a or

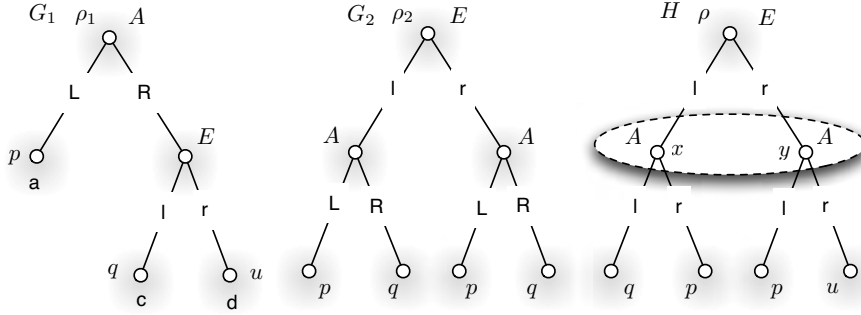


Figure 16.4: Two games G_1 and G_2 that are the same in terms of effectivity. H is an imperfect information game (see Section 3.3).

c (by playing l), but it can also enforce the end-state among a and d (by playing r). Obviously, we also have $E_{\{A,E\}}(\rho_1) = \{\{a\}, \{c\}, \{d\}\}$: players A and E together can enforce the game to end in any end-state. When reasoning about this, we have to restrict ourselves to the properties that are true in those end-states. In coalition logic, what we have just noted semantically would be described as:

$$G_1 \models [A]p \wedge [A](q \vee u) \wedge [E](p \vee q) \wedge [E](p \vee u) \wedge [A,E]p \wedge [A,E]q \wedge [A,E]r$$

Being equipped with the necessary machinery, it now is easy to see that the game G_2 verifies the same formula. Indeed, in terms of what propositions can be achieved, we are in a similar situation as in the previous game: E is effective for $\{p, q\}$ (by playing l) and also for $\{p, u\}$ (by playing r). Likewise, A is effective for $\{p\}$ (play L) and for $\{q, u\}$ (play R). The alert reader will have recognized the logical law $(p \wedge (q \vee r)) \equiv ((p \wedge q) \vee (p \wedge r))$ resembling the “equivalence” of the two games: $(p \wedge (q \vee r))$ corresponds to A ’s power in G_1 , and $((p \wedge q) \vee (p \wedge r))$ to A ’s power in G_2 . Similarly, the equivalence of E ’s powers is reflected by the logical equivalence $(p \vee (q \wedge r)) \equiv ((p \vee q) \wedge (p \vee r))$.

At the same time, the reader will have recognized the two metalinguistic quantifiers in the use of the effectivity function E , laid down in its truth-definition above. A set of outcomes S is in E_C iff *for some* choice of C , we will end up in S , under *all* choices of the complement of C (the other agents). This notion of so-called α -effectivity uses the $\exists\forall$ -order of the quantifiers: what a coalition can establish through the truth-definition mentioned above, their α -ability, is sometimes also called $\exists\forall$ -ability. Implicit within the notion of α -ability is the fact that C have *no knowledge* of the choice that the other agents make; they do not see the choice of \bar{C} (i.e., the complement of C), and then decide what to do, but rather they must make their decision *first*. This motivates the notion of β -ability (i.e., “ $\forall\exists$ ”-ability): coalition C is said to have the β -ability for ϕ if for every choice

(\perp)	$\neg[C]\perp$
(N)	$\neg[\emptyset]\neg\varphi \rightarrow [Ag]\varphi$
(M)	$[C](\varphi \wedge \psi) \rightarrow [C]\psi$
(S)	$([C_1]\varphi_1 \wedge [C_2]\varphi_2) \rightarrow [C_1 \cup C_2](\varphi_1 \wedge \varphi_2)$ where $C_1 \cap C_2 = \emptyset$
(MP)	from φ and $\varphi \rightarrow \psi$ infer ψ
(Nec)	from φ infer $[C]\varphi$

Figure 16.5: The axioms and inference rules of coalition logic.

$\sigma_{\bar{C}}$ available to \bar{C} , there exists a choice σ_C for C such that if \bar{C} choose $\sigma_{\bar{C}}$ and C choose σ_C , then φ will result. Thus C being β -able to φ means that no matter what the other agents do, C have a choice such that, if they make this choice, then φ will be true. Note the “ $\forall\exists$ ” pattern of quantifiers: C are implicitly allowed to make their choice while being aware of the choice made by \bar{C} . We will come back to information about other player’s moves in Section 3.3, and to the pairs of α and β ability in Section 3.4.

We end this section by mentioning some properties of α -abilities. The axioms for $[C]\varphi$ based on α -effectivity (or effectivity, for short) are summarized in Figure 16.5; see also Pauly [82]. The two extreme coalitions \emptyset and the grand coalition Ag are of special interest. $[Ag]\varphi$ expresses that some end-state satisfies φ , whereas $[\emptyset]\varphi$ holds if no agent needs to do anything for φ to hold in the next state.

Some of the axioms of coalition logic correspond to restrictions on effectivity functions $E : S \rightarrow (2^{Ag} \rightarrow 2^{2^S})$. First of all, we demand that $\emptyset \notin E_C$ (this guarantees axiom \perp). The function E is also assumed to be monotonic: for every coalition $C \subseteq Ag$, if $X \subseteq X' \subseteq S$, $X \in E(C)$ implies $X' \in E(C)$. This says that if a coalition can enforce an outcome in the set X , it also can guarantee the outcome to be in any superset X' of X (this corresponds to axiom (M)). An effectivity function E is C -maximal if for all X , if $\bar{X} \notin E(\bar{C})$, then $X \in E(C)$. In words: if the other agents \bar{C} cannot guarantee an outcome outside X (i.e., in \bar{X}), then C is able to guarantee an outcome in X . We require effectivity functions to be Ag -maximal. This enforces axiom (N) – Pauly’s symbol for the grand coalition is N : if the empty coalition cannot enforce an outcome satisfying φ , then the grand coalition Ag can enforce φ . The final principle governs the formation of coalitions. It states that coalitions can combine their strategies to (possibly) achieve more: E is *superadditive* if for all X_1, X_2, C_1, C_2 such that $C_1 \cap C_2 = \emptyset$, $X_1 \in E(C_1)$ and $X_2 \in E(C_2)$ imply that $X_1 \cap X_2 \in E(C_1 \cup C_2)$. This obviously corresponds to axiom (S).

One of the fascinating aspects of coalition logic is its use in social choice theory, and in particular in the specification, development, and verification of *social*

choice procedures. Consider the following scenario, adapted from [80].

Two individuals, A and B , are to choose between two outcomes, p and q . We want a procedure that will allow them to choose that will satisfy the following requirements. First, we definitely want an outcome to be possible – that is, we want the two agents to bring about either p or q . We do not want them to be able to bring about both outcomes simultaneously. Similarly, we do not want either agent to dominate: we want them both to have equal power.

The first point to note is that we can naturally axiomatize these requirements using coalition logic:

$$\begin{array}{ll} [A, B]x & x \in \{p, q\} \\ \neg[A, B](p \wedge q) & \\ \neg[x]p & x \in \{A, B\} \\ \neg[x]q & x \in \{A, B\} \end{array}$$

It should be immediately obvious how these axioms capture the requirements as stated above. Now, given a particular voting procedure, a model checking algorithm can be used to check whether or not this procedure implements the specification correctly. Moreover, a constructive proof of satisfiability for these axioms might be used to *synthesize* a procedure; or else to announce that no implementation exists.

3.2 Strategic Temporal Logic: ATL

In coalition logic one reasons about the powers of coalitions with respect to final outcomes. However, in many multiagent scenarios, the strategic considerations continue during the process. It would be interesting to study a representation language for interaction that *is* able to express the temporal differences in the two games G_1 and G_2 of Figure 16.4. *Alternating-time temporal logic* (ATL) is intended for this purpose.

Although it is similar to coalition logic, ATL emerged from a very different research community, and was developed with an entirely different set of motivations in mind. The development of ATL is closely linked with the development of branching-time temporal logics for the specification and verification of reactive systems [29, 30, 116]. Recall that CTL combines path quantifiers “A” and “E” for expressing that a certain series of events will happen on all paths and on some path, respectively, and combines these with tense modalities for expressing that something will happen eventually on some path (\Diamond), always on some path (\Box), and so on. Thus, for example, using CTL logics, one may express properties such as “on all possible computations, the system never enters a fail state”

($A \Box \neg \text{fail}$). CTL-like logics are of limited value for reasoning about *multiagent* systems, in which system components (agents) cannot be assumed to be benevolent, but may have competing or conflicting goals. The kinds of properties we wish to express of such systems are the powers that the system components have. For example, we might wish to express the fact that “agents 1 and 2 can cooperate to ensure that the system never enters a fail state.” It is not possible to capture such statements using CTL-like logics. The best one can do is either state that something will inevitably happen, or else that it may possibly happen: CTL-like logics have no notion of agency.

Alur, Henzinger, and Kupferman developed ATL in an attempt to remedy this deficiency. The key insight in ATL is that path quantifiers can be replaced by cooperation modalities: the ATL expression $\langle\langle C \rangle\rangle \phi$, where C is a group of agents, expresses the fact that the group C can cooperate to ensure that ϕ . (Thus the ATL expression $\langle\langle C \rangle\rangle \phi$ corresponds to Pauly’s $[C]\phi$.) So, for example, the fact that agents 1 and 2 can ensure that the system never enters a fail state may be captured in ATL by the following formula: $\langle\langle 1, 2 \rangle\rangle \Box \neg \text{fail}$. An ATL formula true in the root ρ_1 of game G_1 of Figure 16.4 is $\langle\langle A \rangle\rangle \bigcirc \langle\langle E \rangle\rangle \bigcirc q$: A has a strategy (i.e., play R in ρ_1) such that the next time, E has a strategy (play l) to enforce u .

Note that ATL generalizes CTL because the path quantifiers A (“on all paths...”) and E (“on some paths...”) can be simulated in ATL by the cooperation modalities $\langle\langle \emptyset \rangle\rangle$ (“the empty set of agents can cooperate to...”) and $\langle\langle Ag \rangle\rangle$ (“the grand coalition of all agents can cooperate to...”).

One reason for the interest in ATL is that it shares with its ancestor CTL the computational tractability of its model checking problem [20]. This led to the development of an ATL model checking system called MOCHA [7, 10]. With MOCHA, one specifies a model against which a formula is to be checked, using a model definition language called REACTIVE MODULES [8]. REACTIVE MODULES is a guarded command language, which provides a number of mechanisms for the structured specification of models, based upon the notion of a “module,” which is basically the REACTIVE SYSTEMS terminology for an agent. Interestingly, however, it is ultimately necessary to define for every variable in a REACTIVE MODULES system which module (i.e., agent) controls it. The powers of agents and coalitions then derive from the ability to control these variables: and this observation was a trigger for [53] to develop a system for propositional control, CL-PC, as a system in its own right. We will come briefly back to this idea in Section 3.4.

ATL has begun to attract increasing attention as a formal system for the specification and verification of multiagent systems. Examples of such work include formalizing the notion of role using ATL [99], the development of epistemic extensions to ATL [49, 50, 51], and the use of ATL for specifying and verifying

cooperative mechanisms [83].

To give a precise definition of ATL, we must first introduce the semantic structures over which formulae of ATL are interpreted. An *alternating transition system* (ATS) is a 6-tuple

$$S = \langle \Pi, Ag, Q, \pi, \delta \rangle, \text{ where:}$$

- Π is a finite, non-empty set of *Boolean variables*;
- $Ag = \{1, \dots, n\}$ is a finite, non-empty set of *agents*;
- Q is a finite, non-empty set of *states*;
- $\pi : Q \rightarrow 2^\Pi$ gives the set of Boolean variables satisfied in each state;
- $\delta : Q \times Ag \rightarrow 2^{2^Q}$ is the system transition function, which maps states and agents to the choices available to these agents. Thus $\delta(q, i)$ is the set of choices available to agent i when the system is in state q . We require that this function satisfy the requirement that for every state $q \in Q$ and every set Q_1, \dots, Q_n of choices $Q_i \in \delta(q, i)$, the intersection $Q_1 \cap \dots \cap Q_n$ is a singleton.

One can think of $\delta(q, i)$ as the possible moves agent i can make in state q . Since in general agent i cannot determine the next state on its own, each specific choice that i makes at q yields a set of possible next states Q_i , which can be further constrained by the choices of the other agents. Indeed, the constraint that $Q_1 \cap \dots \cap Q_n$ gives a singleton $\{q'\}$ resembles that the system as a whole is deterministic: once every agent i has made a decision Q_i at q , the next state q' of q is determined.

The games G_1 and G_2 of the previous section can be conceived of as special cases of alternating transition systems: *turn-based synchronous* systems, where at every decision point (state) of the system, exactly one agent is responsible for the next state. For instance, we have in G_1 that $\delta(\rho_1, A) = \{\{a\}, \{b\}\}$, and $\delta(\rho_1, E) = \{\{a, b\}\}$, denoting that E leaves the choice in ρ_1 to A . To make G_1 a real transition system, the transition function should specify choices for every state, also for the leaves a, c , and d . One could do this, for instance, by looping those states to themselves: $\delta(a, A) = \delta(a, E) = \{\{a\}\}$. In order to reason about them as leaves, one could add a proposition *end* that is true in exactly those states. Turn-based systems satisfy the following property (cf. [51]), which is not valid in ATL in general:

$$\langle\langle Ag \rangle\rangle \bigcirc \varphi \rightarrow \bigvee_{i \in Ag} \langle\langle i \rangle\rangle \bigcirc \varphi$$

An ATL formula, formed with respect to an alternating transition system $S = \langle \Pi, Ag, Q, \pi, \delta \rangle$, is then defined by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle C \rangle\rangle \bigcirc \varphi \mid \langle\langle C \rangle\rangle \square \varphi \mid \langle\langle C \rangle\rangle \varphi \mathcal{U} \varphi$$

where $p \in \Pi$ is a Boolean variable, and $C \subseteq Ag$ is a set of agents. We assume the remaining connectives (“ \perp ”, “ \rightarrow ”, “ \leftarrow ”, “ \leftrightarrow ”, “ \wedge ”) are defined as abbreviations in the usual way, and define $\langle\langle C \rangle\rangle \Diamond \phi$ as $\langle\langle C \rangle\rangle \top \cup \phi$.

To give the semantics of ATL, we need some further definitions. For two states $q, q' \in Q$ and an agent $i \in Ag$, we say that state q' is an *i-successor* of q if there exists a set $Q' \in \delta(q, i)$ such that $q' \in Q'$. Intuitively, if q' is an *i-successor* of q , then q' is a possible outcome of one of the choices available to i when the system is in state q . We denote by $\text{succ}(q, i)$ the set of *i-successors* to state q , and say that q' is simply a *successor* of q if for all agents $i \in Ag$, we have $q' \in \text{succ}(q, i)$; intuitively, if q' is a successor to q , then when the system is in state q , the agents Ag can cooperate to ensure that q' is the next state the system enters.

A *computation* of an ATS $\langle \Pi, Ag, Q, \pi, \delta \rangle$ is an infinite sequence of states $\lambda = q_0, q_1, \dots$ such that for all $u > 0$, the state q_u is a successor of q_{u-1} . A computation $\lambda \in Q^\omega$ starting in state q is referred to as a *q-computation*; if $u \in \mathbb{N}$, then we denote by $\lambda[u]$ the u th state in λ ; similarly, we denote by $\lambda[0, u]$ and $\lambda[u, \infty]$ the finite prefix q_0, \dots, q_u and the infinite suffix q_u, q_{u+1}, \dots of λ , respectively.

Intuitively, a *strategy* is an abstract model of an agent’s decision-making process; a strategy may be thought of as a kind of plan for an agent. Formally, a strategy f_i for an agent $i \in Ag$ is a total function $f_i : Q^+ \rightarrow 2^Q$, which must satisfy the constraint that $f_i(\lambda \cdot q) \in \delta(q, i)$ for all $\lambda \in Q^*$ and $q \in Q$. Given a set $C \subseteq Ag$ of agents, and an indexed set of strategies $F_C = \{f_i \mid i \in C\}$, one for each agent $i \in C$, we define $\text{out}(q, F_C)$ to be the set of possible outcomes that may occur if every agent $i \in C$ follows the corresponding strategy f_i , starting when the system is in state $q \in Q$. That is, the set $\text{out}(q, F_C)$ will contain all possible q -computations that the agents C can “enforce” by cooperating and following the strategies in F_C . Note that the “grand coalition” of all agents in the system can cooperate to uniquely determine the future state of the system, and so $\text{out}(q, F_{Ag})$ is a singleton. Similarly, the set $\text{out}(q, F_\emptyset)$ is the set of all possible q -computations of the system.

We can now give the rules defining the satisfaction relation “ \models ” for ATL, which holds between pairs of the form S, q (where S is an ATS and q is a state in S), and formulae of ATL:

$$S, q \models \top;$$

$$S, q \models p \text{ iff } p \in \pi(q) \quad (\text{where } p \in \Pi);$$

$$S, q \models \neg \phi \text{ iff } S, q \not\models \phi;$$

$$S, q \models \phi \vee \psi \text{ iff } S, q \models \phi \text{ or } S, q \models \psi;$$

$$S, q \models \langle\langle C \rangle\rangle \bigcirc \phi \text{ iff there exists a set of strategies } F_C, \text{ such that for all } \lambda \in \text{out}(q, F_C), \text{ we have } S, \lambda[1] \models \phi;$$

$S, q \models \langle\langle C \rangle\rangle \Box \varphi$ iff there exists a set of strategies F_C , such that for all $\lambda \in \text{out}(q, F_C)$, we have $S, \lambda[u] \models \varphi$ for all $u \in \mathbb{N}$;

$S, q \models \langle\langle C \rangle\rangle \varphi \mathcal{U} \psi$ iff there exists a set of strategies F_C , such that for all $\lambda \in \text{out}(q, F_C)$, there exists some $u \in \mathbb{N}$ such that $S, \lambda[u] \models \psi$, and for all $0 \leq v < u$, we have $S, \lambda[v] \models \varphi$.

Pauly's coalition logic is the fragment of ATL in which the only cooperation modalities allowed are of the form $\langle\langle C \rangle\rangle \bigcirc$ [38, 80, 81]. The truth of a coalition logic formula is determined on an ATS by using the first five items of the definition for satisfaction above. The satisfiability problem for ATL is EXPTIME-complete [27, 118], whereas for coalition logic it is PSPACE-complete in the general case [80, p. 63].

A number of variations of ATL have been proposed over the past few years, for example, to integrate reasoning about obligations into the basic framework of cooperative ability [125], to deal with quantification over coalitions [4], to add the ability to refer to strategies in the object language [108], and to add the ability to talk about preferences or goals of agents [2, 3]. In what follows, we will focus on one issue that has received considerable attention: the integration of *knowledge and ability*.

3.3 Knowledge in Strategic Temporal Logics: ATEL

The semantics of coalition logic and of ATL assume that agents have *perfect information* about the game. This is immediately apparent in the notion of strategy in ATL: by having an agent decide its next action given an element of Q^+ , this makes two strong assumptions. First of all, the agents have perfect information about the state they are in, which obviously is an idealized assumption: typically, agents don't know exactly what the state is. They may be unsure about certain facts in the state they are in, but also about the mental states of other agents, which is crucial in any strategic decision making. Second, the definition of a strategy assumes that the agents have *perfect recall*: they remember exactly what has happened in reaching the current state, so that they can make different decisions even in identical states.

We first address the issue of imperfect information. [51] adds modalities for knowledge to ATL to obtain ATEL (alternating-time temporal epistemic logic). For every individual i , add an operator K_i to the language ($K_i\varphi$ is read as “ i knows φ ”), and for every coalition G , add operators E_G (everybody in G knows), D_G (it is distributed knowledge in G), and C_G (it is common knowledge in C). The following examples of what can be expressed in ATEL are taken from [51].

As we argued in Section 1.1, action and knowledge interact in at least two ways: for some actions, in order to be able to do them properly, some knowledge

is required, and, on the other hand, actions may add to an agent's knowledge. We have already mentioned knowledge preconditions in Section 3. We can formulate knowledge preconditions quite naturally using ATEL and its variants, and the cooperation modality naturally and elegantly allows us to consider knowledge preconditions for *multiagent* plans. The requirement that in order for an agent a to be able to eventually bring about state of affairs ϕ , it must know ψ , might, as a first attempt, be specified in ATEL as $\langle\langle a \rangle\rangle \Diamond \phi \rightarrow K_a \psi$. This intuitively says that knowing ψ is a *necessary* requirement for having the ability to bring about ϕ . However, this requirement is usually too strong. For instance, in order to be able to ever open the safe, I don't necessarily in general have to know the key right *now*. A slightly better formulation might therefore be $\langle\langle a \rangle\rangle \bigcirc \phi \rightarrow K_a \psi$. As an overall constraint of the system, this property may help the agent to realize that it has to possess the right knowledge in order to achieve ϕ . But taken as a local formula, it does not tell us anything about what the agent should know if it wants to bring about ϕ the day after tomorrow, or "sometime" for that matter. Taken as a local constraint, a necessary knowledge condition to bring about ϕ might be $(\neg \langle\langle i \rangle\rangle \bigcirc \phi) \mathcal{U} K_i \psi$. This expresses that our agent is not able to open the safe until it knows its key. The other way around, an example of an ability that is generated by possessing knowledge is the following, expressing that if Bob knows that the combination of the safe is s , then he is able to open it ($\langle\langle b \rangle\rangle \bigcirc o$), as long as the combination remains unchanged.

$$K_b(c = s) \rightarrow \langle\langle b \rangle\rangle (\langle\langle b \rangle\rangle \bigcirc o) \mathcal{U} \neg(c = s) \quad (16.2)$$

One of the properties of the most widely embraced systems for knowledge (see Figure 16.1) is introspection, of which the positive variant says $K_i \phi \rightarrow K_i K_i \phi$. Another well-accepted principle of knowledge has it that from $K_i \phi$ and $K_i(\phi \rightarrow \psi)$, it follows that $K_i \psi$. Such idealized properties have been criticized, since they assume agents to be perfect reasoners who know all the consequences of their knowledge in a blow. One may also use ATEL-formulas to model limited reasoners, i.e., reasoners that do not make all inferences in one strike, but where this can be approximated over time. Positive introspection might then look like

$$K_i \psi \rightarrow \langle\langle i \rangle\rangle \bigcirc K_i K_i \psi \quad (16.3)$$

As a final example, in security protocols where agents i and j share some common secret (a key S_{ij} , for instance), what one typically wants is (16.4), expressing that i can send private information to j , without revealing the message to another agent h :

$$K_i \phi \wedge \neg K_j \phi \wedge \neg K_h \phi \wedge \langle\langle i, j \rangle\rangle \bigcirc (K_i \phi \wedge K_j \phi \wedge \neg K_h \phi) \quad (16.4)$$

Semantically, ignorance of the agents is usually modeled by specifying that each agent is unable to distinguish certain states: the more states it considers indistinguishable from a given state, the weaker its knowledge in that state. In game theory, such an indistinguishability relation is often called a partition [11]. Take the game H in Figure 16.4, for example. The dashed line labeled with agent A denotes that this agent does not know what E 's move was: A cannot distinguish state x from y . It seems reasonable to require strategies of agents to be *uniform*: if an agent does not know whether it is in state s or s' , it should make the same decision in both. But there is more to adding knowledge to decision making. Let us assume that atom p in game H denotes a win for A . Then, in the root ρ we have that $\llbracket E \rrbracket \bigcirc \langle\langle A \rangle\rangle \bigcirc p$: saying that whichever strategy E plays in ρ , in the next state A will be able to reach a winning state in the next state. Note that this is even true if we restrict ourselves to uniform strategies! We even have $H, x \models K_A \langle\langle A \rangle\rangle \bigcirc p$, saying that A knows that it has a winning strategy in x . This, of course, is only true in the *de dicto* reading of knowledge of A : it knows in x that it has a uniform strategy to win, but it does not know which one it is! To obtain a *de re* type of reading of knowledge of strategies, work is still in progress, but we refer to [58] and the recent [60, 119].

3.4 CL-PC

Both ATL and coalition logic are intended as *general purpose* logics of cooperative ability. In particular, neither has anything specific to say about the *origin* of the powers that are possessed by agents and the coalitions of which they are a member. These powers are just assumed to be implicitly defined within the effectivity structures used to give a semantics to the languages. Of course, if we give a specific *interpretation* to these effectivity structures, then we will end up with a logic with special properties. In [53], a variation of coalition logic was developed that was intended specifically to reason about *control* scenarios, as follows. The basic idea is that the overall state of a system is characterized by a finite set of variables, which for simplicity are assumed to take Boolean values. Each agent in the system is then assumed to control some (possibly empty) subset of the overall set of variables, with every variable being under the control of exactly one agent. Given this setting, in the coalition logic of propositional control (CL-PC), the operator $\Diamond_C \phi$ means that there exists some assignment of values that the coalition C can give to the variables under its control such that, assuming everything else in the system remains unchanged, then if they make this assignment, then ϕ would be true. The box dual $\Box_C \phi$ is defined in the usual way with respect to the diamond ability operator \Diamond_C . Here is a simple example:

Suppose the current state of the system is that variables p and q are

false, while variable r is true, and further suppose that agent 1 controls p and r , while agent 2 controls q . Then in this state, we have for example: $\Diamond_1(p \wedge r)$, $\neg\Diamond_1q$, and $\Diamond_2(q \wedge r)$. Moreover, for any satisfiable propositional logic formula ψ over the variables p , q , and r , we have $\Diamond_{1,2}\psi$.

The ability operator \Diamond_C in CL-PC thus captures *contingent* ability, rather along the lines of “classical planning” ability [66]: the ability under the assumption that the world only changes by the actions of the agents in the coalition operator \Diamond_C . Of course, this is not a terribly realistic type of ability, just as the assumptions of classical planning are not terribly realistic. However, in CL-PC, we can define α effectivity operators $\langle\langle C \rangle\rangle_\alpha \phi$, intended to capture something along the lines of the ATL $\langle\langle C \rangle\rangle \bigcirc \phi$, as follows:

$$\langle\langle C \rangle\rangle_\alpha \triangleq \Diamond_C \Box_{\bar{C}} \phi$$

Notice the quantifier alternation pattern $\exists\forall$ in this definition, and compare this to our discussion regarding α - and β -effectivity on page 788.

One of the interesting aspects of CL-PC is that by using this logic, it becomes possible to explicitly reason in the object language about who controls what. Let i be an agent, and let p be a system variable; let us define $ctrl(i, p)$ as follows:

$$ctrl(i, p) \triangleq (\Diamond_i p) \wedge (\Diamond_i \neg p)$$

Thus, $ctrl(i, p)$ means that i can assign p the value true, and i can also assign p the value false. It is easy to see that if $ctrl(i, p)$ is true in a system, then this means that the variable p must be under the control of agent i . Starting from this observation, a more detailed analysis of characterizing control of arbitrary formulae was developed, in terms of the variables controlled by individual agents [53]. In addition, [53] gives a complete axiomatization of CL-PC, and shows that the model checking and satisfiability problems for the logic are both PSPACE-complete. Building on this basic formalism, [52] investigates extensions into the possibility of *dynamic* control, where variables can be “passed” from one agent to another.

4 Conclusion and Further Reading

In this paper, we have motivated and introduced a number of logics of rational agency; moreover, we have investigated the role(s) that such logics might play in the *development* of artificial agents. We hope to have demonstrated that logics for rational agents are a fascinating area of study, at the confluence of many different

research areas, including logic, artificial intelligence, economics, game theory, and the philosophy of mind. We also hope to have illustrated some of the popular approaches to the theory of rational agency.

There are far too many research challenges open to identify in this article. Instead, we simply note that the search for a logic of rational agency poses a range of deep technical, philosophical, and computational research questions for the logic community. We believe that all the disparate research communities with an interest in rational agency can benefit from this search.

We presented logics for MAS from the point of view of *modal logics*. A state-of-the-art book on modal logic is [14], and, despite its maturity, the field is still developing. The references [31, 72] are reasonably standard for epistemic logic in computer science: the modal approach modeling knowledge goes back to Hintikka [46] though. In practical agent applications, information is more quantitative than just being binary-represented as knowledge and belief though: for a logical approach to reasoning about probabilities see [39].

In Section 2, we focused on logics for cognitive attitudes. Apart from the references mentioned, there are many other approaches: in the KARO framework [115], for instance, epistemic logic and dynamic logic are combined (there is work on programming KARO agents [71] and on verifying them [56]). Moreover, whereas we indicated in Section 1.1 how epistemic notions can have natural “group variants,” [5] defines some group proattitudes in the KARO setting. And in the same way as epistemics becomes interesting in a dynamic or temporal setting (see Section 1.1), there is work on logics that address the temporal aspects and the dynamics of intentions as well [109] and, indeed, on the joint revision of beliefs and intentions [57].

Whereas our focus in Section 2 was on logics for cognitive attitudes, due to space constraints we have neglected the social attitude of norms. *Deontic logic* is another example of a modal logic with roots in philosophy, with work by von Wright [117], which models attitudes like permissions and obligations for individual agents. For an overview of deontic logic in computer science, see [73]; for a proposal to add norms to the *social*, rather than the *cognitive*, aspects of a multiagent system, see, e.g., [112].

There is also work on combining deontic concepts with, for instance, knowledge [68] and the ATL-like systems we presented in Section 3: [1] for instance introduce a multidimensional CTL, where, roughly, dimensions correspond with the implementation of a norm. The formal study of norms in multiagent systems was arguably set off by work by Shoham and Tennenholtz [101, 102]. In normative systems, norms are studied more from the multiagent collective perspective, where questions arise like: which norms will emerge, why would agents adhere to them, and when is a norm “better” than another one?

There is currently a flurry of activity in logics to reason about games (see [110] for an overview paper) and modal logics for social choice (see [24] for an example). Often these are logics that refer to the information of the agents (“players,” in the case of games), and their actions (“moves” and “choices,” respectively). The logics for such scenarios are composed from the building blocks described in this chapter, with often an added logical representation of *preferences* [106] or expected utility [59].

5 Exercises

1. **Level 1** S5 Axioms:

- (a) To what extent do you believe that each of the 5 axioms for knowledge captures realistic properties of knowledge as we understand it in its everyday sense?
- (b) Now consider a god-like omniscient entity, able to reason perfectly, although with an incomplete view of its environment (i.e., not able to completely perceive its environment). To what extent do the axioms make sense for such “idealized” reasoners?

2. **Level 1** Branching time:

Give examples of branching-time models in which the following formulae are true:

- (a) $A \bigcirc p \wedge E \bigcirc \neg q$
- (b) $Ap\mathcal{U}(q \wedge \neg r)$
- (c) $(A \Box p) \wedge (E \Diamond \neg q)$

3. **Level 1** Perfect recall:

- (a) Argue that for temporal epistemic logic, perfect recall implies that $K_i \Box \phi \rightarrow \Box K_i \phi$. Hint: use the fact that $\Box \psi$ is equivalent to $(\psi \wedge \bigcirc (\phi \wedge \Box \psi))$.
- (b) What would be wrong if we replaced perfect recall with $K_i \phi \rightarrow \bigcirc K_i \phi$? Hint: think about statements that refer to “now,” or statements that refer to ignorance, i.e., $\neg K_i \psi$. See also [31, p. 130].
- (c) Discuss the converse of perfect recall, i.e., $\bigcirc K_i \phi \rightarrow K_i \bigcirc \phi$. What does it express? Discuss situations where this makes sense, and situations where it does not apply.

References

- [1] T. Ågotnes, W. van der Hoek, J. A. Rodríguez-Aguilar, C. Sierra, and M. Wooldridge. Multi-modal CTL: Completeness, complexity, and an application. *Studia Logica*, 92(1):1–26, 2009.
- [2] T. Ågotnes, W. van der Hoek, and M. Wooldridge. On the logic of coalitional games. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, Hakodate, Japan, 2006.
- [3] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Temporal qualitative coalitional games. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, Hakodate, Japan, 2006.
- [4] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Quantified coalition logic. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, 2007.
- [5] H. Aldewereld, W. van der Hoek, and J.-J. Ch. Meyer. Rational teams: Logical aspects of multi-agent systems. *Fundamenta Informaticae*, 63(2-3):159–183, 2004.
- [6] J. F. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann Publishers: San Mateo, CA, 1990.
- [7] R. Alur, L. de Alfaro, T. A. Henzinger, S. C. Krishnan, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. MOCHA user manual. University of Berkeley Report, 2000.
- [8] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(11):7–48, July 1999.
- [9] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.
- [10] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: Modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pages 521–525. Springer-Verlag: Berlin, Germany, 1998.
- [11] R. J. Aumann. Interactive epistemology I: Knowledge. *International Journal of Game Theory*, 28:263–300, 1999.
- [12] R. J. Aumann and A. Brandenburger. Epistemic conditions for Nash equilibrium. *Econometrica*, 63(5):1161–1180, 1995.

- [13] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A framework for programming in temporal logic. In *REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (LNCS Volume 430)*, pages 94–129. Springer-Verlag: Berlin, Germany, June 1989.
- [14] P. Blackburn, J. van Benthem, and F. Wolter, editors. *Handbook of Modal Logic*. Elsevier, Amsterdam, 2006. To appear.
- [15] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA, 1987.
- [16] M. E. Bratman. What is intention? In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 15–32. The MIT Press: Cambridge, MA, 1990.
- [17] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
- [18] M. A. Brown. On the logic of ability. *Journal of Philosophical Logic*, 17:1–26, 1988.
- [19] B. Chellas. *Modal Logic: An Introduction*. Cambridge University Press: Cambridge, England, 1980.
- [20] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.
- [21] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag: Berlin, Germany, 1981.
- [22] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
- [23] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
- [24] Tijmen R. Daniëls. Social choice and the logic of simple. *Journal of Logic and Computation*, 2010.
- [25] D. C. Dennett. *The Intentional Stance*. The MIT Press: Cambridge, MA, 1987.
- [26] M. d’Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 155–176. Springer-Verlag: Berlin, Germany, 1997.
- [27] G. van Drimmelen. Satisfiability in alternating-time temporal logic. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 208–217, Ottawa, Canada, 2003.

- [28] B. Dunin-Kępicz and R. Verbrugge. *Teamwork in Multi-Agent Systems: A Formal Approach*. Wiley and Sons, 2010.
- [29] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
- [30] E. A. Emerson and J. Srinivasan. Branching time temporal logic. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *REX Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency (LNCS Volume 354)*, pages 123–172. Springer-Verlag: Berlin, Germany, 1988.
- [31] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.
- [32] R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [33] K. Fischer, J. P. Müller, and M. Pischel. A pragmatic BDI architecture. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 203–218. Springer-Verlag: Berlin, Germany, 1996.
- [34] M. Fisher. A survey of Concurrent METATEM — the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag: Berlin, Germany, July 1994.
- [35] J. R. Galliers. A strategic framework for multi-agent cooperative dialogue. In *Proceedings of the Eighth European Conference on Artificial Intelligence (ECAI-88)*, pages 415–420, Munich, Federal Republic of Germany, 1988.
- [36] J. R. Galliers. *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict*. PhD thesis, Open University, UK, 1988.
- [37] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.
- [38] V. Goranko. Coalition games and alternating temporal logics. In J. van Benthem, editor, *Proceeding of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, pages 259–272, Siena, Italy, 2001.
- [39] J. Y. Halpern. *Reasoning about Uncertainty*. Massachusetts Institute of Technology, Cambridge, MA, 2003.

- [40] J. Y. Halpern and M. Y. Vardi. Model checking versus theorem proving: A manifesto. In V. Lifschitz, editor, *AI and Mathematical Theory of Computation — Papers in Honor of John McCarthy*, pages 151–176. The Academic Press: London, England, 1991.
- [41] D. Harel. *First-Order Dynamic Logic (LNCS Volume 68)*. Springer-Verlag: Berlin, Germany, 1979.
- [42] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press: Cambridge, MA, 2000.
- [43] K. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A formal embedding of AgentSpeak(L) in 3APL. In G. Antoniou and J. Slaney, editors, *Advanced Topics in Artificial Intelligence*, number 1502 in LNAI, pages 155–166. Springer, 1998.
- [44] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–402, 1999.
- [45] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A formal semantics for the core of AGENT-0. In E. Postma and M. Gyssens, editors, *Proceedings of the Eleventh Belgium-Netherlands Conference on Artificial Intelligence*, pages 27–34. 1999.
- [46] J. Hintikka. *Knowledge and Belief*. Cornell University Press: Ithaca, NY, 1962.
- [47] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.
- [48] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)*, pages 95–111. Springer-Verlag: Berlin, Germany, 2002.
- [49] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, pages 1167–1174, Bologna, Italy, 2002.
- [50] W. van der Hoek and M. Wooldridge. Model checking cooperation, knowledge, and time — a case study. *Research in Economics*, 57(3):235–265, September 2003.
- [51] W. van der Hoek and M. Wooldridge. Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

- [52] W. van der Hoek and M. Wooldridge. On the dynamics of delegation, cooperation, and control: A logical account. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, pages 701–708, Utrecht, The Netherlands, 2005.
- [53] W. van der Hoek and M. Wooldridge. On the logic of cooperation and propositional control. *Artificial Intelligence*, 164(1-2):81–119, May 2005.
- [54] G. Holzmann. The Spin model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
- [55] M. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents (Agents 99)*, pages 236–243, Seattle, WA, 1999.
- [56] U. Hustadt, C. Dixon, R. A. Schmidt, M. Fisher, J.-J. Ch. Meyer, and W. van der Hoek. Verification with the KARO agent theory (extended abstract). In J. L. Rash, C. A. Rouff, W. Truszkowski, D. Gordon, and M. G. Hinchey, editors, *Procs Formal Approaches to Agent-Based Systems, FAABS 2000*, number 1871 in LNAI, pages 33–47, 2001.
- [57] Thomas Icard, Eric Pacuit, and Yoav Shoham. Joint revision of beliefs and intention. In *KR'10*, pages 572–574, 2010.
- [58] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2-3):185–219, 2004.
- [59] Wojciech Jamroga. A temporal logic for Markov chains. In Padgham, Parkes, Müller, and Parsons, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, pages 697–704, 2008.
- [60] Wojciech Jamroga and Thomas Ågotnes. Constructive knowledge: What agents can achieve under incomplete information. *Journal of Applied Non-Classical Logics*, 17(4):423–475, 2007.
- [61] N. R. Jennings. On being responsible. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 93–102. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.
- [62] N. R. Jennings. Towards a cooperation knowledge level for collaborative problem solving. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 224–228, Vienna, Austria, 1992.
- [63] Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In M. Wooldridge, J. P.

- Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Berlin, Germany, 1996.
- [64] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1996.
- [65] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA, 1990.
- [66] V. Lifschitz. On the semantics of STRIPS. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions & Plans — Proceedings of the 1986 Workshop*, pages 1–10. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
- [67] A. Lomuscio and F. Raimondi. MCMAS: a tool for verifying multi-agent systems. In *Proceedings of The Twelfth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2006)*. Springer-Verlag: Berlin, Germany, 2006.
- [68] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.
- [69] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems — Safety*. Springer-Verlag: Berlin, Germany, 1995.
- [70] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
- [71] J.-J. Ch. Meyer, F. S. de Boer, R. M. van Eijk, K. V. Hindriks, and W. van der Hoek. On programming KARO agents. *Logic Journal of the IGPL*, 9(2):245–256, 2001.
- [72] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press: Cambridge, England, 1995.
- [73] J.-J. Ch. Meyer and R. J. Wieringa, editors. *Deontic Logic in Computer Science — Normative System Specification*. John Wiley & Sons, 1993.
- [74] G. E. Moore. A reply to my critics. In P. A. Schilpp, editor, *The Philosophy of G. E. Moore*, volume 4 of *The Library of Living Philosophers*, pages 535–677. Northwestern University, Evanston, Illinois, 1942.
- [75] R. C. Moore. Reasoning about knowledge and action. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, MA, 1977.

- [76] R. C. Moore. A formal theory of knowledge and action. In J. F. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 480–519. Morgan Kaufmann Publishers: San Mateo, CA, 1990.
- [77] L. Morgenstern. A first-order theory of planning, knowledge, and action. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 99–114. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
- [78] L. Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 867–874, Milan, Italy, 1987.
- [79] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press: Cambridge, MA, 1994.
- [80] M. Pauly. *Logic for Social Software*. PhD thesis, University of Amsterdam, 2001. ILLC Dissertation Series 2001-10.
- [81] M. Pauly. A logical framework for coalitional effectivity in dynamic procedures. *Bulletin of Economic Research*, 53(4):305–324, 2002.
- [82] M. Pauly. A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2002.
- [83] M. Pauly and M. Wooldridge. Logic for mechanism design — a manifesto. In *Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent Systems (GTDT-2003)*, Melbourne, Australia, 2003.
- [84] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science*, pages 46–57, 1977.
- [85] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190, January 1989.
- [86] F. Raimondi and A. Lomuscio. Symbolic model checking of multi-agent systems via OBDDs: An algorithm and its implementation. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, pages 630–637, New York, NY, 2004.
- [87] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038)*, pages 42–55. Springer-Verlag: Berlin, Germany, 1996.

- [88] A. S. Rao. Decision procedures for propositional linear-time Belief-Desire-Intention logics. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 33–48. Springer-Verlag: Berlin, Germany, 1996.
- [89] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.
- [90] A. S. Rao and M. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–344, 1998.
- [91] A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 498–504, Sydney, Australia, 1991.
- [92] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
- [93] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.
- [94] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.
- [95] A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.
- [96] R. Reiter. *Knowledge in Action*. The MIT Press: Cambridge, MA, 2001.
- [97] S. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA, 1986.
- [98] S. J. Rosenschein and L. P. Kaelbling. A situated view of representation and control. In P. E. Agre and S. J. Rosenschein, editors, *Computational Theories of Interaction and Agency*, pages 515–540. The MIT Press: Cambridge, MA, 1996.

- [99] M. Ryan and P.-Y. Schobbens. Agents and roles: Refinement in alternating-time temporal logic. In J.-J. Ch. Meyer and M. Tambe, editors, *Intelligent Agents VIII: Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages, ATAL-2001 (LNAI Volume 2333)*, pages 100–114, 2002.
- [100] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
- [101] Y. Shoham and M. Tennenholtz. Emergent conventions in multi-agent systems. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 225–231, 1992.
- [102] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Diego, CA, 1992.
- [103] M. P. Singh. A critical examination of the Cohen-Levesque theory of intention. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 364–368, Vienna, Austria, 1992.
- [104] V. S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous Agent Systems*. The MIT Press: Cambridge, MA, 2000.
- [105] M. Tambe. Towards flexible teamwork. *Journal of AI Research*, 7:83–124, 1997.
- [106] J. van Benthem, Patrick Girard, and Olivier Roy. Everything else being equal: A modal logic approach to ceteris paribus preferences. *Journal of Philosophical Logic*, 38:83–125, 2009.
- [107] W. van der Hoek, K. V. Hindriks, F. S. de Boer, and J.-J. Ch. Meyer. Agent programming with declarative goals. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII, Proceedings of the 6th Workshop on Agent Theories, Architectures, and Languages (ATAL)*, number 1986 in LNAI, pages 228–243, 2001.
- [108] W. van der Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, pages 157–153, Utrecht, The Netherlands, 2005.
- [109] W. van der Hoek, W. Jamroga, and M. Wooldridge. Towards a theory of intention revision. *Synthese*, 155(2):265–290, 2007.
- [110] W. van der Hoek and M. Pauly. Modal logic for games and information. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 1077–1148. Elsevier, Amsterdam, 2006.

- [111] W. van der Hoek and M. Wooldridge. Multi-agent systems. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 887–928. Elsevier, 2008.
- [112] Leendert van der Torre. Contextual deontic logic: Normative agents, violations and independence. *Annals of Mathematics and Artificial Intelligence*, 37:33–63, 2001.
- [113] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Springer, Berlin, 2007.
- [114] H. P. van Ditmarsch and B. Kooi. The secret of my success. *Synthese*, 151(2):201–232, 2006.
- [115] B. van Linder, W. van der Hoek, and J. J. Ch. Meyer. Formalizing abilities and opportunities of agents. *Fundamenta Informaticae*, 34(1,2):53–101, 1998.
- [116] M. Y. Vardi. Branching vs. linear time: Final showdown. In T. Margaria and W. Yi, editors, *Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 (LNCS Volume 2031)*, pages 1–22. Springer-Verlag: Berlin, Germany, April 2001.
- [117] G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.
- [118] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed ExpTime-complete. *Journal of Logic and Computation*, 16:765–787, 2006.
- [119] Gerhard Weiss and Peter Stone, editors. *Knowing How to Play: Uniform Choices in Logics of Agency*. ACM Press, 2006.
- [120] M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, October 1992.
- [121] M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press: Cambridge, MA, 2000.
- [122] M. Wooldridge, M.-P. Huget, M. Fisher, and S. Parsons. Model checking multi-agent systems: The MABLE language and its applications. *International Journal on Artificial Intelligence Tools*, 15(2):195–225, April 2006.
- [123] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
- [124] M. Wooldridge and N. R. Jennings. The cooperative problem solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.
- [125] M. Wooldridge and W. van der Hoek. On obligations and normative ability. *Journal of Applied Logic*, 3:396–420, 2005.

- [126] E. N. Zalta. Stanford encyclopedia of philosophy. See <http://plato.stanford.edu/>.

Chapter 17

Game-Theoretic Foundations of Multiagent Systems

Edith Elkind and Evangelos Markakis

1 Introduction

Multiagent systems can be roughly grouped into two categories: the *cooperative* systems, where all the agents share a common goal and fully cooperate in order to achieve it, and the *non-cooperative* systems, where each agent has its own desires and preferences, which may conflict with those of other agents. The former situation typically occurs when all agents are controlled by a single owner, which might be the case in multirobot exploration or search-and-rescue missions. In contrast, the latter situation is more likely to occur when agents have different owners. This is the case, for instance, in e-commerce settings, where agents represent different participants in an electronic marketplace, and all participants are trying to maximize their own utility.

Even with cooperative agents, ensuring smooth functioning of a multiagent system is not easy due to a variety of factors, ranging from unreliable communication channels to computational constraints. Going one step further and allowing for non-cooperative agents adds a new dimension to the complexity of this problem, as the agents need to be *incentivized* to choose a desirable plan of action. This difficulty is usually addressed by using the toolbox of *game theory*. Game theory is a branch of mathematical economics that models and analyzes the behavior of entities that have preferences over possible outcomes, and have to choose actions in order to implement these outcomes. Thus, it is perfectly suited to provide a

theoretical foundation for the analysis of multiagent systems that are composed of self-interested agents.

In this chapter, we give a brief overview of the foundations of game theory. We provide formal definitions of the basic concepts of game theory and illustrate their use by intuitive examples; for a more detailed motivation of the underlying theory and more elaborate examples and applications the reader is referred to [13], or, for a multiagent perspective, to [18]. Further reading suggestions are listed in Section 5.

We start by discussing normal-form games, i.e., games where all players have complete information about each others' preferences, and choose their actions simultaneously (Section 2). Then, we consider games where agents act sequentially (Section 3). Finally, we explore the effects of randomness and incomplete information (Section 4).

2 Normal-Form Games

In game theory, a *game* is an interaction among multiple self-interested agents. To begin our exposition, we focus first on games where all participants know each others' likes and dislikes, and select their actions simultaneously. To describe such a setting, we need to specify the following components:

- The set of *agents*, or *players*, that are involved in the game.
- For each agent, the set of *actions*, or *strategies*, available to this agent. We refer to a vector of chosen actions (one for each agent) as an *action profile*.
- The set of possible *outcomes*, i.e., the results of collective actions: for now, we assume that the outcomes are *deterministic*, i.e., are uniquely determined by the actions selected by all agents.
- For each agent, a *payoff function*, which assigns a numeric value (this agent's "happiness") to each outcome.

We limit ourselves to games with a finite number of players, though games with a continuum of players have been considered in the literature as well, e.g., in the context of traffic routing [16]. However, we *do not* assume that the players' sets of actions are finite.

It is assumed that all agents simultaneously choose their actions from their respective sets of available actions; these actions determine the outcome, which, in turn, determines the agents' payoffs. Since the outcomes are uniquely determined by action profiles, to simplify notation, we can omit the outcomes from the description of the game, and define payoff functions as mappings from action

profiles to real numbers. Clearly, each agent would like to select an action that maximizes its payoff; however, the optimal choice of action may depend on other agents' choices.

More formally, a normal-form game is defined as follows:

Definition 17.1 A normal-form game is a tuple $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$, where $N = \{1, \dots, n\}$ is the set of agents and for each $i \in N$, A_i is the set of actions, or strategies, available to agent i and $u_i : \times_{j \in N} A_j \rightarrow \mathbb{R}$ is the payoff function of agent i , which assigns a numeric payoff to every action profile $\mathbf{a} = (a_1, \dots, a_n) \in A_1 \times \dots \times A_n$.

We will often have to reason about the actions of one player while keeping the actions of all other players fixed. For this reason it will be convenient to have a special notation for the vector of actions of all players *other than* player i . Thus, given an action profile $\mathbf{a} = (a_1, \dots, a_n)$ and a player $i \in N$, we will denote by \mathbf{a}_{-i} the vector $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n) \in \times_{j \in N, j \neq i} A_j$. We will also write (\mathbf{a}_{-i}, b) to denote the action profile $(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n)$, where b is some action in A_i . Finally, we set $\mathcal{A} = A_1 \times \dots \times A_n$, and $\mathcal{A}_{-i} = A_1 \times \dots \times A_{i-1} \times A_{i+1} \times \dots \times A_n$.

We will now illustrate the notions we have just introduced with a simple, but instructive example; the game described in Example 17.1 is one of the most well-known normal-form games.

Example 17.1 (Prisoner's Dilemma) *Two suspects X and Y are accused of a crime. The prosecution does not have enough evidence to convict them unless one of the suspects confesses, and is therefore willing to cut a deal with the confessor. If both suspects remain silent, they will be convicted of a minor infraction, so each of them will be jailed for 1 year. If one of them confesses, but the other remains silent, the cooperative witness walks free, while the other suspect is jailed for 4 years. However, if both of them confess, each suspect will be jailed for 3 years. Both suspects are placed in solitary confinement, and each of them has to choose whether to confess (C) or remain silent (S), without consulting with the other suspect.*

This situation can be modeled as a normal-form game with $N = \{X, Y\}$, $A_X = A_Y = \{S, C\}$, and payoff functions u_X, u_Y given by

$$\begin{aligned} u_X(S, S) &= -1, & u_X(S, C) &= -4, & u_X(C, S) &= 0, & u_X(C, C) &= -3; \\ u_Y(S, S) &= -1, & u_Y(S, C) &= 0, & u_Y(C, S) &= -4, & u_Y(C, C) &= -3. \end{aligned}$$

For two-player games with a finite number of actions, the players' payoffs are usually specified more compactly using a *payoff matrix*. In this matrix, the rows correspond to the actions of the first player (who is also called the *row player*); the columns correspond to the actions of the second player (who is called the *column*

	S	C
S	$(-1, -1)$	$(-4, 0)$
C	$(0, -4)$	$(-3, -3)$

Table 17.1: The payoff matrix for the Prisoner's Dilemma.

player); and the cell at the intersection of the i -th row and the j -th column contains a pair (x, y) , where x is the first player's payoff at the action profile (a_i, a_j) , and y is the second player's payoff at this action profile. (You can also think of this matrix as a combination of two matrices, one for each player.) For instance, for the Prisoner's Dilemma example considered above, the payoff matrix is given by Table 17.1.

Now, the main question studied by game theory is how to predict the outcome of a game, i.e., how to determine what strategies the players are going to choose. Such predictions are known as *solution concepts*. We will now discuss several classic solution concepts.

2.1 Dominant Strategy

In the Prisoner's Dilemma, it is not hard to predict what a rational agent would do. Indeed, agent X can reason as follows. If Y confesses, it is more profitable for X to confess (a payoff of -3) than to remain silent (a payoff of -4). Similarly, if Y remains silent, it is more profitable for X to confess (a payoff of 0) than to remain silent (a payoff of -1). Thus, X does not need to know what Y intends to do: in any case, it prefers to confess. Player Y reasons in the same way, so the only rational outcome of this game is for both players to confess.

In this example, it happens that for each player strategy C is preferable to strategy S irrespective of the other player's choice of action. Such strategies are called *dominant*. More formally, we have the following definition.

Definition 17.2 Given a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$ and a pair of actions $a_i, a'_i \in A_i$, the action a_i is said to weakly dominate a'_i if

$$u_i(\mathbf{a}_{-i}, a_i) \geq u_i(\mathbf{a}_{-i}, a'_i) \quad (17.1)$$

for every $\mathbf{a}_{-i} \in \mathcal{A}_{-i}$, and the inequality in (17.1) is strict for at least one $\mathbf{a}_{-i} \in \mathcal{A}_{-i}$. An action a_i is said to strictly dominate a'_i if the inequality in (17.1) is strict for every $\mathbf{a}_{-i} \in \mathcal{A}_{-i}$.

A strategy a_i of agent i is said to be weakly/strictly dominant if it weakly/strictly dominates all other strategies of that agent. Similarly, a strategy a'_i of

	F	M
F	(1, 3)	(0, 0)
M	(0, 0)	(3, 1)

Table 17.2: The payoff matrix for the Battle of the Sexes.

agent i is said to be weakly/strictly dominated if it is weakly/strictly dominated by some other strategy of that agent.

Using the terminology of Definition 17.2, we can see that for both players in the Prisoner's Dilemma game, strategy C strictly dominates strategy S and is therefore a strictly dominant strategy. Suppose now that we change the description of the game so that when X confesses, Y is jailed for 3 years whether it confesses or not. Then, confessing remains a weakly dominant strategy for Y , but it is no longer strictly dominant: if X confesses, Y is indifferent between confessing and remaining silent.

It is easy to see that each player can have at most one weakly dominant strategy (and, *a fortiori*, at most one strictly dominant strategy). Further, if a player possesses such a strategy, it has a very strong incentive to choose it, as it can do no better by choosing any other strategy. However, in many games some or all of the players do not have dominant strategies.

Example 17.2 (Battle of the Sexes) *Alice and Bob would like to spend the evening together. Each of them has to choose between a football game (F) and a movie (M). Bob prefers football to the movies, Alice prefers movies to the football game, but both of them have a strong preference for spending the evening together. Thus, if they end up choosing different activities, the evening is ruined for both of them, so each of them gets a payoff of 0. If they both choose football, Bob's payoff is 3, and Alice's payoff is 1. If they both choose the movie, Alice's payoff is 3, and Bob's payoff is 1. This game can be represented by the payoff matrix given in Table 17.2, where Alice is the row player, and Bob is the column player.*

In Example 17.2, neither player has a (weakly) dominant strategy: Alice prefers M to F when Bob chooses M , but she prefers F to M when Bob chooses F . Indeed, both outcomes (F, F) and (M, M) appear reasonable in this situation. On the other hand, common sense suggests that the outcome (M, F) is less plausible than either (F, F) or (M, M) : indeed, if the outcome is (M, F) , Alice's payoff is 0, but she can unilaterally change her action to F , thereby increasing her payoff to 1. Similarly, Bob can unilaterally increase his payoff from 0 to 1 by changing his action from F to M .

We will now try to formalize the intuition that allows us to say that in this

game, (M, M) is more plausible than (F, M) , so as to apply it to a wider range of settings.

2.2 Nash Equilibrium

In the Battle of the Sexes game, the outcome (M, F) is inherently unstable: a player (or, in this particular case, both players) can change his or her action so as to increase his or her payoff. The notion of *Nash equilibrium* aims to capture the set of outcomes that are resistant to such deviations.

Definition 17.3 *Given a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$, a strategy profile $\mathbf{a} = (a_1, \dots, a_n)$ is said to be a Nash equilibrium if for every agent $i \in A$ and for every action $a'_i \in A_i$ we have*

$$u_i(\mathbf{a}_{-i}, a_i) \geq u_i(\mathbf{a}_{-i}, a'_i), \quad (17.2)$$

i.e., no agent can unilaterally increase its payoff by changing its action.

We say that an action a is a *best response* of player i to a strategy profile \mathbf{a}_{-i} if $u_i(\mathbf{a}_{-i}, a) \geq u_i(\mathbf{a}_{-i}, a')$ for every $a' \in A_i$. Thus, a Nash equilibrium is a strategy profile in which each player's action is a best response to the other players' actions.

It is immediate that in the Battle of the Sexes game (M, M) and (F, F) are Nash equilibria, but (M, F) and (F, M) are not.

We will now analyze another normal-form game. Unlike the examples that we have considered so far, it has more than two players.

Example 17.3 (Building a Bridge) *The council of a town is running a fund-raising campaign to collect money for building a bridge. The town has 10,000 inhabitants, and each of them has to decide whether to contribute or not. Any person who decides to contribute donates \$10. The bridge costs \$50,000 to build. Thus, if at least half of the inhabitants decide to contribute, then the bridge is built; if there is any surplus, it is kept by the council. Otherwise the bridge is not built, but the money donated is kept by the council for potential future use. Each person assigns a utility of 20 to having the bridge built.*

This scenario can be described by a normal-form game with 10,000 players and two actions per player ("contribute", "do not contribute"). A player's utility is \$10 if it contributes and the bridge is built, $-\$10$ if it contributes and the bridge is not built, \$20 if it does not contribute and the bridge is built, and \$0 if it does not contribute and the bridge is not built. Thus, a player prefers to contribute if and only if 4,999 other players chose to contribute, and prefers not to contribute in all other cases. Consequently, an action profile where exactly 5,000 players

	H	T
H	$(1, -1)$	$(-1, 1)$
T	$(-1, 1)$	$(1, -1)$

Table 17.3: The payoff matrix for the Matching Pennies.

contribute to building the bridge is a Nash equilibrium, and so is the profile where none of the players contributes. We leave it as an exercise for the reader to show that this game does not have any other Nash equilibria.

Nash equilibrium is perhaps the most prevalent solution concept in game theory. However, it has a few undesirable properties. First, playing a Nash equilibrium strategy is only rational if all other agents are playing according to the same Nash equilibrium; while this assumption is reasonable if other agents are known to be rational, in many real-life scenarios the rationality of other agents cannot be assumed as given. In contrast, if an agent has a dominant strategy, it does not need to assume anything about other agents. Assuming that all other agents act according to a fixed Nash equilibrium is especially problematic if the Nash equilibrium is not unique. This is the case, for instance, in the Battle of the Sexes game: if Alice and Bob have to choose their actions simultaneously and independently, there is no obvious way for them to choose between (M, M) and (F, F) . Further, there are games that do not have a Nash equilibrium, as illustrated in the next example.

Example 17.4 (Matching Pennies) *Consider a 2-player game where each of the players has a 1p coin, and has to place it on the table so that the upper side is either heads (H) or tails (T). If both coins face the same way (H, H or T, T), the first player (“matcher”) takes both coins, and hence wins 1p. Otherwise, (i.e., if the outcome is H, T or T, H), the second player (“mismatcher”) takes both coins. This game corresponds to the payoff matrix given in Table 17.3.*

It is not hard to check that in this game none of the action profiles is a Nash equilibrium: if the two players choose the same action, the second player can deviate and change its payoff from -1 to 1 , and if the two players choose different actions, the first player can profitably deviate.

However, the game in Example 17.4 does have a stable state if we allow the players to randomize. In the next section, we will discuss the effects of randomization in normal-form games.

2.3 Mixed Strategies and Mixed Nash Equilibrium

If we try to actually engage in a game of matching pennies, either as player 1 or as player 2, we will quickly realize that the best strategy is to toss the coin so that it has equal chances of landing heads or tails. The corresponding game-theoretic notion is a *mixed strategy*.

Definition 17.4 Given a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$ in which each agent's set of actions is finite (without loss of generality, we can assume that each agent has exactly m strategies available to it), a mixed strategy of an agent i with a set of actions $A_i = \{a_i^1, \dots, a_i^m\}$ is a probability distribution over A_i , i.e., a vector $\mathbf{s}_i = (s_i^1, \dots, s_i^m)$ that satisfies $s_i^j \geq 0$ for all $j = 1, \dots, m$, $s_i^1 + \dots + s_i^m = 1$. A mixed strategy profile is a vector $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ of mixed strategies (one for each agent).

The *support* $\text{supp}(\mathbf{s}_i)$ of a mixed strategy \mathbf{s}_i is the set of all actions that are assigned non-zero probability under \mathbf{s}_i : $\text{supp}(\mathbf{s}_i) = \{a_i^j \mid s_i^j > 0\}$.

Given a mixed strategy profile $(\mathbf{s}_1, \dots, \mathbf{s}_n)$, the *expected payoff* of player i is computed as

$$U_i(\mathbf{s}_1, \dots, \mathbf{s}_n) = \sum_{(a_1^1, \dots, a_n^1) \in \mathcal{A}} s_1^1 \dots s_n^1 u_i(a_1^1, \dots, a_n^1). \quad (17.3)$$

We are now ready to define our next solution concept, namely, the *mixed Nash equilibrium*.

Definition 17.5 Given a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$, a mixed strategy profile $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ is a mixed Nash equilibrium if no agent can improve its expected payoff by changing its strategy, i.e.,

$$U_i(\mathbf{s}_1, \dots, \mathbf{s}_n) \geq U_i(\mathbf{s}_1, \dots, \mathbf{s}_i', \dots, \mathbf{s}_n)$$

for every agent $i \in N$ and every mixed strategy \mathbf{s}_i' of player i .

Observe that an action a_i^j corresponds to a mixed strategy \mathbf{s}_i given by $s_i^j = 1$, $s_i^\ell = 0$ for $\ell \neq j$; we will refer to strategies of this form as *pure strategies* of player i , and the notion of Nash equilibrium defined in Section 2.2 as *pure Nash equilibrium*.

While the notion of mixed Nash equilibrium suffers from many of the same conceptual problems as the pure Nash equilibrium, as well as some additional ones, it has the following attractive property:

Theorem 17.1 Any normal-form game with a finite number of players and a finite number of strategies for each player admits a Nash equilibrium in mixed strategies.

This result was proved by John Nash in his PhD thesis [11], and is one of the cornerstones of modern game theory.

Going back to Example 17.4, we can verify that the mixed strategy profile $(\mathbf{s}_1, \mathbf{s}_2)$, where $\mathbf{s}_1 = \mathbf{s}_2 = (1/2, 1/2)$, is a mixed Nash equilibrium for the Matching Pennies game; it can be seen that it is the only mixed Nash equilibrium of this game.

We will now list several important properties of mixed Nash equilibria that can be used to simplify the computation of the equilibrium strategies; for the proofs, the reader is referred to [13].

- (1) If $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ is a mixed Nash equilibrium in a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$, then for each player $i \in N$, all actions in $\text{supp}(\mathbf{s}_i)$ are i 's best responses to $(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n)$: if $a_i \in \text{supp}(\mathbf{s}_i)$, then

$$U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, a_i, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n) \geq U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, \mathbf{s}'_i, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n)$$

for every mixed strategy \mathbf{s}'_i of player i . Hence for any two actions $a_i, a_j \in \text{supp}(\mathbf{s}_i)$, we have

$$U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, a_i, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n) = U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, a_j, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n).$$

- (2) To verify if a profile $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ is a mixed Nash equilibrium, it suffices to check deviations to pure strategies only. That is, $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ is a mixed Nash equilibrium in a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$ if for every player i and every action a_i of player i it holds that

$$U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, \mathbf{s}_i, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n) \geq U_i(\mathbf{s}_1, \dots, \mathbf{s}_{i-1}, a_i, \mathbf{s}_{i+1}, \dots, \mathbf{s}_n).$$

- (3) If a strategy profile $(a_1^{i_1}, \dots, a_n^{i_n})$ is a pure Nash equilibrium in a normal-form game $G = \langle N, (A_i)_{i \in N}, (u_i)_{i \in N} \rangle$, then it is also a mixed Nash equilibrium in G .

As an illustration, consider the Matching Pennies game. Suppose that we want to verify that the strategy profile $(\mathbf{s}_1, \mathbf{s}_2)$, where $\mathbf{s}_1 = \mathbf{s}_2 = (1/2, 1/2)$, is a mixed Nash equilibrium of this game. First, let us compute the expected payoffs. We have

$$\begin{aligned} U_1(\mathbf{s}_1, \mathbf{s}_2) &= \frac{1}{4}u_1(T, T) + \frac{1}{4}u_1(T, H) + \frac{1}{4}u_1(H, T) + \frac{1}{4}u_1(H, H) = 0; \\ U_2(\mathbf{s}_1, \mathbf{s}_2) &= \frac{1}{4}u_2(T, T) + \frac{1}{4}u_2(T, H) + \frac{1}{4}u_2(H, T) + \frac{1}{4}u_2(H, H) = 0. \end{aligned}$$

By property (2), it suffices to verify that no player can increase its payoff by deviating to a pure strategy. We have $U_1(H, \mathbf{s}_2) = \frac{1}{2}u_1(H, H) + \frac{1}{2}u_1(H, T) = 0$,

$U_1(T, s_2) = \frac{1}{2}u_1(T, H) + \frac{1}{2}u_1(T, T) = 0$, and similarly $U_2(s_1, H) = U_2(s_1, T) = 0$. Hence, (s_1, s_2) is a mixed Nash equilibrium for this game.

Now, let us try to compute the mixed Nash equilibria of the Battle of the Sexes game. By property (3), (M, M) and (F, F) are mixed Nash equilibria of this game. To determine if the game has any other mixed Nash equilibria, we will use property (1). Suppose that (s_1, s_2) is a mixed Nash equilibrium with $s_1^1 \neq 0$, $s_1^2 \neq 0$. Then by property (1) we have $U_1(F, s_2) = U_1(M, s_2)$, or, equivalently,

$$s_2^1 u_1(F, F) + s_2^2 u_1(F, M) = s_2^1 u_1(M, F) + s_2^2 u_1(M, M).$$

From this, we can derive that $s_2^1 = 3s_2^2$; together with $s_2^1 + s_2^2 = 1$, this implies that $s_2^1 = 3/4$, $s_2^2 = 1/4$. Similarly, we can derive that if $s_2^1 \neq 0$, $s_2^2 \neq 0$, then $s_1^1 = 1/4$, $s_1^2 = 3/4$. Thus, the Battle of the Sexes game admits a mixed Nash equilibrium in which each player chooses its preferred action with probability $3/4$.

2.4 Elimination of Dominated Strategies

In this section, we will discuss a solution concept that is weaker than equilibrium in dominant strategies, but stronger than Nash equilibrium.

We have argued that if a player possesses a dominant strategy, it has a very strong incentive to use it. Similarly, if one of the player's strategies is dominated, it is unlikely to ever use it (this is especially true if this strategy is strictly dominated, so in what follows we focus on strict dominance). This makes it rational for other players to assume that this strategy will never be used and therefore eliminate it from that player's list of strategies; in the modified game, some of their own strategies may become dominated, and can therefore be eliminated, etc. This procedure is usually referred to as *iterated elimination of strictly dominated strategies*. While it does not always lead to a single strategy profile, it may reduce the number of strategies considerably.

Example 17.5 Consider two players X and Y who are engaged in the following game: each player needs to name an integer between 1 and 10, and a player wins (and gets a payoff of 1) if its number is closer to half of the average of the two numbers; if both players name the same number, none of them wins. For instance, if player X names $n_X = 9$ and player Y names $n_Y = 5$, then Y wins, since 5 is closer to 3.5 than 9 is.

Player X can reason as follows. Half of the average of the two numbers never exceeds 5, so for player Y setting $n_Y = 10$ is strictly dominated by setting $n_Y = 1$. Thus, it can assume that player Y never chooses $n_Y = 10$. The same argument works for player X itself. Thus, both players can assume that the action space is, in fact, limited to all integers between 1 and 9. Hence, half of the average of

	L	R
T	(2, 2)	(3, 1)
B	(1, 3)	(3, 3)

(a) Nash equilibrium in weakly dominated strategies

	L	R
T	(4, 1)	(0, 1)
C	(0, 2)	(4, 0)
B	(1, 4)	(1, 5)

(b) Domination by a mixed strategy

Table 17.4: Elimination of strictly dominated strategies: two examples.

the two numbers does not exceed 4.5, and therefore for both players choosing 9 is strictly dominated by choosing 1. By repeating this argument, we can conclude that the only pair of strategies that cannot be eliminated in this way is (1, 1). Note that under this strategy profile neither of the players wins.

It can be shown that any pure Nash equilibrium will always survive iterated elimination of strictly dominated strategies. For instance, in Example 17.5 the strategy profile (1, 1) is a Nash equilibrium. Moreover, if an action belongs to the support of a mixed Nash equilibrium, it will not be eliminated either. Thus, iterated elimination of strictly dominated strategies can be viewed as a useful preprocessing step for computing mixed Nash equilibria. Further, the set of strategies surviving this procedure is independent of the elimination order. However, none of these statements remains true if we eliminate weakly dominated strategies rather than strictly dominated strategies: indeed, in the game given in Table 17.4a, R is weakly dominated by L and B is weakly dominated by T , yet (B, R) is a pure Nash equilibrium. Nevertheless, every game admits a (mixed) Nash equilibrium that does not have any weakly dominated strategies in its support.

We remark that the notion of dominance introduced in Definition 17.2 extends naturally to mixed strategies. Further, it can be shown that if an action a of player $i \in N$ dominates its action b in the sense of Definition 17.2, this remains to be the case when the other players are allowed to use mixed strategies. On the other hand, an action that is not dominated by any other action may nevertheless be dominated by a mixed strategy. For instance, in the game given in Table 17.4b, action B is not dominated (strictly or weakly) by either T or C , but is dominated by their even mixture. Thus, when eliminating strictly dominated strategies, we need to check for dominance by mixed strategies.

2.5 Games with Infinite Action Spaces

Whereas in all of our examples so far each player had a finite action space, all definitions in this chapter except for the ones in Section 2.3 apply to games whereby

players possess infinite action spaces. Such games may appear at first glance to be esoteric, but they are in fact quite common: a player's strategy may be the amount of time it spends on a certain activity, the amount of an infinitely divisible commodity it produces, or the amount of money it spends (money is usually assumed to be infinitely divisible in game-theoretic literature). The reader can easily verify that the notions of pure Nash equilibria, best response, and weakly/strictly dominated strategies make perfect sense in these settings. The notion of mixed Nash equilibrium can be extended to infinite action spaces as well, although a somewhat heavier machinery is required, which we do not discuss here. However, for this more general setting Nash's celebrated result [11] no longer applies and the existence of a mixed Nash equilibrium is, in general, not guaranteed.

We will now present a simple example that highlights some of the issues that arise in the analysis of games with infinite action spaces.

Example 17.6 *Alice and Bob are bidding for a painting that is being sold by means of a first-price sealed-bid auction (see Chapter 7): first, each player submits a bid in a sealed envelope, and then the envelopes are opened and the player who submits the higher bid wins and pays his or her bid; the ties are broken in Alice's favor.*

Suppose Alice values the painting at \$300, Bob values the painting at \$200, and they both know each other's values. Then if Alice wins the painting and pays \$x, her payoff is \$300 − x, whereas if Bob wins and pays \$y, his payoff is \$200 − y; note that both players' payoffs may be negative. Each player can bid any non-negative real number, i.e., each player's strategy space is \mathbb{R}_+ .

We can immediately observe that (200, 200) is a Nash equilibrium of the game described in Example 17.6. Indeed, under this bid vector Alice wins because of the tie-breaking rule, and her payoff is 100. If Alice bids more, she will have to pay more, and if she bids less, she will lose the auction, so her payoff will go down from 100 to 0. On the other hand, if Bob bids less, the outcome will remain the same, and if he bids more, he wins, but his payoff will be negative. The same argument shows that any action profile of the form (x, x) , where $200 \leq x \leq 300$, is a Nash equilibrium. In contrast, no action profile of the form (x, y) , where $x \neq y$, can be a Nash equilibrium of this game: the player who submits the higher bid has an incentive to lower his or her bid a little (so that he or she remains the winning bidder, but pays less).

Now, suppose that Alice values the painting at \$200, whereas Bob values it at \$300. We claim that in this case our game has no pure Nash equilibrium. Indeed, the same argument as above shows that in any Nash equilibrium, Alice and Bob submit the same bid (and hence Alice wins). Further, this bid is at least \$300, since otherwise Bob can profitably deviate by increasing his bid. However, this

means that Alice is losing money and can profitably deviate by lowering her bid so as to lose the auction.

2.5.1 Games with Differentiable Payoff Functions

In some classes of games with infinite action spaces, we can argue about the Nash equilibria in a more systematic way, provided that the payoff functions satisfy some “niceness” conditions. This is the case for games where for each $i \in N$ it holds that $A_i = \mathbb{R}$ and u_i is continuous and differentiable. Note that this does not hold for the auction of Example 17.6: there the payoff for Alice can drop from 100 to 0 if at the state (200, 200) she lowers her bid by an arbitrarily small amount, hence there is no continuity. Under differentiable payoff functions, and when there are no restrictions on the strategy spaces, the best response of a player to the other players’ strategies is achieved at the point where the partial derivative of u_i with respect to player i ’s choice becomes zero. Hence, we can identify the set of Nash equilibria by solving the following system of n equations, and then verifying that the obtained solutions are indeed Nash equilibria of the game.

$$\frac{\partial u_i(a_1, \dots, a_n)}{\partial a_i} = 0, \quad i = 1, \dots, n. \quad (17.4)$$

If the set A_i is a strict subset of \mathbb{R} , e.g., a bounded interval, or if there are any other restrictions on the strategy space, we should take them into account when solving the system (17.4). We illustrate this approach with the following simple example.

Example 17.7 *Consider a task allocation scenario with two agents. Each agent is assigned one individual task, and there is also a joint task that they need to work on. Each agent derives utility from both tasks and the utility is a function of the total effort spent. Let a_i denote the percentage of time that agent i devotes to the joint task, with $a_i \in [0, 1]$. The remaining percentage, $1 - a_i$, is then spent on the individual task. For given choices (a_1, a_2) of the two agents, the payoff to agent i is given by*

$$u_i(a_1, a_2) = \beta_i \ln(a_1 + a_2) + 1 - a_i, \quad i = 1, 2,$$

where $\beta_i \in (0, 1]$ shows the relative importance that each agent assigns to the joint task.

To analyze this example, note first that the tuple $(0, 0)$ is not a Nash equilibrium: at this strategy profile each player has an incentive to work on the joint task so as to avoid a payoff of $-\infty$. Hence we restrict ourselves to the region $a_1 + a_2 > 0$, so that the derivative of the logarithm is defined. Now, to find the

best response of each agent, we should solve the equations $\partial u_i / \partial a_i = 0$ and at the same time take into account the fact that a_i should be in $[0, 1]$. We obtain the following best-response equations within the feasible region:

$$a_1 = \beta_1 - a_2, \quad a_2 = \beta_2 - a_1.$$

It will be convenient to split the rest of the analysis into three cases:

Case 1: $\beta_1 = \beta_2 = \beta$. We can conclude that there are many Nash equilibria, which are given by the set

$$\{(a_1, a_2) \mid a_1 + a_2 = \beta, a_i \in [0, 1]\}.$$

For example, if $\beta = 0.7$, then the Nash equilibria are given by the tuples of the form $\{(a_1, 0.7 - a_1) \mid a_1 \in [0, 0.7]\}$.

Case 2: $\beta_1 > \beta_2$. Combining the best-response equations above with the constraint $a_i \in [0, 1]$ for $i = 1, 2$, we see that the best response of player 1 when player 2 chooses a_2 is

$$a_1 = \begin{cases} \beta_1 - a_2 & \text{if } a_2 \leq \beta_1 \\ 0 & \text{if } \beta_1 \leq a_2 \leq 1 \end{cases}$$

We have a similar best-response description for player 2. As $\beta_1 > \beta_2$, we cannot have $a_1 = \beta_1 - a_2$ and $a_2 = \beta_2 - a_1$ simultaneously, and hence either $a_1 = 0$ or $a_2 = 0$. In fact, we can see that the first of these options does not lead to a Nash equilibrium, and hence the only Nash equilibrium is $(\beta_1, 0)$, i.e., the player who assigns more value to the joint task is the only person to put any effort into it.

Case 3: $\beta_2 > \beta_1$. By a similar argument, $(0, \beta_2)$ is the only Nash equilibrium.

2.6 Zero-Sum Games

In this section, we will focus on normal-form games that express direct competition among the players. These are games where the benefit of one player equals the loss of the other players. Historically, this is the first class of games that was studied extensively before moving to general normal-form games. Indeed, zero-sum games were the focus of the first textbook on game theory by von Neumann and Morgenstern [23]. For simplicity we will only consider the two-player case. For two players, zero-sum games have a number of useful properties that make them easier to analyze than general two-player games.

Definition 17.6 A 2-player normal-form game G is a zero-sum game if for every strategy profile (\mathbf{s}, \mathbf{t}) it holds that $u_1(\mathbf{s}, \mathbf{t}) + u_2(\mathbf{s}, \mathbf{t}) = 0$.

For games with more than two players this definition is adjusted in a natural way: the sum of the utilities of all players is required to be 0. We can replace the constant 0 in Definition 17.6 with any other constant: any game in the resulting class can be transformed into a zero-sum game that has the same set of Nash equilibria by subtracting this constant from the payoff matrix of one of the players. Obviously, a two-player zero-sum game is completely specified by the payoffs of the row player, since the payoffs of the column player are simply the negative of those of the row player. We will adopt this convention within this subsection, and we will use the following running example to illustrate our exposition.

Example 17.8 Consider the 2×2 game represented by the following payoff matrix for the row player.

$$A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

One way to start thinking about how to play in such a game is to be pessimistic and assume the worst-case scenario for any chosen strategy. For the row player, this means to assume that, no matter what it chooses, the other player will make the choice that minimizes the payment to the row player, i.e., that for every row i that it might choose, the value $\min_j A_{ij}$ will be realized. Hence the row player would play so as to maximize this minimum value. In the same spirit, if the column player also thinks pessimistically, it will think that, no matter what it chooses, the other player will pick the action that maximizes the payment, i.e., for a column j the column player will end up paying $\max_i A_{ij}$. The column player will then want to ensure that it minimizes this worst-case payment. Therefore, the two players are interested in achieving, respectively, the values

$$v_1 := \max_i \min_j A_{ij}, \text{ and } v_2 := \min_j \max_i A_{ij}.$$

Let us now look at Example 17.8. We see that if the row player chooses the first row, the worst-case scenario is that it receives 2 units of payoff. If it picks the second row, it may do even worse and receive only 1 unit of payoff. Hence, it would select the first row and $v_1 = 2$. Similarly, we can argue that $v_2 = 3$ and that the column player would choose the second column. We can see, however, that this type of pessimistic play does not lead to a Nash equilibrium, since the row player would have an incentive to deviate. In fact, this game does not have a Nash equilibrium in pure strategies.

We can now carry out the same reasoning over the space of mixed strategies. For this we need to define the values

$$\bar{v}_1 := \max_{\mathbf{s}} \min_{\mathbf{t}} u_1(\mathbf{s}, \mathbf{t}), \text{ and } \bar{v}_2 := \min_{\mathbf{t}} \max_{\mathbf{s}} u_1(\mathbf{s}, \mathbf{t}),$$

where the maximization and minimization above are over the set of mixed strategies. Since we are now optimizing over a larger set, it follows that

$$v_1 \leq \bar{v}_1 \leq \bar{v}_2 \leq v_2.$$

To compute \bar{v}_1 in Example 17.8 (and generally for any 2×2 zero-sum game), we observe that for any mixed strategy $\mathbf{s} = (s_1, 1 - s_1)$ of the row player, the payoff of each pure strategy of the column player will be a linear function of s_1 . Further, when computing $\min_{\mathbf{t}} u_1(\mathbf{s}, \mathbf{t})$ for a given \mathbf{s} , it suffices to consider the pure strategies of the column player only. Hence, in the absence of dominated strategies, the solution will be found at the intersection of two linear functions. In particular, for our example, we have:

$$\begin{aligned} \bar{v}_1 &= \max_{\mathbf{s}} \min_{\mathbf{t}} u_1(\mathbf{s}, \mathbf{t}) \\ &= \max_{s_1} \min\{4s_1 + 1 - s_1, 2s_1 + 3(1 - s_1)\} \\ &= \max_{s_1} \min\{3s_1 + 1, 3 - s_1\}. \end{aligned}$$

We are seeking to maximize the minimum of two linear functions, an increasing one and a decreasing one. This is achieved at the intersection of the two lines. It follows that the strategy $\mathbf{s} = (1/2, 1/2)$ guarantees a payoff of 2.5 to the row player. Hence, we see that $\bar{v}_1 > v_1$ in this case, i.e., the row player can do better by playing a mixed strategy. In the same manner, we can analyze the behavior of the column player and find that the strategy $\mathbf{t} = (1/4, 3/4)$ guarantees to the column player that it never has to pay more than $\bar{v}_2 = 2.5$, which is better than v_2 . Furthermore, one can check that (\mathbf{s}, \mathbf{t}) is a Nash equilibrium of the game.

The fact that in Example 17.8 we obtained $\bar{v}_1 = \bar{v}_2$ and that the recommended strategies led to a Nash equilibrium was not an accident. On the contrary, this holds for every finite zero-sum game, and was one of the first theoretical results, proved by von Neumann in [22], which spurred the development of game theory. The main theorem about zero-sum games can be stated as follows:

Theorem 17.2 *For every finite zero-sum game:*

1. $\bar{v}_1 = \bar{v}_2$, and this common value, denoted by \bar{v} , is referred to as the value of the game.
2. The profile (\mathbf{s}, \mathbf{t}) where \bar{v} is achieved is a Nash equilibrium.
3. All Nash equilibria yield the same payoffs to the players, \bar{v} and $-\bar{v}$, respectively. Furthermore, if (\mathbf{s}, \mathbf{t}) and $(\mathbf{s}', \mathbf{t}')$ are Nash equilibria, then $(\mathbf{s}, \mathbf{t}')$ and $(\mathbf{s}', \mathbf{t})$ are also Nash equilibria with the same payoff.

Therefore, in zero-sum games the notion of Nash equilibrium does not face the criticism regarding the existence of multiple equilibria and the problem of choosing among them, since all equilibria provide the same payoffs and coordination is never an issue. Furthermore, from the algorithmic perspective, even though finding \bar{v} for an arbitrary zero-sum game is not as easy as for the 2×2 case of Example 17.8, it is still a tractable problem (for more details, see Section 2.7).

2.7 Computational Aspects

Many researchers have argued that for game-theoretic approaches to be relevant in practice, the game in question should admit a succinct representation and the chosen solution concept should be efficiently computable. These criteria are usually taken to mean that the representation size of a game and the running time of the algorithms for computing solution concepts should be at most polynomial in n and m , where n is the number of players and m is the number of actions available to each player.

For two-player games (and generally, for games with a constant number of players) representation complexity is not an issue: if each of the players has m actions and all payoffs can be rescaled to be integers with absolute value that does not exceed $2^{\text{poly}(m)}$, the size of the matrix representation is polynomial in m . However, if the number of players is large, the size of the matrix representation explodes: for an n -player game with two actions per player, the matrix has 2^n cells. This is, in a sense, inevitable: a standard counting argument shows that any formalism that can express every n -player game will produce exponential-sized descriptions for most games. Therefore, for multiplayer games one usually considers application-specific representation formalisms that may fail to be universally expressive, but provide a succinct encoding for the relevant class of games; two prominent examples are congestion games and graphical games (see, respectively, [21] and [7] for details).

As for efficient computability of solution concepts, strictly/weakly dominant strategies and pure Nash equilibria fare well in this respect, at least for games with a constant number of players: it is straightforward to check whether a strategy is dominant or whether a strategy profile forms a pure Nash equilibrium. However, mixed Nash equilibria fail this test: computing a Nash equilibrium is known to be complete for the complexity class PPAD [1, 4], even for two-player games with 0–1 payoffs [2]. While PPAD is a somewhat exotic complexity class, it is known to contain several important problems that many experts believe do not admit a polynomial-time algorithm. Moreover, finding a Nash equilibrium with certain desirable properties (such as, e.g., a Nash equilibrium that maximizes the sum of players' utilities) is known to be NP-complete [3, 6]. In fact, even finding an approximate Nash equilibrium, i.e., a strategy profile in which each player only

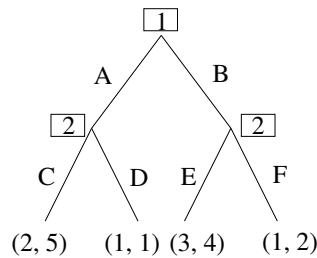


Figure 17.1: An extensive-form game with two players.

has a small incentive to deviate, appears to be computationally difficult: the best of the currently known polynomial-time algorithms are only guaranteed to output a solution in which no player can increase its payoff by more than approximately 33% [19]. If one is willing to move to higher running times, then for games with a constant number of players there exist superpolynomial algorithms (which achieve a better than exponential – yet not polynomial – running time) that can find strategy profiles where the incentive to deviate is at most ϵ , for any constant $\epsilon > 0$ (see [8, 20]). Finally, we note that for zero-sum games, a mixed Nash equilibrium can be computed efficiently. This is done by using arguments from linear programming duality, and the resulting algorithm is based on solving a single linear program.

3 Extensive-Form Games

The games studied in Section 2 clearly cannot capture situations where players move sequentially. There are, however, many examples, such as board games (chess, go, etc.), negotiation protocols, and open-cry auctions, which may evolve in several rounds, so that players take turns and make a decision after they are informed of the decisions of their opponents. In this section, we develop the tools for analyzing such games.

To start, we will describe a simple game with sequential moves that will be used as our running example throughout this section.

Example 17.9 Consider the game with two players depicted in Figure 17.1. In this game, player 1 moves first and has to decide between the alternatives A and B. After that, player 2 moves; depending on what player 1 chose, it will have to decide either between C and D or between E and F. Finally, after the move of player 2, the game ends and each player receives a payoff depending on the terminal state that the game reached. For instance, if player 1 chooses A and

player 2 chooses C , their payoffs are 2 and 5, respectively.

This example illustrates what needs to be specified to define an extensive-form game. Such a game is typically represented by a rooted tree, where the root denotes the start of the game and each leaf is a possible end of the game. Throughout this section, we assume that this tree has a finite depth, i.e., there is a finite bound on the maximum length of a path from the root to a leaf; however, nodes may have infinite fan-outs, corresponding to an infinite number of actions available to a player. All internal nodes of the tree depict states of the game. In each state, either one of the players has to make a decision or there is a random event (such as a coin toss) that determines the next state; we will refer to the choice made at a node of the latter type as a *move by nature*. Given a tree that represents a game, a *history* is simply a valid sequence of actions starting from the beginning of the game, i.e., it is a path from the root to some node of the tree. We also allow the empty set, \emptyset , to be a valid history. A *terminal history* is a history that ends in a leaf. Hence, a terminal history depicts a possible play of the game. In a tree, there is a unique path from the root to a leaf, and therefore there is a one-to-one correspondence between terminal histories and leaves. We will now make this description more formal.

Definition 17.7 An extensive-form game is given by a tuple $G = \langle N, T, (u_i)_{i \in N} \rangle$, where

- $N = \{1, \dots, n\}$ is a set of agents;
- $T = (V, E)$ is a rooted tree, where the set of nodes is partitioned into disjoint sets $V = V_1 \cup V_2 \cup \dots \cup V_n \cup V_c \cup V_L$. Here V_i , $i = 1, \dots, n$, denotes the set of nodes where it is agent i 's turn to make a decision, V_c denotes the set of nodes where a move is selected by chance, and V_L denotes the set of leaves. For every node $v \in V_c$, we are also given a probability distribution on the set of edges leaving v .
- For each $i \in N$, $u_i : V_L \rightarrow \mathbb{R}$ is a payoff function that maps each leaf of T to a real value.

Given that there is a one-to-one correspondence between terminal histories and leaves, in what follows, by a slight abuse of notation, we will sometimes view the payoff functions as mappings from terminal histories to real numbers.

To illustrate the notions introduced in Definition 17.7, consider again the game of Figure 17.1. There, the set of all histories is $H = \{\emptyset, A, B, (A, C), (A, D), (B, E), (B, F)\}$ and the set of terminal histories is $H^T = \{(A, C), (A, D), (B, E), (B, F)\}$. The set of nodes is partitioned into V_1, V_2, V_L , where V_1 contains only the root, V_2 contains the two children of the root, and

V_L is the set of the four leaves. There are no chance moves, so $V_c = \emptyset$. The payoff function of player 1 is given by

$$u_1((A,C)) = 2, \quad u_1((A,D)) = 1, \quad u_1((B,E)) = 3, \quad u_1((B,F)) = 1,$$

whereas the payoff function of player 2 is given by

$$u_2((A,C)) = 5, \quad u_2((A,D)) = 1, \quad u_2((B,E)) = 4, \quad u_2((B,F)) = 2.$$

We will now present a less abstract example that illustrates the applicability of extensive-form games in the analysis of realistic scenarios.

Example 17.10 *Alice and Bob have to share 8 identical cupcakes; the cupcakes cannot be cut into pieces, so each player has to be allocated an integer number of cupcakes. The cupcakes become stale quickly, so after each round of negotiation half of them will have to be thrown out. The negotiation procedure consists of two rounds of offers. In the first round, Alice proposes the number of cupcakes n_A that she should receive, where $0 \leq n_A \leq 8$. Bob can either accept this offer (in which case Alice gets n_A cupcakes, Bob gets $8 - n_A$ cupcakes, and the game terminates), or reject it. If Bob rejects, in the second round he can decide on the number of cupcakes n_B that he should receive; however, by this time half of the cupcakes will have perished, so we have $0 \leq n_B \leq 4$. This game can be described by the tree given in Figure 17.2.*

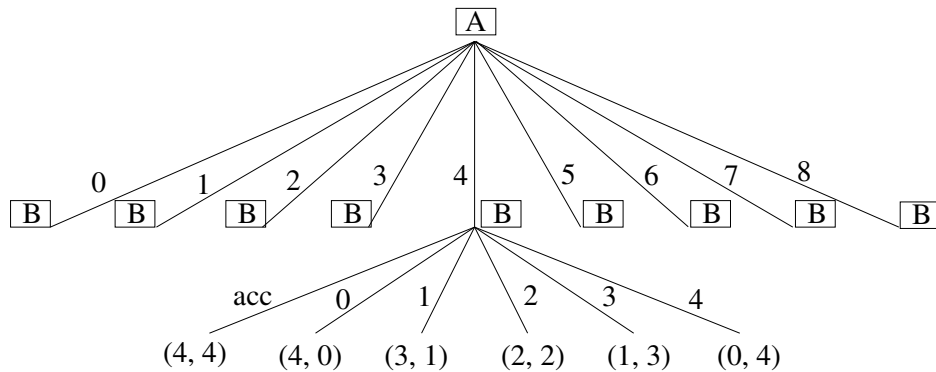


Figure 17.2: Cupcake division game. A branch labeled with i corresponds to the number of cupcakes the player decides to keep, and “acc” corresponds to Bob’s decision to accept Alice’s offer. To save space, we only show Bob’s responses to one of Alice’s actions.

3.1 Nash Equilibrium and Critiques

In order to define an appropriate solution concept for extensive-form games, we need to formalize the notion of a strategy. To begin, observe that each player i may have to move several times throughout the game. Further, at every node where it is player i 's turn to move, the available choices correspond to the edges leaving that node. A strategy in game theory is perceived as a complete plan of action, so that even a representative (e.g., a robot) could carry it out for you once you have decided upon it. Therefore, in the context of extensive-form games, a strategy for player i must specify a choice for every node that belongs to V_i , even if a node may not be reachable in a particular play of the game.

Definition 17.8 Let $G = \langle N, T, (u_i)_{i \in N} \rangle$ be an extensive-form game. For an agent $i \in N$ and a node $v \in V_i$, let $A(v)$ denote the set of actions available to agent i when the game reaches node v . Then a (pure) strategy for agent i in G is a function that assigns to each node $v \in V_i$ an action from $A(v)$.

The reason for demanding that a player specifies an action in each node of V_i is that the player cannot a priori know what the other players will do, and hence needs to be prepared if it finds itself in any node where it is required to make a decision. For instance, in Example 17.9, player 1 has two available strategies, whereas player 2 has four strategies that correspond to the choices at the two nodes of V_2 ; one such strategy is (C if A , E if B). In general, we can represent a strategy by a sequence of actions, starting from the top node of V_i and going downward in a left-to-right fashion. In this notation, the strategy described above can be encoded as (C, E) .

Given a strategy profile $\mathbf{s} = (s_1, \dots, s_n)$, let $o(\mathbf{s})$ be the terminal history (leaf) that results when players act according to the profile \mathbf{s} . The payoff for player i will then be $u_i(o(\mathbf{s}))$. Hence, analogous to Section 2, a Nash equilibrium in this context can be defined as follows.

Definition 17.9 A strategy profile $\mathbf{s} = (s_1, \dots, s_n)$ is a Nash equilibrium of an extensive-form game $G = \langle N, T, (u_i)_{i \in N} \rangle$ if for every agent $i \in N$ and every strategy s'_i of player i it holds that

$$u_i(o(\mathbf{s})) \geq u_i(o(s'_i, \mathbf{s}_{-i})).$$

Revisiting again Example 17.9, we can check that the profiles that form Nash equilibria are $(B, (C, E))$, $(B, (D, E))$, and $(A, (C, F))$. Let us inspect, for instance, the first of these profiles. Player 1 has decided to play B , and player 2 has decided to play C if it sees that player 1 played A , and E if player 1 played B . This results in payoffs of 3 and 4, respectively. If player 1 were to change to strategy A , it

would obtain a payoff of 2, hence there is no incentive to switch. Similarly, player 2 has no incentive to switch, because given that player 1 chooses B , E is the best action to take. Note that since player 1 chooses B , player 2 is indifferent toward its choices in the left subtree.

We now want to argue that Nash equilibria in extensive-form games may fail to take into account the sequential structure of the game. Let us inspect the profile $(B, (D, E))$ in more detail. This is a Nash equilibrium, because given that player 1 has chosen B , player 2 has no incentive to switch to another strategy, i.e., player 2 is simply indifferent toward the possible choices it has in the left subtree. However, imagine the situation in which player 1, either by mistake or due to other reasons, fails to play B and decides to choose A . We can see that in this case (D, E) is not a good strategy for player 2: its payoff is 1, so it would rather switch to play C and obtain a payoff of 5. Thus, the equilibrium $(B, (C, E))$ is “better” than the equilibrium $(B, (D, E))$ in the sense that in the former profile, player 2 is prepared to play the best possible action at every node where it is its turn to move, regardless of whether the first player has stuck to its plan of action.

3.2 Subgame-Perfect Equilibrium

The discussion above motivates the definition of a different solution concept, which is more suitable for extensive-form games. Before we define this concept, which was introduced by Selten [17], we need to introduce the notion of a *subgame*. Given a history h of chosen actions from the start of the game up until an internal node of the tree, the subgame following h is simply the game that remains after history h has occurred. A formalization of this is given below.

Definition 17.10 Let $G = \langle N, T, (u_i)_{i \in N} \rangle$ be an extensive-form game. Given a nonterminal history $h = (a_1, \dots, a_k)$, let p be the node where h ends and let T_p be the subtree that is rooted at p . The subgame that corresponds to history h , denoted by $G(h)$, is the game where

1. the set of agents is the same as in G .
2. the nodes of the tree T_p follow the partitioning of the original tree into the disjoint sets $V_1 \cap T_p, V_2 \cap T_p, \dots, V_n \cap T_p, V_c \cap T_p, V_L \cap T_p$.
3. the set of (terminal) histories is simply the set of all sequences h' for which (h, h') is a (terminal) history in G .
4. the payoffs for all agents at all leaves of T_p are the same as their payoffs at the respective leaves of T .

In Example 17.9, there are three subgames, namely the game G itself, which corresponds to $G(\emptyset)$; the game $G(A)$, in which player 1 has no choice, but player 2 has to choose between C and D ; and the subgame $G(B)$, where player 1 has no choice and player 2 has to choose between E and F . In other words, the number of subgames is equal to the number of nonterminal histories.

Given the notion of a subgame, what we would like to capture now is a stability concept that is robust against possible mistakes or changes in the plan of action of the other players. Consider a strategy profile $\mathbf{s} = (s_1, \dots, s_n)$. Suppose now that the game has started some time ago, and a (non-terminal) history h has occurred. This history may or may not be consistent with the profile \mathbf{s} : due to possible mistakes or unexpected changes some of the players may have deviated from \mathbf{s} . Intuitively, the profile \mathbf{s} is robust if for every player i , the strategy s_i projected on the subgame $G(h)$ is the best that i can do if, from the beginning of $G(h)$ and onward, the rest of the players adhere to \mathbf{s} . This approach can be formalized as follows.

Definition 17.11 *Given a strategy profile $\mathbf{s} = (s_1, \dots, s_n)$ in an extensive-form game $G = \langle N, T, (u_i)_{i \in N} \rangle$, let $o_h(\mathbf{s})$ be the terminal history that arises if, after the occurrence of a history h , the agents play according to the profile \mathbf{s} . Then \mathbf{s} is a subgame-perfect equilibrium (SPE) if for every agent i , for every history h after which it is agent i 's turn to move, and for every strategy s'_i of agent i in the game $G(h)$, we have*

$$u_i(o_h(\mathbf{s})) \geq u_i(o_h(s'_i, \mathbf{s}_{-i})).$$

That is, at a subgame-perfect equilibrium, each strategy is not just a best response at the start of the game, but also remains a best response to the other players' strategies at any possible point that the game may reach. It is worth looking at Example 17.9 again to clarify the definition. We can see that out of the three Nash equilibria that we identified, $(B, (C, E))$ is indeed a subgame-perfect equilibrium, since at both nodes where player 2 has to move, it is playing its optimal strategy. On the other hand, $(B, (D, E))$ is not an SPE. Indeed, B is a best response of player 1 to (D, E) ; however, in the left subtree where it is player 2's turn to move, it is not playing an optimal strategy and should have chosen C instead. Similarly, $(A, (C, F))$ is not an SPE because player 2 is not playing optimally in the right subtree. This shows that the concept of subgame-perfect equilibrium is stronger than that of a Nash equilibrium.

Fact 17.1 *Every subgame-perfect equilibrium is a Nash equilibrium; however, the converse is not true.*

3.3 Backward Induction

In this section, we will argue that subgame-perfect equilibria always exist and can be found in a systematic way; the procedure for computing them is known as *backward induction*.

In Example 17.9, we were able to conclude that there is a unique SPE, since an SPE has to be a Nash equilibrium (see Fact 17.1 mentioned above), and two of the three Nash equilibria did not qualify as an SPE. Actually, we could have derived this unique SPE by the following simple argument: Consider the smallest possible subgames, i.e., the subgames where player 2 has to move, which correspond to the left and right subtree. In the left subtree, the optimal strategy for player 2 is to play C . In the right subtree, the optimal strategy for player 2 is to play E . Next, at the root, where player 1 has to decide what to play, we can argue that given the optimal choices of player 2, the best choice for player 1 is to play B . This yields the equilibrium $(B, (C, E))$. Hence, player 1 uses the assumption that player 2 will behave as a rational agent in order to decide what action to take at the root of the tree.

The above procedure, which is known as *backward induction*, can be easily applied to any game as follows. We define the *length* of a subgame to be the length of the longest history in this subgame (i.e., the longest path from the root to a leaf). We start by finding the optimal strategies in all subgames of length 1 (games where only one player has to make a move). Then we continue inductively, moving toward the root, and at step k we find the optimal strategies of the players who move at the beginning of all subgames of length k , *given* the optimal strategies that we have found in the shorter subgames.

There are, however, a few issues that need to be handled carefully. First, if some agents have an infinite number of actions available to them at some nodes, the procedure may get stuck simply because there is no attainable optimal strategy. Consider, for instance, a game where a player has to choose a rational number in the interval $[0, B)$ for some bound B , and the payoff is an increasing function of the chosen number. Then there is no optimal strategy. Second, nodes that correspond to chance moves require special processing. For any such node we need to compute the expected payoffs for all players given the strategies we have computed in the subgames rooted at the descendants of this node. Third, an optimal strategy at a given node may not be unique. In this case, every combination of optimal strategies that we identify throughout the analysis yields a different subgame-perfect equilibrium. We illustrate the second and the third issue in the following example.

Example 17.11 Consider the game with two players given in Figure 17.3. Player 1 has to move first and decide between the alternatives A and B . After that, there is a random event represented by the two chance nodes, and then player 2

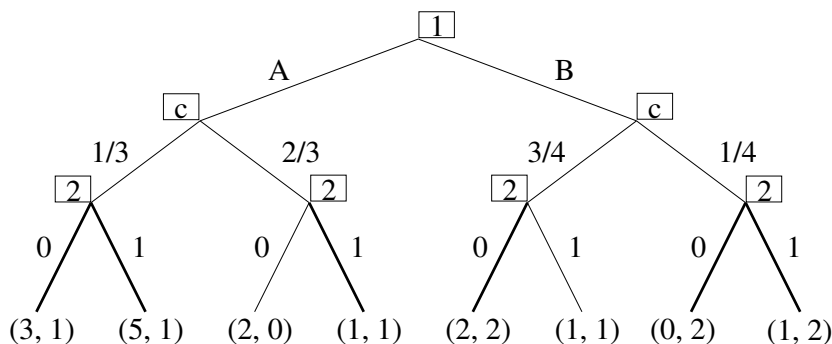


Figure 17.3: An extensive-form game with two players, chance moves, and non-unique optimal strategy for player 2.

has to move. Note that among the terminal states, there are states where player 2 is indifferent, e.g., when choosing an action in the leftmost and the rightmost subtree.

For games with finite strategy spaces, or, more generally, for games where an optimal action exists at every node (given any strategy profile for the subgame downward from this node), the extension of backward induction to handle all the above-mentioned issues is as follows:

- Starting with $k = 1$, examine at round k all subgames of length k .
- Look at the root of a subgame of length k under consideration.
 - If it is a chance node, then simply compute the expected payoffs for all players, given the probability distribution at the chance node. Do this for every combination of optimal strategies that has already been computed for the shorter subgames.
 - If it is a node where player i needs to make a move, then for every combination of optimal strategies for the shorter subgames, we find the optimal action of player i at this node. If there is more than one, we record them all, so that we continue to keep track of all combinations of optimal actions at all nodes considered so far.

Let us consider again Example 17.11. To make the notation more compact, we can use a 4-bit string to encode the strategy of player 2 for the four nodes where it has to move, with the intended meaning that 0 stands for choosing the left leaf and 1 stands for choosing the right leaf. Given this notation, the analysis proceeds as follows:

1. There are four subgames of length 1, in all of which player 2 has to move. We can see that player 2 is indifferent in the leftmost subgame and in the rightmost subgame. In the other two subgames there is a unique optimal action. Hence, in total we have four optimal strategies for player 2, namely, 0100, 0101, 1100, and 1101.
2. There are two subgames of length 2, which both start with a chance move. For all four choices of player 2 identified in the previous round, we find the expected payoffs for the two players.
 - In the left chance node, these are, respectively, $(5/3, 1)$, $(5/3, 1)$, $(7/3, 1)$, and $(7/3, 1)$.
 - In the right chance node, these are, respectively, $(6/4, 2)$, $(7/4, 2)$, $(6/4, 2)$, and $(7/4, 2)$.
3. There is only one subgame of length 3, namely the game itself, and player 1 has to decide at the root whether to play A or B . For each of the optimal actions of player 2, it can compare its expected payoff and see that it should play B only when player 2 chooses the strategy 0101. Hence we have identified four subgame-perfect equilibria, namely, $(A, 0100)$, $(B, 0101)$, $(A, 1100)$, and $(A, 0101)$.

One can prove that the above procedure indeed captures all the subgame-perfect equilibria of a game. For games where an optimal action does not exist for some node in the tree, the procedure cannot terminate with a solution, and we can conclude that the game does not possess a subgame-perfect equilibrium. Hence the main conclusion of this subsection can be summarized in the theorem below.

Theorem 17.3 *The strategy profiles returned by the backward induction procedure are precisely the subgame-perfect equilibria of the game.*

When the action spaces of all players are finite, we always have an optimal action at every node of the game. Hence, the procedure of backward induction will always terminate. This leads to the following corollary.

Corollary 17.1 *Every extensive-form game where each node has a finite fan-out has at least one subgame-perfect equilibrium.*

4 Bayesian Games

So far we have assumed that players have full knowledge regarding the game they are playing, i.e., they know all the possible plays of the game, they are always

informed about the current configuration of the game, and they know the preferences of the other players. However, in many settings in multiagent systems, players may not be aware of what the other players think, and will have to choose their strategy under uncertainty. Hence, there is a need for a framework that can capture selfish behavior in such circumstances.

We will now present a model that addresses this issue. To keep the presentation simple, we will focus on games that are played simultaneously, and we will first illustrate some of the main concepts with two examples.

4.1 Two Examples

We start with a simple variant of the Battle of the Sexes game. Again, Alice (player 1) and Bob (player 2) are choosing between football (F) and movies (M). This time, each of them is indifferent toward the two activities, so the only thing that matters to each of them is whether they end up choosing the same activity. Now, Alice has a positive payoff only when she manages to coordinate with Bob and choose the same action, and Bob is aware of this. On the other hand, Alice is uncertain about Bob, and thinks that with probability $2/3$ Bob also wants to coordinate, but with probability $1/3$ he does not. To describe things more compactly, we will say that Bob is of type m_2 (“meet”) when he wants to coordinate with Alice, and he is of type a_2 (“avoid”) when he wants to avoid Alice. Hence, Alice thinks that with probability $2/3$ the game she is playing is the one on the left-hand side of Table 17.5, whereas with probability $1/3$ she is playing the game on the right-hand side of the figure.

	F	M
F	(3, 3)	(0, 0)
M	(0, 0)	(3, 3)

(a) The game when Bob is of type m_2 .

	F	M
F	(3, 0)	(0, 3)
M	(0, 3)	(3, 0)

(b) The game when Bob is of type a_2 .

Table 17.5: An example of a Bayesian game.

We can view this situation as a game with two possible states that are completely specified by the type of player 2 (Bob). We can imagine that the game is played as follows. Just before the game starts, the actual state is not known. Then a random coin flip occurs that determines Bob’s type, i.e., his type is drawn according to some probability distribution, which is public, and hence known to Alice. For example, Bob may represent an entity chosen from some population

that follows this distribution (i.e., all guys Alice ever socialized with). Alternatively, the type may depend on certain parameters of the environment, which follow this publicly known distribution. Therefore, when the game starts, Bob is fully informed of his own type and of Alice's preferences, whereas Alice only knows the probability distribution over Bob's possible types.

To continue, we first need to determine what constitutes a strategy for Bob in this game. Bob receives a signal (in our example, learns his type) in the beginning of the game. Therefore, a strategy here is a plan of action for every possible value of the signal (for every possible realization of the state of the game). For instance, a possible strategy for Bob is to play M if his actual type is m_2 and F if his actual type is a_2 . When the game starts, Bob receives his signal (and hence learns his type) and performs the action that corresponds to this type.

To start analyzing the game, let us consider Alice first. In order for Alice to choose an action, she needs to estimate her expected payoff for all possible strategies of Bob. Since there are two possible actions, and two possible types, there are exactly four different strategies for Bob. In Table 17.6, we see the expected payoff of each action of Alice, against Bob's possible strategies. We denote each strategy of Bob by a tuple (x, y) , where x is Bob's action when he is of type m_2 and y is Bob's action when he is of type a_2 .

	(F, F)	(F, M)	(M, F)	(M, M)
F	3	2	1	0
M	0	1	2	3

Table 17.6: Payoffs in a Bayesian game.

For instance, in the strategy profile $(F, (F, M))$, where Alice chooses F , her payoff is 3 with probability $2/3$ and 0 with probability $1/3$, yielding an expected payoff of 2. It is now easy to verify that the pair of strategies $(F, (F, M))$ is stable, i.e., no player wants to change his or her actions. Indeed, given Bob's strategy (F, M) , Alice cannot improve her expected payoff by switching to M . As for Bob, when he is of type m_2 , the final outcome will be that both players select F , hence Bob receives the maximum possible payoff. Similarly, when he is of type a_2 , he will play M , which means that the outcome of the game is (F, M) ; again, this yields the maximum possible payoff for Bob, according to the right-hand side of Table 17.5. As we see, for Bob we need to check that he does not have an incentive to deviate for every possible type. This notion of stability is referred to as the *Bayes–Nash equilibrium*, which we will define formally later in this section.

We now move to a slightly more complicated example. Building on the first

example, suppose now that Alice's type is uncertain as well. Let m_1 and a_1 be the two possible types for Alice: Alice wants to meet Bob if she is of type m_1 , and avoid him if she is of type a_1 . Suppose that these types occur with probability $1/2$ each. Hence, when the game begins, the actual type of each player is determined and communicated to each player separately, and each player retains the uncertainty regarding the type of the other player.

We can see that in this example there are 4 possible states of the game, namely, the states (m_1, m_2) , (m_1, a_2) , (a_1, m_2) , and (a_1, a_2) . Alice cannot distinguish between states (m_1, m_2) and (m_1, a_2) since she receives the signal m_1 in both of them, and neither can she distinguish between (a_1, m_2) and (a_1, a_2) . Similarly, Bob cannot distinguish between the states (m_1, m_2) and (a_1, m_2) . For each player, and for each possible type of player, the beliefs induce a probability distribution on the state space, e.g., for Alice's type m_1 , the distribution assigns probability $2/3$ to the state (m_1, m_2) , $1/3$ to the state (m_1, a_2) , and 0 to the other two states, whereas for Alice's type a_1 , the distribution is $2/3$ on state (a_1, m_2) , $1/3$ on state (a_1, a_2) , and 0 on the remaining states.

To state the stability conditions, as in the first example, we assume that the players select their plan of action before they receive their signal. Hence a strategy for either player is to choose an action for every possible type that may be realized. Informally, a strategy profile is in equilibrium if for each type of player, the expected payoff cannot be improved, given (1) the belief of the player about the state that the game is in and (2) the actions chosen for each type of other player. Essentially this is as if we treat each type of a player as a separate player and ask for a Nash equilibrium in this modified game.

Before moving to the formal definitions, we claim that in the second example the tuple of strategies $((F, M), (F, F))$ is stable under this Bayesian model. To see this, consider first Alice's type m_1 . Under the given strategy profile, when Alice is of type m_1 , she chooses F . We can easily see that playing F is optimal, given Alice's belief about Bob's type and given Bob's strategy (F, F) . This is because when Alice is of type m_1 , she wants to coordinate with Bob. Since Bob's strategy is (F, F) , then obviously the best choice for Alice is to choose F . In a similar way we can verify that the same holds when Alice is of type a_1 , in which case she chooses M . Then, given that she does not want to meet Bob, this is the best she can do against (F, F) . Regarding Bob, when he is of type m_2 , his expected payoff by playing F is $3/2$, as he coordinates with Alice only half the time (according to Bob's belief). This is the best that he can achieve given his belief, hence there is no incentive to deviate. The same holds when he is of type a_2 .

4.2 Formal Definitions

Here we provide the formal definitions of a Bayesian game and of a Bayes–Nash equilibrium. Since this requires some setup that may seem heavy at first sight, we encourage the reader to first understand well the examples given above before moving on to this part.

For ease of presentation, we define here a slightly restricted version of Bayesian games. We discuss some extensions later on. Like all other games, a Bayesian game consists of a set of agents $N = \{1, \dots, n\}$, where each agent i has an available set of actions A_i . The extra ingredient in these games is that for each player i , there is a set of possible *types* T_i , modeling all possible avatars of player i . The type of player can describe his or her behavior (e.g., in the previous examples, it indicates whether the player wants to coordinate), or it may model the amount of information that the player has about the game; more generally, the type captures all relevant characteristics of the player. A profile of types for the players determines the *state* that the game is in. Hence the set of possible states is $\mathcal{T} = T_1 \times T_2 \times \dots \times T_n$, and we set $\mathcal{T}_{-i} = T_1 \times \dots \times T_{i-1} \times T_{i+1} \times \dots \times T_n$.

We assume that at the start of the game each player receives a signal specifying his or her own type, and hence a particular state is realized. The players do not observe the state of the game, but only their own type. The uncertainty about the state is captured for each player by a probability distribution on the type profiles of the remaining players, which in turn induces a distribution on the set of states. A strategy for player i in this setting is a plan of action that specifies the action that this player will choose for each possible type in T_i , i.e., a mapping $s_i : T_i \rightarrow A_i$. As before, we write $\mathbf{s} = (s_1, \dots, s_n)$ to denote the vector of all players' strategies; note, however, that the components of this vector are functions rather than numbers. Finally, for every possible type $t_i \in T_i$ and for every strategy profile, the payoff of i is the expected payoff given the belief of i about the state that the game is in, and given the strategies of all other players.

Definition 17.12 *A Bayesian game consists of a set of agents $N = \{1, \dots, n\}$, and for each agent i :*

- *a set of actions A_i ;*
- *a set of types T_i ;*
- *a belief p_i that for each type t_i of agent i specifies a probability distribution over all states consistent with t_i : $p_i(t_i, \mathbf{t}_{-i})$ is the probability that agent i assigns to the profile $\mathbf{t}_{-i} \in \mathcal{T}_{-i}$ when it is of type t_i ;*
- *a utility function u_i defined on pairs (\mathbf{a}, \mathbf{t}) , where \mathbf{a} is an action profile and \mathbf{t} is a state. Abusing notation, we write $u_i(\mathbf{s}, \mathbf{t})$ to denote the payoff of agent i*

in state \mathbf{t} when all players choose their actions according to \mathbf{s} . The expected payoff for agent i under a strategy profile \mathbf{s} , given that it is of type t_i , is then

$$\sum_{\mathbf{t}_{-i}} p_i(t_i, \mathbf{t}_{-i}) u_i(\mathbf{s}, (t_i, \mathbf{t}_{-i})).$$

In the Bayesian games discussed in the beginning of this section, both players had finitely many possible types. While Bayesian games with infinite type spaces may appear esoteric, they model a very important class of multiagent interactions, namely, auctions. Auctions and their applications to multiagent system design are discussed in detail in Chapter 7; here, we will just explain how they fit into the framework of Bayesian games.

For concreteness, consider the first-price auction with one object for sale and n players (bidders). Each player's action space is the set of all bids it can submit, i.e., the set \mathbb{R}_+ of all non-negative real numbers. Given the bids, the mechanism selects the highest bidder (say, breaking ties according to a fixed player ordering), allocates to it the item, and charges it the bid it has submitted. Each player's type is the value it assigns to the object, which can be any real number in a certain interval; by normalizing, we can assume that each player's value for the object is a real number between 0 and 10. This value determines the utility it derives from different auction outcomes: if it receives the object after bidding \$5, its utility is 1 when its value is \$6, and -1 if its value is \$4. The players' types are assumed to be drawn from a distribution \mathcal{D} over $[0, 10]^n$; at the beginning of the auction, each player learns its value, but, in general, does not observe other players' values. Then each player's belief about the state of the game when its value is v_i is the probability density function of \mathcal{D} , conditioned on i 's value being v_i . For instance, in the setting of *independent private values*, where \mathcal{D} is a product distribution, player i remains ignorant about other players' types, and therefore $p(v_i, \mathbf{v}_{-i}) = p(v'_i, \mathbf{v}_{-i})$ for any $v_i, v'_i \in [0, 10]$ and any $\mathbf{v}_{-i} \in [0, 10]^{n-1}$. In contrast, in the case of *common values*, where \mathcal{D} only assigns non-zero weight to vectors of the form (v, \dots, v) , once player i learns that its value is v_i , it assigns probability 1 to the state (v_i, v_i, \dots, v_i) and probability 0 to all other states.

We can now formalize our intuition of what it means for an outcome of a Bayesian game to be stable.

Definition 17.13 A Bayes–Nash equilibrium of a Bayesian game

$$\langle N, (A_i)_{i \in N}, (T_i)_{i \in N}, (p_i)_{i \in N}, (u_i)_{i \in N} \rangle$$

is a strategy profile $\mathbf{s} = (s_1, \dots, s_n)$ such that for each $i \in N$ and each $t_i \in T_i$ the expected payoff of agent i with type t_i at \mathbf{s} is at least as high as its expected payoff

if it were to change its strategy to any other available strategy $s'_i : T_i \rightarrow A_i$:

$$\sum_{t_{-i}} p_i(t_i, \mathbf{t}_{-i}) u_i(\mathbf{s}, (t_i, \mathbf{t}_{-i})) \geq \sum_{t_{-i}} p_i(t_i, \mathbf{t}_{-i}) u_i((s'_i, \mathbf{s}_{-i}), (t_i, \mathbf{t}_{-i})).$$

One can see from this definition that an alternative way to define a Bayes–Nash equilibrium is to consider each type of player as a separate player and consider the Nash equilibria of this expanded game. The reader can easily verify that for the second variant of the Battle of the Sexes game considered in this section (where both Alice and Bob can be of type “meet” or “avoid”), the argument given in the end of Section 4.1 shows that the strategy profile $((F, M), (F, F))$ is a Bayes–Nash equilibrium. In contrast, $((F, M), (M, F))$ is not a Bayes–Nash equilibrium of this game. Indeed, under this strategy profile, if Alice is of type m_1 , her expected payoff is 1: she coordinates with Bob only if he is of type a_2 , i.e., with probability $1/3$. On the other hand, by playing M when her type is m_1 , she would increase her expected payoff for this type to 2, i.e., she can profitably deviate from this profile.

Also, for the first-price auction example considered above, it can be verified that if $n = 2$ and each player draws its value independently at random from the uniform distribution on $[0, 10]$ (i.e., $\mathcal{D} = U[0, 10] \times U[0, 10]$), then the game admits a Bayes–Nash equilibrium in which each player bids half of its value, i.e., $s_i(v_i) = v_i/2$ for $i = 1, 2$.

5 Conclusions

We have provided a brief overview of three major classes of non-cooperative games, namely, normal-form games, extensive-form games, and Bayesian games, and presented the classic solution concepts for such games. A recent textbook of Shoham and Leyton–Brown [18] discusses game-theoretic foundations of multi-agent systems in considerably more detail. Undergraduate-level textbooks on game theory include, among others, [5] and [12]; finally, there are plenty of more advanced or graduate-level books ranging from classic ones such as [9, 14] to more modern ones like [10, 13, 15].

6 Exercises

1. **Level 1** Two players decide to play the following game. They start driving toward each other and a collision is unavoidable unless one (or both) of the drivers decides to change its driving course (chickens out). For each player, the best outcome is that it keeps driving straight while the other player chickens out. The next best outcome is that they both chicken out.

The third-best option is that the player itself chickens out while the other player drives straight, and finally the worst outcome is that they both keep driving straight till the collision occurs. Write down a normal-form game that represents this situation, and find its pure Nash equilibria.

2. **Level 1** Consider a two-player game defined by the following payoff matrix:

	<i>W</i>	<i>X</i>	<i>Y</i>	<i>Z</i>
<i>A</i>	(15, 42)	(13, 23)	(9, 43)	(0, 23)
<i>B</i>	(2, 19)	(2, 14)	(2, 23)	(1, 0)
<i>C</i>	(20, 2)	(20, 21)	(19, 4)	(3, 1)
<i>D</i>	(70, 45)	(3, 11)	(0, 45)	(1, 2)

Decide whether the following statements are true or false. Explain your answer.

- (a) *A* strictly dominates *B*.
 - (b) *Z* strictly dominates *W*.
 - (c) *C* weakly dominates *D*.
 - (d) *X* weakly dominates *W*.
 - (e) *C* is a best response to *X*.
 - (f) *Z* is a best response to *A*.
3. **Level 1** Show that $(\frac{1}{3}L + \frac{2}{3}M, \frac{1}{2}T + \frac{1}{2}B)$ is a mixed Nash equilibrium of the following game:

	<i>L</i>	<i>M</i>	<i>R</i>
<i>T</i>	(6, 22)	(3, 26)	(47, 22)
<i>C</i>	(4, 4)	(4, 2)	(99, 42)
<i>B</i>	(3, 22)	(4.5, 18)	(19, 19)

Does this game have any pure Nash equilibria?

4. **Level 1** Find all actions that are strictly dominated (possibly by mixed strategies) in the following game:

	<i>L</i>	<i>M</i>	<i>R</i>
<i>T</i>	(1, 5)	(3, 16)	(10, 10)
<i>C</i>	(7, 8)	(9, 3)	(0, 5)
<i>B</i>	(5, 0)	(7, 6)	(2, 3)

5. **Level 2** Two business partners are working on a joint project. In order for the project to be successfully implemented, it is necessary that both partners engage in the project and exert the same amount of effort. The payoff from the project is 1 unit to each partner, whereas the cost of the effort required is given by some constant c , with $0 < c < 1$. This can be modeled by the following game (where W stands for Work and S for Slack):

	S	W
S	$(0, 0)$	$(0, -c)$
W	$(-c, 0)$	$(1 - c, 1 - c)$

Find all pure and mixed Nash equilibria of this game. How do the mixed equilibria change as a function of the effort c ?

6. **Level 2** Show that every 2×2 normal-form game that has more than two pure strategy Nash equilibria possesses infinitely many mixed Nash equilibria.
7. **Level 2** Find all pure and mixed Nash equilibria in the following game.

	A	B	C	D
X	$(0, 0)$	$(5, 2)$	$(3, 4)$	$(6, 5)$
Y	$(2, 6)$	$(3, 5)$	$(5, 3)$	$(1, 0)$

Hint: One way to reason about $2 \times n$ normal-form games is to consider first a candidate mixed strategy for player 1, say $(\pi, 1 - \pi)$. Then one should consider the payoff provided by the pure strategies of player 2, given $(\pi, 1 - \pi)$, and find all the possible values of π for which a mixed Nash equilibrium is possible, i.e., one should find the values of π for which at least two of the column player's strategies can belong to the support of an equilibrium. For this, you may exploit the properties listed after Theorem 17.1.

8. **Level 2** Two students have to complete a joint assignment for a course. The final grade depends on the amount of effort exerted by the students. Each student wants to have the assignment completed, but at the same time each does not want to work much more than the other. This is captured by the following utility function: let a_1 and a_2 denote the amount of effort exerted by each of the students, $a_1, a_2 \in \mathbb{R}$. Then

$$u_i(a_i, a_j) = a_i(c + a_j - a_i), \quad i = 1, 2, j = 3 - i,$$

where c is a given constant.

This function captures the fact that if a_i exceeds a_j by at least c , then the utility of player i becomes negative. Find the pure Nash equilibria of this game.

9. **Level 2** Find the value and the Nash equilibria in the following zero-sum games:

(i)

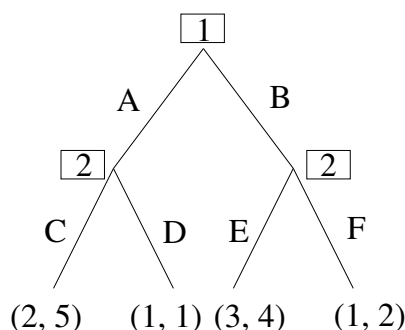
	<i>A</i>	<i>B</i>
<i>X</i>	2	7
<i>Y</i>	4	3

(ii)

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
<i>X</i>	5	2	3	4
<i>Y</i>	4	6	5	8

Hint: For (ii), try to generalize the technique described in Section 2.6 to $2 \times n$ zero-sum games.

10. **Level 2** Show that the existence of a polynomial-time algorithm for solving 3-player zero-sum games with finitely many actions per player would imply the existence of a polynomial-time algorithm for finding a Nash equilibrium in any 2-player normal-form game with finitely many actions per player.
11. **Level 2** Consider the following game G with two players P1 and P2:



- Use backward induction to compute all subgame-perfect equilibria of this game.
 - Describe the normal-form game $N(G)$ that corresponds to G .
 - Find all pure Nash equilibria of $N(G)$. Does it have any equilibria that are not subgame-perfect equilibria of G ?
12. **Level 2** Two candidates A and B compete in an election. There are n voters; k of them support A , and $m = n - k$ of them support B . Each voter can either vote (V) or abstain (A), and incurs a cost c , $0 < c < 1$, when he or she votes. Each voter obtains a payoff of 2 when its preferred candidate wins

(gets a strict majority of votes), a payoff of 1 if both candidates get the same number of votes, and a payoff of 0 if its preferred candidate loses. Thus, if a voter abstains, its payoffs for win, tie, and loss are 2, 1, and 0, respectively, and if he or she votes, the payoffs are $2 - c$, $1 - c$, and $-c$, respectively. Find the pure Nash equilibria of this game.

13. **Level 2** Consider the voting game with abstentions described in the previous exercise, but suppose that players vote one by one in a fixed order, and each player observes the actions of all players that vote before him or her. For any fixed ordering of the players, the resulting game is an extensive-form game with n players, in which each player moves exactly once and chooses between voting (V) and abstaining (A). We will say that a voter is an A -voter if he or she prefers candidate A over candidate B , and a B -voter otherwise. Compute the subgame-perfect equilibrium for the following sequences of 3 voters:
 - (a) A, B, A .
 - (b) B, A, A .
 - (c) A, A, B .
14. **Level 2** The process of backward induction in extensive-form games can be applied even when the nodes of the tree have an infinite fan-out as long as one can find an optimal action among the infinity of choices. To illustrate this, consider the following game, known in the literature as the *ultimatum game*. Two players have to decide how to split one unit of money with the following protocol: First, player 1 will offer a split of the form $(x, 1 - x)$, where $x \in [0, 1]$. Then player 2 will decide whether to accept or reject the split. If the split is accepted by player 2, then every player receives the amount specified by the split. Otherwise the split is rejected and no player receives anything. Find the subgame-perfect equilibria of this game.
15. **Level 2** Consider a variant of the previous exercise that intends to capture the fact that players in such games do care about the amount of money received by the other players. Suppose that the protocol is the same as before, but now the utility of each player is given by $u_i = x_i - \beta x_{3-i}$, $i = 1, 2$, where x_i is the amount received by player i , x_{3-i} is the amount received by the opponent, and β is a positive constant capturing how envious the players are, i.e., how much each player cares about the amount received by the other player. Find the subgame-perfect equilibria of this new game.
16. **Level 2** Consider the following extensive-form game: we have placed n disks on a vertical axis. Two players take turns, and each time one of the

players has to decide whether to remove one or two disks from the axis. The person who will remove the last disk wins 1 unit of money, paid by the other player. Suppose that player 1 moves first.

- (a) For $n = 3$ and $n = 4$, draw the corresponding representation as an extensive-form game and find the subgame-perfect equilibria. Is there a winning strategy for either of the two players?
 - (b) Generalize to arbitrary n . Is there a winning strategy for some player for every value of n ?
17. **Level 2** Two agents are involved in a dispute. Each of them can either fight or yield. The first agent is publicly known to be of medium strength; the second agent is either strong (i.e., stronger than the first agent) or weak (i.e., weaker than the first agent). The second agent knows its strength; the first agent thinks that agent 2 is strong with probability α and weak with probability $1 - \alpha$. The payoffs are as follows: if an agent decides to yield, its payoff is 0 irrespective of what the other agent chooses. If an agent fights and its opponent yields, its payoff is 1, irrespective of their strength. Finally, if they both decide to fight, the stronger agent gets a payoff of 1, and the weaker agent gets a payoff of -1 . Find all Bayes–Nash equilibria of this game if
- (a) $\alpha < 1/2$.
 - (b) $\alpha > 1/2$.

References

- [1] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of ACM*, 56(3), 2009.
- [2] Xi Chen, Shang-Hua Teng, and Paul Valiant. The approximation complexity of win-lose games. In *SODA'07: 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 159–168, 2007.
- [3] Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.
- [4] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.
- [5] Robert Gibbons. *A Primer in Game Theory*. Pearson Education, 1992.

- [6] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.
- [7] Michael Kearns. Graphical games. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, pages 159–178. Cambridge University Press, 2007.
- [8] Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *EC'03: ACM Conference on Electronic Commerce*, pages 36–41, 2003.
- [9] Duncan Luce and Howard Raiffa. *Games and Decisions*. Wiley, New York, 1957.
- [10] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, Massachusetts, 1991.
- [11] John F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.
- [12] Martin Osborne. *An Introduction to Game Theory*. Oxford University Press, 2004.
- [13] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [14] Guillermo Owen. *Game Theory*. Academic Press, New York, 2nd edition, 1982.
- [15] Hans Peters. *Game Theory—A Multi-Leveled Approach*. Springer, 2008.
- [16] Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. MIT Press, 2005.
- [17] Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4:25–55, 1975.
- [18] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations*. Cambridge University Press, 2009.
- [19] Haralampos Tsaknakis and Paul Spirakis. An optimization approach for approximate Nash equilibria. *Internet Mathematics*, 5(4):365–382, 2008.
- [20] Haralampos Tsaknakis and Paul Spirakis. Practical and efficient approximations of Nash equilibria for win-lose games based on graph spectra. In *WINE'10: 6th Workshop on Internet and Network Economics*, pages 378–390, 2010.
- [21] Berthold Vöcking. Congestion games: Optimization in competition. In *2nd Workshop on Algorithms and Complexity in Durham*, pages 11–20, 2006.
- [22] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.
- [23] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

Subject Index

- \mathcal{C} -optimality, 573
- k -optimality, 571
- k -size optimality, 571
- n -armed bandit, 430
- t -distance optimality, 573
- 3APL, 781
- 3T architecture, 42
- A-DPOP, 564
- AAII, 698, 699
- ability, 768
 - logic of, 786
 - strategic, 785
- acceptance testing, 729
- achievement goal, 598
- action, 286–289, 293, 307, 700, 766, 812, 813
 - knowledge-producing, 767, 768
 - ontic, 768
- action logic, 667
- action profile, 287, 288, 293, 812, 813
- active object, 12
- activity rule, 316
- actuator, 6
- adaptation
 - organizational, 86
- additive valuation function, 254
- ADELFE, 736
- ADEM, 698, 699
- ADOPT, 555
- agent, 103, 219
 - autonomic agent, 103
 - autonomy, 103
 - BDI, 14
 - belief-desire-intention, 14
 - heterogeneity, 103
 - incentives, 427
 - intelligent agent, 103
 - layered, 14
 - logic-based, 13
 - reactive, 13, 20
 - reward, 427
 - situated, 20
- agent architecture, 6, 13
- agent implementation
 - logic programming, 666
- agent program, 665
- agent specification, *see* agent, *see*
 - formal specification, *see*
 - temporal specification
- dynamic epistemic logic, 654
- KARO, 653
- Agent UML, 698
- agent verification
 - AJPF, 679
 - heterogeneous, 678
 - MCAPL, 679
 - rewriting, 675
- agent-based software engineering, 696
- agent-oriented software engineering, 696
- AGENT0, 781
- AGENTFACTORY, 602

- agents-entities-relation, 620
- AgentSpeak, 35
- AgentSpeak(L), 732
- AGENTSPEAK(L), 781
- agentTool III, 727
- agreement technology, 407
- AIL semantic toolkit, 677, 678
- air traffic control, 460
- AJPF, 677
- algorithmic verification, 660
 - agents, 667
 - model checking, *see* model checking
- alignment, 428
- alternating offers protocol, 151
- alternating temporal logic, 646, 650
 - semantics, 650
- alternating transition system, 792
- alternating-time temporal logic, *see* logic, ATL
- alternative, 220
- AML, 699
- anonymity (axiom), 227
- answer set programming, 605
- ant
 - colony optimization, 453
 - inspired algorithms, 453
 - system, 453
- anti-plurality rule, 228
- anytime algorithm, 356, 359
- AOSE, 696
- 3APL, 665
- 2APL, 602, 665
- application
 - social choice theory, 217
- approval voting, 228
- architecture, 101
 - component, 101
 - connector, 101
- ARCHON, 13
- argument
 - admissible, 182
 - graph, 181
 - labeling, 183
- argumentation, 168, 177
 - trust and reputation, 408
- argumentation mechanism design, 196
- argumentation protocol, 185
- argumentation scheme, 180
- Arrow's theorem, 222, 224
- artifact, 613
- asynchronous distributed optimization, 555
- ATL, *see* alternating temporal logic formulae, 650
- ATL, *see* logic
- ATL*, *see* alternating temporal logic
- attitude, 764
- auction, 304
 - combinatorial, *see* combinatorial auction
 - Dutch, *see* Dutch auction
 - elimination, *see* elimination auction
 - English, *see* English auction
 - first-price, *see* first-price auction
 - generalized first-price, *see* generalized first-price auction
 - generalized second-price, *see* generalized second-price auction
 - Japanese, *see* Japanese auction
 - kth*-price, *see* *kth*-price auction
 - position, *see* position auction
 - sealed-bid, *see* sealed-bid auction
 - second-price, *see* second-price auction
 - simultaneous ascending, *see* simultaneous ascending auction

- single-good, 304
- Vickrey, *see* second-price auction
- auction theory, 285, 304–320
- AUML, 698, 712, 717
- autonomic computing, xxxvi
- autonomous, 700
- autonomous action, 4
- autonomous agent, 3
- autonomy, 124
 - and regulation, 60
- axiomatic approach, 147
- axiomatic verification, 782
- back-edge, 563
- backward induction, 153, 834–836
- ballot-stuffing, 406
- Banks set, 226
- Barcan formula, 772
- bargaining, 147
 - bargaining function, 148
 - bargaining problem, 147
 - bargaining procedure, 151
 - cooperative bargaining, 147
 - global bargaining, 156
 - non-cooperative bargaining, 147
 - separate bargaining, 156
 - sequential bargaining, 156
- battle
 - sexes, 444
- Battle of the Sexes, 815, 817, 820, 837, 842
- Bayes–Nash equilibrium, 287, 288, 290, 291, 310
- Bayes–Nash incentive compatibility, 289
- Bayesian game, 286–289, 291–293, 307, 837, 838, 840, 841
 - state of, 840
- Bayesian game setting, 286, 286, 287, 288, 292, 293
- BDI, 386, 646, 677, 717, 727, 732, 737, 781
 - semantics, 653
- BDI architecture, 28
- BDI logic, *see* logic
- BDI model, 773
- bee
 - colony optimization, 455
 - dance, 455
- belief, 117, 385, 593, 597, 774
 - trust and reputation, 385
- belief merging, 258
- belief state, 519
 - generalized, 519
 - multiagent, 519
- belief-desire-intention model, 773
- belief-desire-intention architecture, 28
- belief-goal compatibility, 778
- Bellman Equation, 434
- best response, 308, 312, 816
 - in a Bayesian game, 308, 312
- bidding language, 250
- bidding rule, 288
- bipartisan set, 226
- BnB-ADOPT, 560
- body, *see* plan
- bold agent, 31
- bootstrapping, 434
- Borda’s rule, 228
- bounded max-sum, 574
- bounded rationality, 165
- BRAHMS, 603
- branching temporal logic, 649
 - CTL, 649
 - CTL*, 649
- bribery in elections, 237
- Brooks, Rodney, 21
- Bucklin’s rule, 231
- budget balance, 294, 301–304, 316

- ex ante*, 294
 - weak, 294, 302, 316
- budget-balanced VCG mechanism, 322
- business pattern, 125
- Büchi automata
 - basis for model checking, 661
 - synchronous product of, 661
- C++, 10
- cake-cutting algorithm, 248
- calculative rationality, 18, 20
- candidate, 220
- canonical auction family, 305
- capability, 710, 767
 - effectoric, 618
 - sensoric, 618
- cardinal preference, 220
- CARTAGO, 595, 612, 613, 621, 625, 630
- CASL, 666
- CASLve, 666
- cautious agents, 31
- CGS, *see* concurrent game structure
- change propagation, 734
- characteristic function, 331–338, 340–342, 344, 349, 351, 352, 362, 364, 368, 369, 373
- characteristic function game, 331
- choice rule, 293
- choice-set monotonicity, 301, 301, 302
- choreography, 110
- CI-net, *see* conditional importance network
- CL-PC, *see* logic
- Clarke tax, 299, 299
- click-through rate, 313, 314
- cloud computing, xxxvii
- coalition logic, *see* logic
 - of propositional control, *see* logic, CL-PC
- coalition structure, 330–335, 340, 345, 352–364, 366–374
- coalitional game
 - assignment game, 349
 - coalitional skill game, 351, 366
 - combinatorial optimization game, 348
 - convex game, 333, 334, 340, 341, 348
 - induced subgraph game, 348, 373
 - matching game, 349
 - monotone game, 333, 335, 340, 341
 - network flow game, 348, 349
 - simple game, 333, 335, 339, 340, 343–348
 - superadditive game, 333–336, 338, 339, 341–343, 350
 - weighted voting game, 344–348, 374
- coevolution, 460
- cognitive concept, 114, 117
 - belief, 114
 - goal, 114
 - intention, 114
- collective strategy, 650
- collective utility function, 251
- collusion, 317
 - reputation, 406
- column player, 814
- combinatorial auction, 250, 254, 304, 316, 315–320
- combinatorial domain, 242
- commitment, 105, 118, 316
 - assign, 119
 - cancel, 119
 - conditional, 118
 - create, 119
 - delegate, 119
 - detached, 118

- discharged, 118
- release, 119
- common prior, 307
- commonsense psychology, 769
- communication, 259
 - direct, 455
 - indirect, 455
- communication structure, 72
- communicative act, 106
 - commissive, 107
 - directive, 107
 - illocutionary point, 107
 - informative, 107
 - performative, 107
- comparing methodology, 735
- complementarity, 315, 315
- complexity factors, 430
- compliance, 106, 109, 113, 114, 129
- computation, 647
- computational complexity
 - model checking, 672
- computational tree logic, *see* logic
- computations, 793
- computing paradigm, xxxvi
- Concurrent METATEM System*, 659
- Concurrent METATEM*, 607
- concurrent game structure, *see*
 - alternating temporal logic,
 - semantics
- conditional importance network, 251
- conditional preference network, 243
- conditional utility independence, 293, 293
- Condorcet extension, 229
- Condorcet loser, 261
- Condorcet paradox, 220
- Condorcet winner, 214, 229
- conference scenario, 78
- conflict, 770
- conformance, 102
- congestion, 463
- Congo, 605
- ConGolog, 732
- CONGOLOG, 781
- constraint network, 549
- constructive manipulation, 237
- context, *see* plan
 - organizational, 609
 - social, 609
- Contract-net protocol, 495
- contraction (axiom), 225
- control
 - decentralized, 512
- control logic, 796
- control of elections, 237
- controllable entity, 618
- conventions, 490
- convergence, 425, 427
- cooperation, 770
- cooperation modality, 650
- coordination, 103
- Copeland's rule, 215, 229
- correlation device, 523
- coverage, 731
- CP-net, *see* conditional preference
 - network
- credit assignment, 425, 426
- CTL, *see* branching temporal logic
 - formulae, 650
- CTL, 790
- CTL*, *see* branching temporal logic,
 - CTL*
- CTL *, 766
- CUF, *see* collective utility function
- curse of dimensionality, 426
- daemon, 8
- DARPA
 - COORDINATORS, 531
- DARPA Knowledge Sharing Effort, 116

- DBA, 567
- DCOP, 550
- deal, 256
 - individually rational deal, 256
- debate, 191
- debugging, 729
- DEC-POMDP, 513
 - complexity, 518
 - observation independence, 519
 - transition independence, 519
 - value preserving transformation, 524
- DEC-POMDP algorithm
 - DP-JESP, 520
 - exact dynamic programming, 521
 - MBDP, 522
 - non-linear programming, 526
 - policy iteration, 525
- decision making, 443
- decision procedure, 643
- declarative achievement goal, 594, 599
- declarative goal, 598
- deduction, 14, 761
- deduction rule, 15
- deductive verification, 660
 - agents, 663
- delayed feedback, 426
- deliberation, 28, 165
- Dennett, Daniel, 769
- deontic logic, *see* logic
- deployment context, 613
- deployment diagram, 727
- design, 710
 - see also* detailed design 717
- design artifact, 696
- DESIRE, 698, 699
- desire, 653, 774
- destructive manipulation, 237
- detailed design, 717
- dialogue, 186
 - system, 188
- dilemma
 - exploration-exploitation, 434
 - Prisoner's, 444
- direct mechanism, 289, 289, 290, 294, 297, 308
- direct-revelation mechanism, 194
- discount factor, 26, 434
- dispositional trust, 393
- dispute, 186
- distance rationalizability, 231
- distributed breakout algorithm, 567
- distributed constraint handling, 547
- distributed constraint optimization problem, 550
- distributed constraint processing, 550
- distributed interpretation, 493
- distributed sensor network, 491, 496, 529
- distributed stochastic algorithm, 566
- distributed system, 101
- divisible issue, 151
- Dodgson's rule, 216, 230
- domain predicate, 16
- domain restriction, 233
- dominance
 - strict, 814
 - weak, 814
- dominant strategy, 193, 287–291, 297–299, 302, 308–310, 314
- double auction, 304
- DPOP, 561
- DSA, 566
- DSA-2, 572
- Duggan-Schwartz Theorem, 240
- Dutch auction, 306, 306, 310
- DyLog, 667
- dynamic logic, 652, *see* logic
- dynamic programming, 345, 354, 355, 368, 369, 373, 374, 436

- dynamic programming optimization protocol, 561
- early requirement, 709
- ebBP, 110
- eCAT, 730
- economic efficiency, *see* efficiency
- effectivity function, 787
- effector, 6
- efficiency, 294, 294, 295, 296, 303, 311
- egalitarianism, 251
- EIS, 612, 617, 620
- electronic institution, 611
- emergence, 737
- emergent behavior, 20
- emergent system, 736
- encapsulation, 11
- endpoint, 109
- English auction, 304, 305, 305, 306, 308, 309, 314
- environment, 425, 453, 609, 611, 700
 - accessible, 6
 - continuous, 7
 - deterministic, 6
 - discrete, 7
 - dynamic, 7
 - episodic, 6
 - inaccessible, 6
 - interface standard, 620
 - management system, 620
 - non-deterministic, 6
 - non-episodic, 6
 - static, 7
- envy-freeness, 253
- epistemic logic, *see* logic
 - dynamic, 768
- equilibrium
 - Bayes–Nash, 838, 841
 - mixed Nash, 818, 827
 - Nash, 444
 - Nash (of an extensive-form game), 831
 - point, 427
 - pure Nash, 816, 818, 827
 - subgame-perfect, 833–836
- equilibrium in dominant strategies, 287
- ergodic set, 438
- evaluation, 458
- event, 700, 701
 - and state, 75
- event selection, 600
- evolutionary
 - algorithm, 458
 - computation, 458
 - dynamics, 451
 - game theory, 430, 443, 448
 - stable strategy, 445
- ex ante* budget balance, *see* budget balance, *ex ante*
- ex post* equilibrium, 288
- executable specification, 603
- executable temporal logic, *see* temporal execution
- execution, 761
- execution as model building, 780
- expansion (axiom), 225
- expected payoff, 818
- expected reward, 26
- expected utility,
- emphex interim311
 - ex interim*, 298, 313, 314
- expert system, 12, 115
- explicit representation, 243, 249
- exploration
 - ϵ -greedy, 435
 - Boltzmann, 435
 - greedy, 434
- exposure problem, 316, 316
- extensional logic, *see* logic
- extensive-form game, 829–836

- history in, 829
 - of imperfect information, 307
 - terminal history in, 829
- factoredness, 428
- false-name bidding, 317
- false-name-proofness, 259
- feasible sets, 223
- feature-based comparison, 735
- finite action learning automata, 450
- FIPA, 56, 111, 602
 - ACL, 116
 - request interaction protocol, 111
- first-price auction, 306, 310–313, 319
- folk psychology, 769
- formal method, 642, 732
 - verification, 659
- formal specification
 - agent logics, 644
- formal verification, 659, 732
- Foundation of Intelligent Physical Agents, *see* FIPA
- frequency adjusted Q-learning, 450
- frequency of manipulability, 262
- functional system, 9
- functionally-accurate cooperation, 493
- fuzzy set, 384

- Gaia, 698, 699
- game
 - semantics, 190
- game theory, 146, 443, 448, 764, 786, 811
- games
 - Markov, 430, 432, 438
 - matrix, 443
 - repeated, 439
- GDL, 568
- generalised distributive law, 568
- generalized first-price auction (GFP), 313
- generalized second-price auction (GSP), 313, 314
- generalized Vickrey auction, *see* Groves mechanism
- generation, 458
- Gibbard-Satterthwaite theorem, 233
- global collective goal, 610
- GOAL, 781
- GOAL, 605
- goal, 594, 598, 700, 701, 705
 - achievement, 598
 - declarative, 598
 - long-term, 653
 - prioritized, 244
- goal model, 707
- GOLOG, 19, 666, 781
- goods, 249
- gradient ascent, 441
- grand coalition, 334–339, 345, 356, 373
- graph coloring problem, 554
- grid computing, xxxvi
- grounded semantics, 779
- group, 610
- group-strategyproofness, 235
- Groves mechanism, 297, 297, 298, 299, 301, 303, 308, 319

- hard constraint, 549
- higher-order representation, 672
- history of AOSE, 698
- Hoare logic, 653
- holonic manufacturing, 622, 702
- horizontal layering, 36

- i^* , 709
- I-POMDP, 514
- IDK, 727
- IIA, *see* independence of irrelevant alternatives
- IMPACT

- agent program, 665
- reasoning
 - probabilistic, 666
- IMPACT, 780
- impartial culture assumption, 263
- implementation
 - in Bayes–Nash equilibrium, 288
 - in dominant strategies, 287, 288, 290
 - in *ex post* equilibrium, 288
- implementation rule, 156
- implementation theory, 285
- imputation, 333, 343
- incentive compatibility in dominant strategies, 289
- incentive compatible, 195
- independence of irrelevant alternatives (axiom), 221, 224
- independence of the common utility pace (axiom), 252
- independent private value (IPV), 307, 308, 309, 319
- individual, 219
- individual rational, 149
- individual rationality, 301, 333, 343
 - ex interim*, 294, 304
 - ex post*, 295, 302, 316
- individually rational deal, 256
- indivisible issue, 159
- information attitude, 770
- INGENIAS, 698, 699, 730
- InstAL, 78
- institution, 73, 131, 701, 717
 - electronic, 611
- institutional modeling, 78
- integer partition, 353, 354, 357–359, 363
- integer program, 317
- integration testing, 729
- intelligence, 103
- intelligent agent, 3
- intended means, 595
- intensional logic, *see* logic
- intention, 117, 595, 653
- intention filter, 771
- intention logic, *see* logic, 770
- intention side effects, 771
- intentional stance, 769
- intentional system, 769
- intentions, 28, 774
- interaction protocol, *see* protocol, 712
- interaction structure, 70
- interaction testing, 729
- interleaved strategy, 665
- intermodal consistency, 777
- interoperation, 102
- interpreted system, 671, 672
- InteRRaP architecture, 40
- invocation condition, 774
- IRMA, 774
- irresoluteness, 239
- Isabelle, 660
- JACK, 727
- JADE, 123, 717, 727, 728, 730
 - behavior, 123
- JADEx, 601
- Japanese auction, 305, 305, 307, 308
- Java, 10
- Java Agent Development Framework,
see JADE
- joint action space, 426
- judgment aggregation, 258
- JUnit, 729
- KARO, 646
- Kemeny’s rule, 222, 231
- knowledge
 - de dicto*, 768
 - de re*, 768
- knowledge precondition, 768

- Knowledge Query and Manipulation Language, *see* KQML
- knowledge-producing action, 768
- KQML, 116
- Kripke model, *see* Kripke structure, 765
- Kripke semantics, 767
- Kripke structure, 647
- kth*-price auction, 306
- language, 761
 - implementation, 780–782
 - programming, 778
 - specification, 778–780
 - verification, 778, 782–785
- layered architecture, 36
- learnability, 428
- learning
 - automata, 449
 - automaton, 430
 - cross, 431
 - joint action, 439
 - problem, 425
- lenient
 - FAQ-learning, 450
 - Q-learning, 449
- level
 - abstraction, 613
 - basic, 613
 - interaction-mediation, 613
- leximin ordering, 252
- linear program, 317, 318
 - relaxation, 318
- linear temporal logic, 646, *see*
 - temporal logic, linear
- linear-time temporal logic, *see* logic, LTL
- link, 610
- locally envy free, 314
- logic, 761–799
 - alternating-time temporal logic, 786
 - ATL, 790–794
 - ATL, 786
 - BDI, 665, 773–777, 784
 - CL-PC, 791, 796–797
 - coalition, 786–789, 794
 - complexity, 762
 - computational tree, 766
 - control, 791
 - CTL, 766, 791
 - deontic, 785, 798
 - dynamic, 766
 - epistemic, 764
 - expressiveness, 762
 - extensional, 763
 - intensional, 763
 - intention, 770–773
 - LTL, 765, 766, 783
 - modal, 763, 766, 771, 779, 780, 782, 784, 786, 798, 799
 - of knowledge, 645
 - temporal, 782
- logic program
 - annotated, 666
- logic programming
 - abductive, 666
- LTL, *see* linear temporal logic
 - formulae, 649
- LTL*, *see* linear temporal logic
- MABLE, 784
- maintenance goals, 598
- majority rule, 220
- manipulability, 233
- manipulation, 237
- manipulation problem, 236
- marginal contribution net (MC-net), 349, 350, 364, 373
- Markov assumption, 26
- Markov decision process, 25

- Markov property, 433
- MAS-CommonKADS, 698, 699, 709
- MaSE, 698, 699, 727
- matching, 258
- Matching Pennies, 444, 817, 819
- MAUDE System, 675
- max-product, 568
- max-sum, 568
- maximin rule, 229
- maximum gain messaging, 567
- maximum likelihood approach, 222, 247
- maxmin fairness, 295, 296
- MB-DPOP, 564
- MCAPL, 677
- MCMAS, 674
- MDP, 432–434, 437, 512
- meaning, 118, 125, 128
 - message, 104
- means-ends reasoning, 28
- mechanism, 194, 287, 304–306, 308, 311, 313, 314, 316
 - Groves, *see* Groves mechanism
 - quasilinear, 292, 292, 293–297, 299, 306
 - direct, 293, 297, 299
 - Vickrey–Clarke–Groves (VCG), *see* Vickrey–Clarke–Groves (VCG) mechanism
- mechanism design, 232, 253, 258, 285, 285, 304, 306, 307
- mechanism implementation, 194
- median-rank dictator, 265
- median-voter rule, 234
- meeting scheduling, 552
- mental state, 764
- MESSAGE, 698, 699
- message, 102, 700
- message sequence chart, 111
- meta-model
 - of CARTAGO, 614
 - of EIS, 620
- MetateM, 19
- METATEM, 780, 781
- method engineering, 739
- methodology, 127, 696
- MGM, 567
- MGM-2, 572
- minimal covering set, 226
- mission, 610
- MOCHA, 791
- modal logic, *see* logic
 - KD, 654
 - KD45, 653
 - knowledge, 645
 - S5, 645, 654
 - semantics, *see* Kripke structure
- modal operator, *see* modality
- modality, 771, 772, 775, 776, 781, 784, 785, 787, 790, 791, 794, 795
 - cooperation, 787, 791, 794, 795
 - deontic, 781
 - temporal, 785
- model, 696
- model building, 780
- model checker, 643
- model checking, 661, 732, 784
 - agent programs, 676
 - automata-theoretic view, 661
 - complexity, *see* computational complexity, model checking
 - JAVA PATHFINDER, 663
 - on-the-fly, 661
- model-driven development, 728
- modular interpreted system, 671
- modular programming language, 602
- MOISE, 610
- Moise+, 736
- monotonicity (axiom), 262
- moving-knife procedure, 248

- MSC, *see* message sequence chart
 MTD, 514
 multiagent
 formulations, 437
 learning, 424, 460
 reinforcement learning, 432
 system, 448
 multiagent communication, 494
 multiagent control
 definition, 486
 multiagent organization, 51, 55
 multiagent planning
 continuous, 529
 decision-theoretic, 512
 definition, 486
 execution, 527
 hierarchical task network, 510
 multiagent plan coordination
 problem, 497
 partial global planning, 529
 partial-order causal-link, 501
 plan combination search, 499
 plan modification, 505
 plan monitoring, 527
 plan-space search, 499
 recovery/repair, 528
 state-space search, 498
 team-oriented, 510
 multiagent resource allocation, 247
 multiagent system, xxxv, 53
 relevance of, xxxvi
 variety, xxxvi
Multi-Agent Programming Contest,
 619, 620, 631
 multiple-election paradox, 243
 mutation, 446
 MYCIN, 13

 Nanson's rule, 230
 Nash equilibrium, 193
 Nash product, 252
 Nash-Q, 440
 necessary winner problem, 240
 negotiation, 143
 decision function, 161
 reputation, 410
 trade-off, 158
 trust, 410
 negotiation agenda, 156, 161
 endogenous, 160
 exogenous, 160
 optimal, 164
 neural network, 457
 neuro-evolutionary
 algorithms, 430
 approach, 457
 neutrality (axiom), 227
 no learning, 768
 no negative externalities, 301, 301, 302
 no single-agent effect, 302, 302, 303
 non-determinism, 6
 non-dictatorship (axiom), 221, 224,
 233
 non-imposition (axiom), 233, 238, 261
 norm, 73, 132, 410, 596, 610, 701
 reputation, 410
 social, 596
 trust, 410
 normal-form game, 812–826
 normative attitude, 770
 normative structure, 71
 notation, 696
 NuSMV, 677

 O-MaSE, 698, 705, 709
 obligation, 77, 596, 610, 770
 occurrent trust, 386
 OCL, 699, 734
 online auction, 309
 ontology, 709
 open system, 717
 open-outcry auction, 306

- OperA, 65, 736
- operator, 765, 772, 775
 - ability, 786
 - branching-time, 766
 - dynamic, 767
 - epistemic, 767
 - for until, 765
 - K_i , 764, 794
 - linear-time, 766
 - modal, 771, 779, 786, 787
 - necessity, 765, 767
 - temporal, 766
 - temporal precedence, 772
 - tense, 766
- OperettA, 611
- 2OPL, 611
- opportunity, 767
- OR-language, 250
- orchestration, 110
- order statistic, 312, 312
- ordinal preference, 220
- organization, 51, 73, 131, 596
 - as agent, 59
 - as institution, 58
 - as structure, 56
 - design, 491
 - emergent, 87
 - meta-level, 530
 - ORGANIZATIONSEARCH
 - algorithm, 492
 - reputation, 411
 - structuring, 490
 - trust, 411
- organization concept, 64
- organization modeling, 65
- organizational adaptation, 86
- organizations, 609
- overcommitment, 773
- P-Goal, 772
- package deal procedure, 156
- Pareto optimality, 444
 - axiom, 221, 224, 255
- partial global planning, 529
- partition function game, 331
- PASSI, 698, 699, 709, 730
- passive service provider, 12
- path integration, 455
- pattern, 124
 - enactment, 125
- payment rule, 288, 293, 299
- payoff function, 812, 813, 829
- payoff matrix, 813
- PDT, 727, 730
- peer-to-peer computing, xxxvi
- percept, 700
- perfect information, 794
- perfect recall, 767
- permission, 77, 610, 770
- persistent goal, 772
- pervasive computing, xxxvi
- Petri net, 109, 717
- pheromone, 454
- Pigou-Dalton principle (axiom), 253
- pivot mechanism, *see* Groves
 - mechanism
- plan, 595, 599
 - body of a, 599
 - context of a, 599
 - trigger of a, 599
- plan library, 774
- planning
 - partial order, 500
- planning domain definition language, 605
- plurality rule, 214, 228
- plurality with runoff, 261
- policy, 26, 432, 515
 - finite-state controller, 515, 522
 - stochastic, 516
 - tree, 515, 519

- POMDP, 512
 - decentralized, 513
- position auction, 312–314, 320
 - generalized first-price, *see*
 - generalized first-price auction
 - generalized second-price, *see*
 - generalized second-price auction
- positional scoring rule, *see* scoring rule
- possible winner problem, 240
- postcondition, 9
- power, 77
- practical reasoning, 28, 774
- precondition, 6, 9
- preference, 220, *see also* social
 - preference function, 286, 291, 292, 307
 - cardinal preference, 220
 - ordinal preference, 220
 - quasilinear preference, 253
 - single-caved preference, 262
 - value-restricted preference, 235
- preference aggregation, 219
- preference profile, 291
- preference representation language, 243, 249
- preferences, 286, 288, 290–292
- price of anarchy, 296
 - minimization, 296
- prioritized goal, 244
- Prisoner's Dilemma, 147, 813, 814
- privacy, 293
- pro-attitude, 770
- proactiveness, 8
- proactive, 700
- probability distribution, 384
- procedural knowledge, 19
- procedural reasoning system, 774
- process, 696
- process algebra, 109
- profile, 220, *see also* profile of preferences
- profile of preferences, 220
- program
 - environment, 628
- program verification, 662
- program-based preference representation, 251
- programming
 - agent-oriented, 621
 - organization-oriented, 621
- PROLOG, 774
- Prometheus, 698, 699, 730
- Prometheus design tool, 730
- proof system, 643
- proportionality, 248
- propositional logic
 - dynamic, 652
- Protégé, 709
- protocol, 101, 130, 700
 - aggregation, 129
 - argumentation protocol, 185
 - foreign exchange, 102
 - GDSN, 102
 - HITSP, 102
 - HL7, 102
 - refinement, 129
 - requirements, 102
 - RosettaNet, 102
 - specifications, 102
 - standard protocol, 102
 - supply chain, 102
 - TWIST, 102
- proxy bidding, 309
- PRS, 774
- pseudonymous bidding, 317
- PTK, 728
- PVS, 666
- Q-learning, 427, 435

- quasilinear mechanism, *see*
 - mechanism, quasilinear
- quasilinear preference, 253, 291, 292, 307
- quasilinear utility function, 291, 303, 304
- random dictator, 238
- ranked pairs (voting rule), 230
- rating, 405
- rational agent, 764
- rational balance, 30, 770
- rationality, 301
- reactive, 700
- reactive module, 672
- REACTIVE MODULES, 791
- reactiveness, 8
- realizability, 778
- reasoning
 - probabilistic, 666
- reasoning cycle, 600
- reasoning rule, 598
- regimentation, 596
- regret minimization, 449, 450
- regulation, 60
- reinforcement (axiom), 228
- reinforcement learning, 425, 427, 430, 432, 435, 443
- relaxation method, 318
- reliability, 387
- replicator
 - dynamics, 446
 - equations, 443, 460
- representation, 458
- reputation, 388, 390, 396
 - argumentation, 408
 - as source of trust, 404
 - collusion, 406
 - communicated reputation, 400
 - dynamic personality, 406
 - inherited reputation, 400
 - pitfalls of using reputation, 405
 - reputation model, 402
 - reputation value, 382
 - social order, 405
 - whitewashing, 406
- reputation evaluation, 398
- reputation representation, 383
- requirement, 704
- resoluteness, 227
- resource allocation, 247
- revelation principle, 195, 232, 288–291, 319
- revenue maximization, 295
- reward, 25
 - cumulative, 433
 - difference, 429, 465
 - immediate, 434
 - local, 429
 - structure, 427, 465
 - system, 428
- risk attitude, 310
 - risk neutral, 293, 310, 311
- risk neutrality, 292
- ROADMAP, 736
- role, 102, 110, 130, 610, 705
- role skeleton, *see* role
- row player, 813
- run-time verification, 663
- RUP, 699
- safety case, 733
- Santa Claus problem, 252
- SARSA, 435
- scenario, 705
- SCF, *see* social choice function
- scheme, 610
- scoring rule, 227
- SDS, *see* social decision scheme
- Seagent, 730
- sealed-bid auction, 305, 306, 306, 307, 308, 310, 311

- second-price auction, 306, 308–311, 319
- selection, 446, 458
- self-interested, 192
- semantics, 761, 763, 765
 - fixed-point, 666
- semi-supervised learning, 432
- sensitivity, 428
- sequential decision making, 433
- sequential voting, 242, 245
- service engagement, 118
- service provider, 12
- service-oriented computing, xxxvii
- set packing problem, 318
- shill bid, 317
- simple exchange, 303, 303
- simultaneous ascending auction, 316
- single transferable vote, 214, 231
- single-caved preference, 262
- single-good auction, 304
- single-peaked preferences, 234
- single-sided auction, 302, 305
- situated, 700
- situated automata, 20, 782
- situation, 776
- situation calculus, 19, 666
- Slater's rule, 222, 231
- social, 700
- social ability, 8
- social choice, 789
- social choice function, 194, 223, 287–291, 297, 298, 302
- social commitment, *see* commitment, 701
- social control, 382
- social decision scheme, 238
- social laws, 489
- social level, 596
- social plan, 596
- social preference function, 222
- social state, 128
- social structure, 68
- social welfare function, 219, 220
- social welfare maximization, 294
- social welfare ordering, 252
- SODA, 736
- soft constraint, 549
- soft security, 382
- software daemon, 8
- software evolution, 734
- software maintenance, 734
- solution concept, 287, 329, 335, 337, 338, 341–343, 349, 814
 - Banzhaf index, 337, 338, 342, 344, 345
 - bargaining set, 342, 343
 - core, 338–343, 345, 346, 348–350, 372–374
 - kernel, 342, 343
 - least core, 341–343, 346, 349
 - nucleolus, 342, 343, 346, 349, 374
 - Shapley value, 335–337, 341, 342, 344, 345, 348, 350, 372, 373
- specification, 14, 761
- speech act, *see* communicative act
- SPEM, 739
- sphere of influence, 619
- SPIN System, 662, 675, 677
- sponsored search auction, *see* position auction
- SQL, 605
- standardization, 738
- state machine, 109
- state space, 426
- statechart, 109
- stigmergy, 453
- strategic structure of environment, 764
- strategy, 192, 791, 793, 812
 - in an extensive-form game, 831
 - mixed, 818

- pure, 813, 818
- strictly dominant, 814, 827
- strictly dominated, 815
- support of (mixed), 818
- weakly dominant, 814, 827
- weakly dominated, 815
- strategy profile, 192
 - mixed, 818
- strategyproof, 195, *see* incentive compatibility in dominant strategies
- strategyproofness, 233
- STRATUM, 736
- strict budget balance, *see* budget balance
- strict Pareto efficiency, 294
- strong retentiveness, 226
- structural subset, 776
- structural superset, 776
- STV, *see* single transferable vote
- subadditive valuation function, *see* substitutes
- subgame, 832
- substitutes, 315
 - partial, 315
 - strict, 315
- subsumption architecture, 21
- subsumption hierarchy, 21
- subworld, 776
- sum-product, 568
- superadditive valuation function, *see* complementarity
- swarm intelligence, 430, 451
- SWF, *see* social welfare function
- SWO, *see* social welfare ordering
- Sybil attack, 407
- symbolic AI, 14
- synergy coalition group (SCG), 350
- system
 - dynamics, 425
 - performance, 462
 - system overview diagram, 716
 - system testing, 729
- TAOM4E, 727
- target tracking, 553
- task accomplishing behavior, 21
- temporal difference, 427, 435
- temporal execution
 - CHRONOLOG, 658
 - METATEM, 658, 665
 - multiagent systems, 659
 - TEMPLOG, 658
- temporal logic, 19, *see* logic knowledge, 645
 - linear, 648
- temporal logic programming, *see* temporal execution
- temporal specification
 - deadlock freedom, 648
 - direct execution, *see* temporal execution
 - fairness, 648
 - implementation, 656
 - liveness, 648
 - reachability, 648
 - refinement, 657
 - safety, 648
 - synthesis, 657
- testing, 729
- theorem prover
 - semi-automated, 660
- theorem proving, 14, 732
- theory of mind, 769
- threat strategy, 150
- time points, 775
- top cycle, 225
- total unimodularity, 318
- TOURINGMACHINE architecture, 38
- tournament equilibrium set, 226
- tractability, 295, 296

- trade-off, 158
- transferable utility, 292
- transferable utility game, 330, 331
- transition relation
 - serial, 647
- transition system
 - labeled, 650
 - unlabeled, 647
- trigger, *see* plan
- Tropos, 699, 730
- trust
 - argumentation, 408
 - definition, 388
 - trust value, 382
- trust belief, 395
- trust decision, 388, 390
- trust evaluation, 388
 - image, 388
- trust representation, 383
- trustee, 388
- trustor, 388
- truth-telling, 297
- truthful, 288, 289, 289, 290, 291, 294, 297–299, 301, 302, 308, 309, 314, 316, 319
- turn-based synchronous system, 792
- type, 286–288, 290, 291, 293, 294, 307, 311
- ubiquitous computing, xxxvi
- UML, 699, 709
- UML sequence diagram, 109
- uncovered set, 226
- undercommitment, 773
- uniform strategy, 796
- unit testing, 729
- utilitarianism, 251
- utility function, 287, 290–293, 297, 303, 304, 306, 307, 310
- utility vector, 251
- valuation, 293, 305, 307, 308, 310–313, 315–317
- valuation function, 249, 297, 315
 - non-additive, 315
 - subadditive, *see* substitutes
 - superadditive, *see* complementarity
- value
 - independent private, *see* independent private value
- value function, 433
- value iteration, 26
- value-restricted preference, 235
- VCG mechanism, *see* Vickrey–Clarke–Groves mechanism
- verification, 761, 782
 - algorithmic, 660
 - axiomatic, 782–784
 - formal verification, 659
 - model checking, 783–786
 - run-time, 663
 - semantic, 783–786
- vertical layering, 36
- veto player, 340, 345
- veto rule, *see* anti-plurality rule
- Vickrey auction, *see* second-price auction
- Vickrey–Clarke–Groves (VCG)
 - mechanism, 299, 299, 301–303, 308, 314–316, 319
- voter, 219
- voting rule, 227
- WARP, *see* weak axiom of revealed preference
- weak axiom of revealed preference, 224
- weak budget balance, *see* budget balance, weak
- weak Condorcet winner, 264
- weighted goals, 249